runway Documentation

Release 2.6.12-post.33

Onica Group

Jan 10, 2024

RUNWAY

Runway is a lightweight wrapper around infrastructure deployment (e.g. CloudFormation, Terraform, Serverless) & linting (e.g. yamllint) tools to ease management of per-environment configs & deployment.

CHAPTER

ONE

INSTALLATION

To enable Runway to conform to our users' varying use cases, we have made it available via three different install methods - *cURL*, *npm*, and *pip*.

1.1 cURL

Arguably the easiest way to install Runway is by using curl. Use one of the endpoints below to download a single-binary executable version of Runway based on your operating system.

Operating System	Endpoint
Linux	https://oni.ca/runway/latest/linux
macOS	https://oni.ca/runway/latest/osx
Windows	https://oni.ca/runway/latest/windows

Linux

```
curl -L https://oni.ca/runway/latest/linux -o runway
```

macOS

```
curl -L https://oni.ca/runway/latest/osx -o runway
```

Windows

```
Invoke-WebRequest -Uri "https://oni.ca/runway/latest/windows" -OutFile runway
```

Note: To install a specific version of Runway, you can replace `latest` with a version number.

Usage

To use the single-binary, run it directly as shown below. Please note that after download, you may need to adjust the permissions before it can be executed. (eg. Linux/macOS:`chmod +x runway`)

```
$ ./runway deploy
```

Suggested use: CloudFormation or Terraform projects

1.2 npm

Runway is published on npm as `@onica/runway`. It currently contains binaries to support macOS, Ubuntu, and Windows.

While Runway can be installed globally like any other npm package, we strongly recommend using it per-project as a dev dependency. See *Why Version Lock Per-Project* for more info regarding this suggestion.

```
$ npm i -D @onica/runway
```

Usage

```
$ npx runway deploy
```

Suggested use: Serverless or AWS CDK projects

1.3 pip

Runway runs on Python 2.7 and Python 3.5+.

Runway is hosted on PyPI as the package named `runway`. It can be installed like any other Python package, but we instead strongly recommend using it per-project with poetry. See *Why Version Lock Per-Project* for more info regarding this suggestion.

Suggested use: Python projects

poetry

```
poetry add runway
```

pip

```
pip install --user runway
# or (depending on how Python was installed)
pip install runway
```

Usage

poetry

```
poetry run runway --help
```

pip

```
runway --help
```

1.4 Why Version Lock Per-Project

Locking the version of Runway per-project will allow you to:

- Specify the version(s) of Runway compatible with your deployments config
- Ensure Runway executions are performed with the same version (regardless of where/when they occur – avoids the dreaded "works on my machine")

CHAPTER

TWO

UPGRADES

Contents

During a Runway upgrade (especially coming from a `0.x` version) you may required to make changes to your configuration or modules. This page will describe common issues when upgrading and how to resolve them.

2.1 Updating the Runway Config File

You may need to update your Runway config file structure depending on how old of a Runway version you are upgrading from. Some config keys have changed in spelling, may make use of _ instead of -, or have different functionality altogether. For example, compare this old config file:

```yaml
deployments:
  - modules:
      - module-1.cfn
      - module-2.cfn
      - module-3.cfn
    regions:
      - us-east-1
    account-id:
      prod: 123412341234
    assume-role:
      post_deploy_env_revert: true
      prod: arn:aws:iam::123412341234:role/my-deployment-role
```

(continues on next page)

(continued from previous page)

```
    environments:
      prod:
        namespace: my-account
        environment: prod
```

To this newer version:

```
variables:
  account_id:
    prod: 123412341234
  assume_role:
    prod: arn:aws:iam::123412341234:role/my-deployment-role
  parameters:
    prod:
      namespace: my-account
      environment: prod

deployments:
  - modules:
      - module-1.cfn
      - module-2.cfn
      - module-3.cfn
    regions:
      - us-east-1
    account_id: ${var account_id.${env DEPLOY_ENVIRONMENT}}
    assume_role:
      arn: ${var assume_role.${env DEPLOY_ENVIRONMENT}}
      post_deploy_env_revert: true
    environments:
      prod: true
    parameters: ${var parameters.${env DEPLOY_ENVIRONMENT}}
```

In the above example, we've taken advantage of Runway's `variables` key and we dynamically reference it based on our `DEPLOY_ENVIRONMENT`. We also updated `account-id` and `assume-role` to `account_id` and `assume_role`.

Runway is very flexible about how you can structure your config, the above is only an example of one way to adjust it. Just keep in mind while upgrading that if you receive errors as soon as you start a `runway` command, it is likely due to a config file error or no longer supported directive.

2.2 Migration from Stacker to CFNgin

Older versions of Runway used Stacker, which was then forked and included into the Runway project as CFNgin. This causes a few concerns while migrating older deployments.

2.2.1 Update References to Stacker

See *Migrating from Stacker* for info on converting your older Stacker modules into CFNgin compatible versions.

2.2.2 Migration of Stacker Blueprints

In some environments, you may see usage of Stacker "Blueprints". These are Python scripts leveraging Troposphere to generate CloudFormation templates programmatically. While these can be incredibly powerful, they also come with a Python experience dependency and are prone to breaking due to AWS or Runway changes. In older deployments if the blueprint contains references to `stacker` it will also need to be updated to use the new `cfngin` library after a Runway upgrade, as described in *Migrating from Stacker*.

In most cases it is easiest to:

1. Navigate to the AWS CloudFormation Console,
2. find the stack that was deployed using the blueprint,
3. copy its CloudFormation template data (optionally converting it to YAML on the way); and,
4. convert the deployment in Runway to use that static template so you can eliminate the blueprint.

This process leaves you with a much more simple to manage static template.

2.2.3 A Note on Tagging in Stacker Modules

stacker_namespace / cfngin_namespace tag

If a Stacker/CFNgin deployment doesn't have a `tags` key defined, a default value is used:

Stacker:

```
stacker_namespace: ${namespace}
```

CFNgin:

```
cfngin_namespace: ${namespace}
```

Because of this if you are upgrading a Stacker module without the `tags` key defined, you'll see Runway attempting to adjust the tags on every resource in the module. This is because it is updating the default `stacker_namespace` tag to a `cfngin_namespace` tag. If you'd like to prevent this behavior, you can add a `tags` key as follows:

```
tags:
  stacker_namespace: ${namespace}
```

The above usage will cause CFNgin to keep the old `stacker_namespace` tag with its original value, eliminating the need for changes to tags on resources.

Int to String Errors

When defining a `tags` key directly onto a CFNgin stack definition (not the top level `tags` key in the CFNgin config file), you may see an error regarding using an `int` instead of a `string`. For instance:

```
# This may return a "must be of type string" error
my-stack-definition:
  template_path: ./my-templates/my-cloudformation-template.yaml
  tags:
    CostCenter: 1234
```

This can be resolved by enclosing your numerical value in quotes:

```
# This may return a "must be of type string" error
my-stack-definition:
  template_path: ./my-templates/my-cloudformation-template.yaml
  tags:
    CostCenter: "1234"
```

2.3 Updates to Lookups

Some lookup usage may have changed slightly. Here's some examples:

```
# This generates a deprecation warning in newer Runway versions
VpcId: ${rxref vpc::VpcId}

# This is the new usage
VpcId: ${rxref vpc.VpcId}
```

```
# This generates an unknown lookup error
SlackUrl: ${ssmstore us-east-1@/devops/slack_hook}

# This is the new usage
SlackUrl: ${ssm /devops/slack_hook}
```

CHAPTER

THREE

GETTING STARTED GUIDE

Contents

3.1 Basic Concepts

Welcome to Runway! To get a basic understanding of Runway, we have listed out the key concepts below that you will need to get started with deploying your first module.

3.1.1 Runway Config File

The Runway config file is usually stored at the root of a project repo. It defines the modules that will be managed by Runway.

3.1.2 Deployment

A deployment contains a list of modules and options for all the modules in the deployment. A Runway config file can contain multiple deployments and a deployment can contain multiple modules.

3.1.3 Module

A module is a directory containing a single infrastructure as code tool configuration of an application, a component, or some infrastructure (eg. a set of CloudFormation templates). It is defined in a deployment by path. Modules can also contain granular options that only pertain to it.

3.1.4 Deploy Environment

Deploy environments are used for selecting the options/variables/parameters to be used with each modules <module>. They can be defined by the name of a directory (if its not a git repo), git branch, or environment variable (`DEPLOY_ENVIRONMENT`). Standard environments would be something like prod, dev, and test.

No matter how the environment is determined, the name is made available to be consumed by your modules as the `DEPLOY_ENVIRONMENT` environment variable.

3.2 Deploying Your First Module

1. Create a directory for our project and change directory into the new directory.

```
$ mkdir sample-project && cd sample-project
```

2. Initialize the the new directory as a git repository and checkout branch **ENV-dev**. This will give us an environment of **dev**.

```
$ git init && git checkout -b ENV-dev
```

3. Download Runway using *curl*. Be sure to use the endpoint that corresponds to your operating system. Then, change the downloaded file's permissions to allow execution.

   Linux

```
$ curl -L https://oni.ca/runway/latest/linux -o runway
$ chmod +x runway
```

   macOS

```
$ curl -L https://oni.ca/runway/latest/osx -o runway
$ chmod +x runway
```

   Windows

```
Invoke-WebRequest -Uri "https://oni.ca/runway/latest/windows" -OutFile runway
```

4. Use Runway to generate a sample module using the *gen-sample* command. This will give us a preformatted module<runway-module that is ready to be deployed after we change a few variables. To read more about the directory structure, see *Repo Structure*.

```
$ ./runway gen-sample cfn
```

5. To finish configuring our CloudFormation module, lets open the `dev-us-east-1.env` file that was created in `sampleapp.cfn/`. Here is where we will define values for our stacks that will be deployed as part of the **dev** environment in the **us-east-1** region. Replace the place holder values in this file with your own information. It is important that the `cfngin_bucket_name` value is globally unique for this example as it will be used to create a new S3 bucket.

Listing 1: dev-us-east-1.env

```
namespace: onica-dev
customer: onica
environment: dev
region: us-east-1
# The CFNgin bucket is used for CFN template uploads to AWS
cfngin_bucket_name: cfngin-onica-us-east-1
```

6. With the module ready to deploy, now we need to create our Runway config file. To do this, use the *new* command to generate a sample file at the root of the project repo.

```
$ ./runway new
```

Listing 2: runway.yml

```
---
# See full syntax at https://docs.onica.com/projects/runway
deployments:
  - modules:
      - nameofmyfirstmodulefolder
      - nameofmysecondmodulefolder
      # - etc...
  regions:
    - us-east-1
```

7. Now, we need to modify the `runway.yml` file that was just created to tell it where the module is located that we want it to deploy and what regions it will be deployed to. Each module type has their own configuration options which are described in more detail in the *Module Configurations* section but, for this example we are only concerned with the *CloudFormation module configuration*.

Listing 3: runway.yml

```
---
# See full syntax at https://docs.onica.com/projects/runway
deployments:
  - modules:
      - sampleapp.cfn
  regions:
    - us-east-1
```

8. Before we deploy, it is always a good idea to know how the module will impact the currently deployed infrastructure in your AWS account. This is less of a concern for net-new infrastructure as it is when making modifications. But, for this example, lets run the *plan* command to see what is about to happen.

```
$ ./runway plan
```

9. We are finally ready to deploy! Use the *deploy* command to deploy our module.

```
$ ./runway deploy
```

We have only scratched the surface with what is possible in this example. Proceed below to find out how to delete the module we just deployed or, review the pages linked throughout this section to learn more about what we have done to this point before continuing.

3.3 Deleting Your First Module

From the root of the project directory we created in *Deploying Your First Module* we only need to run the *destroy* command to remove what we have deployed.

```
$ ./runway destroy
```

3.4 Execution Without A TTY (non-interactive)

Runway allows you to set an environment variable to allow execution without a TTY or if STDIN is closed. This allows users to execute Runway *deployments* in their CI/CD infrastructure as code deployment systems avoiding the `EOF when reading a line` error message. In order to execute runway without a TTY, set the `CI` environment variable before your `runway [deploy|destroy]` execution.

Important: Executing Runway in this way will cause Runway to perform updates in your environment without prompt. Use with caution.

CHAPTER

FOUR

QUICKSTART GUIDES

4.1 CloudFormation Quickstart

1. Prepare the project directory. See *Repo Structure* for more details.

```
$ mkdir my-app && cd my-app
$ git init && git checkout -b ENV-dev
```

2. Download/install Runway. To see available install methods, see *Installation*.
3. Use Runway to *generate a sample* CloudFormation module<runway-module, edit the values in the environment file, and create a Runway config file to use the module.

   POSIX

```
$ runway gen-sample cfn
$ sed -i -e "s/CUSTOMERNAMEHERE/mydemo/g; s/ENVIRONMENTNAMEHERE/dev/g; s/cfngin-/
↪cfngin-$(uuidgen|tr "[:upper:]" "[:lower:]")-/g" sampleapp.cfn/dev-us-east-1.env
$ cat <<EOF >> runway.yml
---
# Full syntax at https://github.com/onicagroup/runway
deployments:
  - modules:
      - sampleapp.cfn
    regions:
      - us-east-1
EOF
```

   Windows

```
$ runway gen-sample cfn
$ (Get-Content sampleapp.cfn\dev-us-east-1.env).replace('CUSTOMERNAMEHERE', 'mydemo
↪') | Set-Content sampleapp.cfn\dev-us-east-1.env
$ (Get-Content sampleapp.cfn\dev-us-east-1.env).replace('ENVIRONMENTNAMEHERE', 'dev
↪') | Set-Content sampleapp.cfn\dev-us-east-1.env
$ (Get-Content sampleapp.cfn\dev-us-east-1.env).replace('cfngin-', 'cfngin-' +␣
↪[guid]::NewGuid() + '-') | Set-Content sampleapp.cfn\dev-us-east-1.env
$ $RunwayTemplate = @"
---
# Full syntax at https://github.com/onicagroup/runway
deployments:
  - modules:
```

(continues on next page)

(continued from previous page)

```
      - sampleapp.cfn
    regions:
      - us-east-1
"@
$RunwayTemplate | Out-File -FilePath runway.yml -Encoding ASCII
```

4. *Deploy* the stack.

```
$ runway deploy
```

Now our stack is available at `mydemo-dev-sampleapp`, e.g.:

```
$ aws cloudformation describe-stack-resources --region us-east-1 --stack-name mydemo-dev-
↪sampleapp
```

4.2 Conduit (Serverless & CloudFront) Quickstart

The Medium.com-clone "RealWorld" demo app named Conduit provides a simple demonstration of using Runway to deploy a Serverless Framework backend with an Angular frontend.

Contents

4.2.1 Prerequisites

- An AWS account, and configured terminal environment for interacting with it with an admin role.
- The following installed tools:
  - awscli
  - git (Available out of the box on macOS)
  - npm
  - python
  - yarn

4.2.2 Setup

1. Prepare the project directory. See *Repo Structure* for more details.

```
$ mkdir conduit
$ cd conduit
$ git init
$ git checkout -b ENV-dev
```

2. Download/install Runway. To see available install methods, see *Installation*.
3. Download the source files.

   POSIX

```
$ curl -O https://codeload.github.com/anishkny/realworld-dynamodb-lambda/zip/v1.0.0
$ unzip v1.0.0
$ rm v1.0.0
$ mv realworld-dynamodb-lambda-1.0.0 backend && cd backend
$ sed -i '/package-lock\.json/d' .gitignore
$ echo '.dynamodb' >> .gitignore
$ npm install
$ cd ..
$ curl -O https://codeload.github.com/gothinkster/angular-realworld-example-app/zip/
↪35a66d144d8def340278cd55080d5c745714aca4
$ unzip 35a66d144d8def340278cd55080d5c745714aca4
$ rm 35a66d144d8def340278cd55080d5c745714aca4
$ mv angular-realworld-example-app-35a66d144d8def340278cd55080d5c745714aca4␣
↪frontend && cd frontend
$ mkdir scripts
$ cd scripts && { curl -O https://raw.githubusercontent.com/onicagroup/runway/
↪master/quickstarts/conduit/build.js ; cd -; }
$ sed -i 's/^\s*"build":\s.*$/    "build": "node scripts\/build",/' package.json
$ sed -i 's/^\s*"rxjs":\s.*$/    "rxjs": "~6.3.3",/' package.json
$ npm install
$ curl -O https://raw.githubusercontent.com/onicagroup/runway/master/quickstarts/
↪conduit/update_env_endpoint.py
$ cd ..
$ curl -O https://raw.githubusercontent.com/onicagroup/runway/master/quickstarts/
↪conduit/runway.yml
```

   Windows

```
$ [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
$ Invoke-WebRequest "https://codeload.github.com/anishkny/realworld-dynamodb-lambda/
↪zip/v1.0.0" -OutFile v1.0.0.zip
$ Expand-Archive v1.0.0.zip .
$ Remove-Item v1.0.0.zip -Force
$ Rename-Item realworld-dynamodb-lambda-1.0.0 backend && cd backend
$ (gc .\.gitignore -raw).Replace("package-lock.json`n", "") | sc .\.gitignore
$ ".dynamodb`n" | Out-File .\.gitignore -Append -Encoding UTF8
$ (gc .\package.json) -replace "dynamodb install .*$", "dynamodb install `"`" | Out-
↪File .\package.json -Force -Encoding UTF8
$ npm install
$ cd ..
```

(continues on next page)

(continued from previous page)

```
$ Invoke-WebRequest "https://codeload.github.com/gothinkster/angular-realworld-
↪example-app/zip/35a66d144d8def340278cd55080d5c745714aca4" -OutFile␣
↪35a66d144d8def340278cd55080d5c745714aca4.zip
$ Expand-Archive 35a66d144d8def340278cd55080d5c745714aca4.zip .
$ Remove-Item 35a66d144d8def340278cd55080d5c745714aca4.zip -Force
$ Rename-Item angular-realworld-example-app-
↪35a66d144d8def340278cd55080d5c745714aca4 frontend && cd frontend
$ (gc .\package.json -raw).Replace('"rxjs": "^6.2.1",', '"rxjs": "~6.3.3",')␣
↪| sc .\package.json
$ mkdir scripts
$ Invoke-WebRequest "https://raw.githubusercontent.com/onicagroup/runway/master/
↪quickstarts/conduit/build.js" -OutFile scripts/build.js
$ (gc .\package.json) -replace "^\s*""build"":\s.*$", "    ""build"": ""node␣
↪scripts/build"","" | Out-File .\package.json -Force -Encoding UTF8
$ npm install
$ Invoke-WebRequest "https://raw.githubusercontent.com/onicagroup/runway/master/
↪quickstarts/conduit/update_env_endpoint.py" -OutFile update_env_endpoint.py
$ cd ..
$ Invoke-WebRequest "https://raw.githubusercontent.com/onicagroup/runway/master/
↪quickstarts/conduit/pyproject.toml" -OutFile pyproject.toml
$ Invoke-WebRequest "https://raw.githubusercontent.com/onicagroup/runway/master/
↪quickstarts/conduit/runway.yml" -OutFile runway.yml
```

4.2.3 Deploying

Execute `runway deploy`, enter `all` (to deploy the backend followed by the frontend). Deployment will take some time (mostly waiting for the CloudFront distribution to stabilize).

The CloudFront domain at which the site can be reached will be displayed near the last lines of output once deployment is complete, e.g.:

```
staticsite: sync & CF invalidation of E17B5JWPMTX5Z8 (domain ddy1q4je03d7u.cloudfront.
↪net) complete
```

4.2.4 Teardown

Execute `runway destroy`, enter `all`.

The backend DynamoDB tables will still be retained after the destroy is complete. They must be deleted separately.

POSIX

```
for i in realworld-dev-articles realworld-dev-comments realworld-dev-users; do aws␣
↪dynamodb delete-table --region us-east-1 --table-name $i; done
```

Windows

```
foreach($table in @("realworld-dev-articles", "realworld-dev-comments", "realworld-dev-
↪users"))
{
  CMD /C "aws dynamodb delete-table --region us-east-1 --table-name $table"
}
```

4.2.5 Next Steps / Additional Notes

The serverless-plugin-export-endpoints plugin is a good alternative to the custom `update_env_endpoint.py` script used above to update the environment file.

4.2.6 Permissions

The specific IAM permissions required to manage the resources in this demo are as follows

```
# CloudFormation
- cloudformation:CreateStack
- cloudformation:DeleteStack
- cloudformation:CreateChangeSet
- cloudformation:DescribeChangeSet
- cloudformation:DeleteChangeSet
- cloudformation:DescribeStackResource
- cloudformation:DescribeStackResources
- cloudformation:DescribeStacks
- cloudformation:DescribeStackEvents
- cloudformation:GetTemplate
- cloudformation:UpdateStack
- cloudformation:ExecuteChangeSet
- cloudformation:ValidateTemplate
# Serverless
- apigateway:GET
- apigateway:DELETE
- apigateway:POST
- apigateway:PUT
- lambda:AddPermission
- lambda:CreateAlias
- lambda:CreateFunction
- lambda:DeleteAlias
- lambda:DeleteFunction
- lambda:GetFunction
- lambda:GetFunctionConfiguration
- lambda:ListVersionsByFunction
- lambda:PublishVersion
- lambda:UpdateAlias
- lambda:UpdateFunctionCode
- lambda:UpdateFunctionConfiguration
- iam:CreateRole
- iam:DeleteRole
- iam:DeleteRolePolicy
- iam:GetRole
- iam:PassRole
- iam:PutRolePolicy
- logs:CreateLogGroup
- logs:DeleteLogGroup
- logs:DescribeLogGroups
- s3:CreateBucket
- s3:DeleteBucket
- s3:DeleteBucketPolicy
```

(continues on next page)

(continued from previous page)

```
- s3:DeleteObject
- s3:DeleteObjectVersion
- s3:GetObjectVersion
- s3:ListBucket
- s3:ListBucketVersions
- s3:PutBucketVersioning
- s3:PutBucketPolicy
- s3:PutLifecycleConfiguration
# Frontend
- cloudfront:CreateCloudFrontOriginAccessIdentity
- cloudfront:CreateDistribution
- cloudfront:CreateInvalidation
- cloudfront:DeleteCloudFrontOriginAccessIdentity
- cloudfront:DeleteDistribution
- cloudfront:GetCloudFrontOriginAccessIdentity
- cloudfront:GetCloudFrontOriginAccessIdentityConfig
- cloudfront:GetDistribution
- cloudfront:GetDistributionConfig
- cloudfront:GetInvalidation
- cloudfront:ListDistributions
- cloudfront:TagResource
- cloudfront:UntagResource
- cloudfront:UpdateCloudFrontOriginAccessIdentity
- cloudfront:UpdateDistribution
- s3:DeleteBucketWebsite
- s3:GetBucketAcl
- s3:GetObject
- s3:PutBucketAcl
- s3:GetBucketWebsite
- s3:PutBucketWebsite
- s3:PutObject
- ssm:GetParameter
- ssm:PutParameter
# Backend
- dynamodb:CreateTable
- dynamodb:DeleteTable
- dynamodb:DescribeTable
- dynamodb:TagResource
- dynamodb:UntagResource
- dynamodb:UpdateTable
```

4.3 Other Ways to Use Runway

While we recommend using one of the install methods outlined in the *Installation* section, we realize that these may not be an option for some so we have provided a *CloudFormation* template for spinning up a deploy environment in AWS and a *Docker* image/Dockerfile that can be used to run Runway.

Contents

4.3.1 CloudFormation

This CloudFormation template is probably the easiest and quickest way to go from "zero to Runway" as it allows for using an IAM Role eliminate the need to configure API keys. The template will deploy your preference of Linux or Windows Runway host. Windows Runway host includes Visual Studio Code, which some users may find easier for manipulating Runway config files.

4.3.2 Docker

Docker users can build their own Docker image to run a local Runway container or modify this Dockerfile to build a Runway image to suit specific needs.

4.4 Private Static Site (*Auth@Edge*) Quickstart

The Runway built-in sample generation of a basic React app will be used as a simple demonstration of creating an authentication backed single page application.

Contents

4.4.1 Prerequisites

- An AWS account, and configured terminal environment for interacting with it with an admin role.
- The following installed tools:
  - git (Available out of the box on macOS)
  - npm

4.4.2 Setup

Project Setup

1. Download/install Runway. To see available install methods, see *Installation*.
2. From a directory of your choosing run the following to generate a sample React project:

```
$ runway gen-sample static-react
```

3. A new folder will be created entitled `static-react`. If you'd like your project to have a different name feel free to change it at this time.

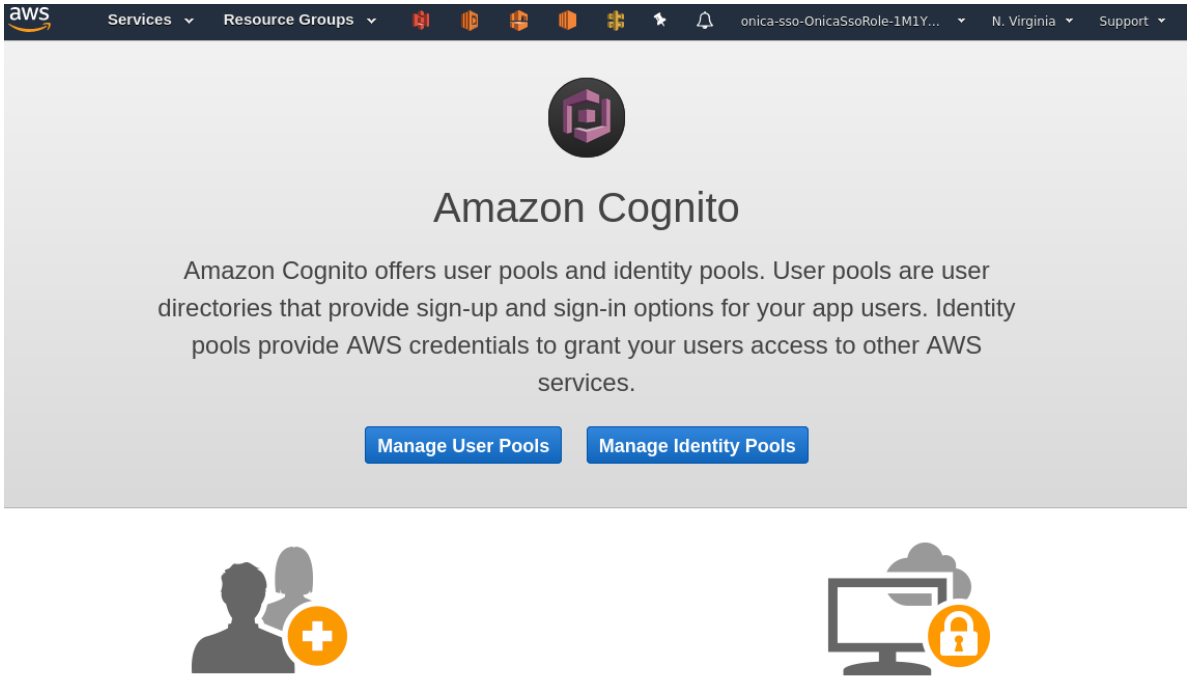
```
$ mv static-react my-static-site
```

4. Change directories into the new project folder and prepare the project directory. See *Repo Structure* for more details.

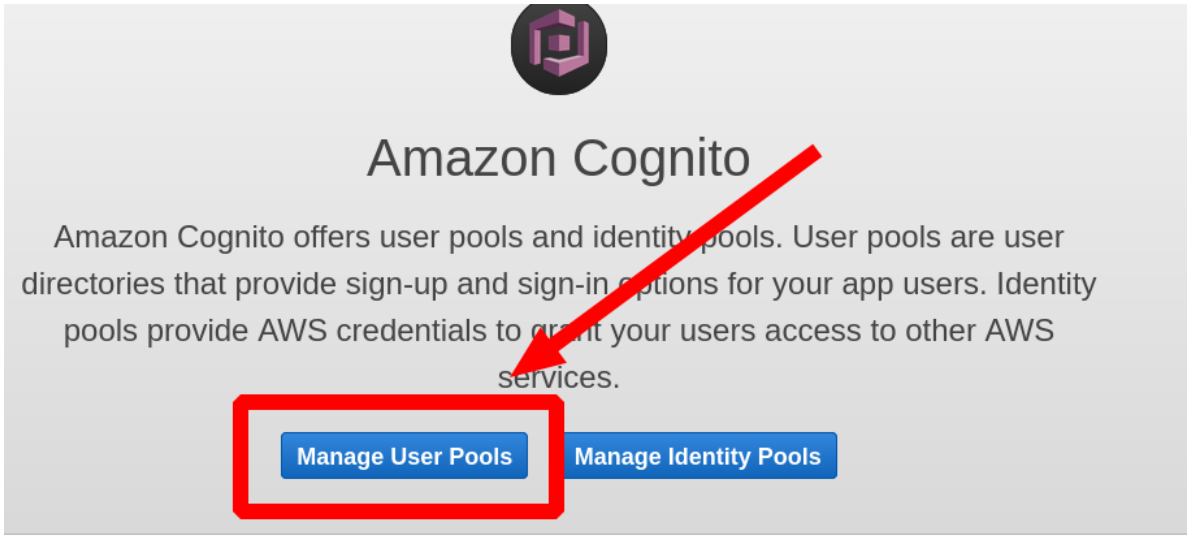
```
$ cd my-static-site
$ git init && git checkout -b ENV-dev
```

User Pool Setup

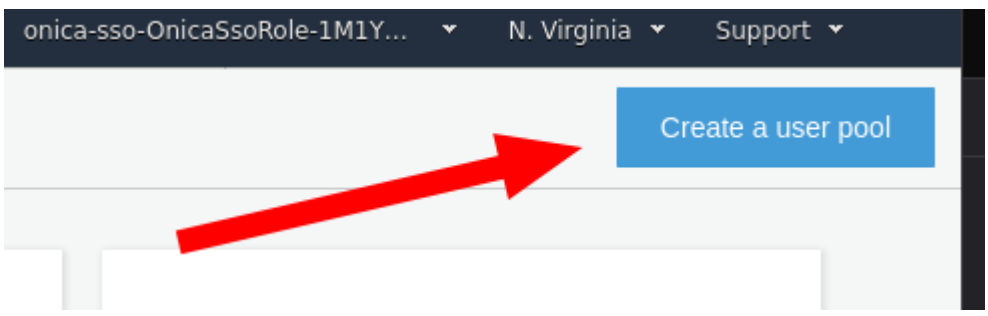
1. The default `runway.yml` document that is provided with `gen-sample static-react` is a good baseline document for deploying a standard static single page application without the need of authentication. In this example we'll be leveraging `Auth@Edge` to provide protection to our application, not allowing anyone to view or download site resources without first authenticating. To accomplish this we need to create a Cognito UserPool. Login to your AWS Console and search for *cognito*.



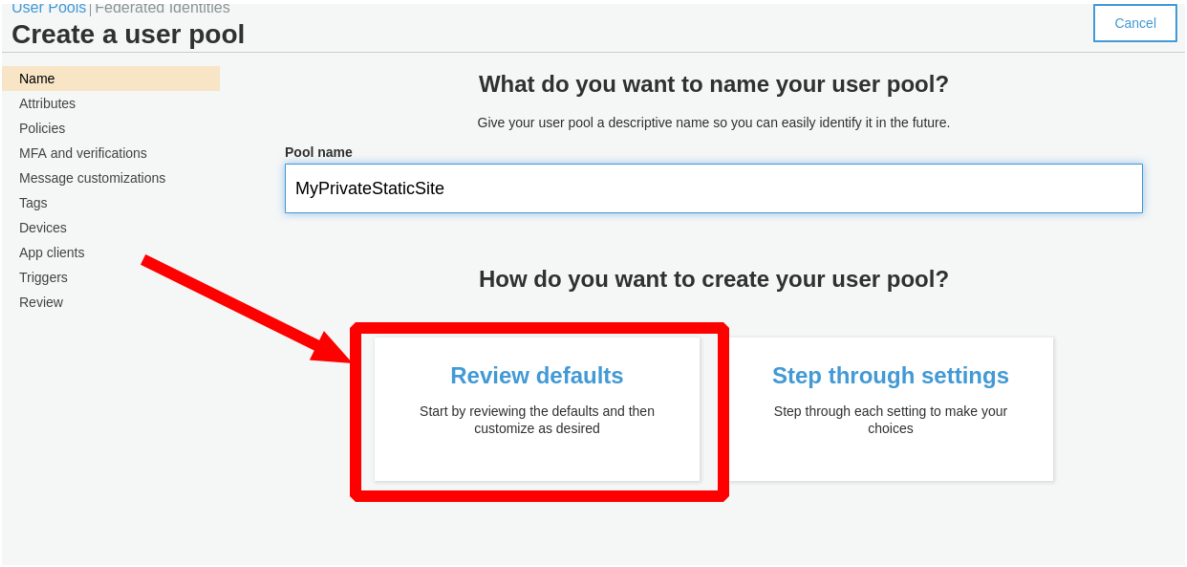
2. Click `Manage User Pools`



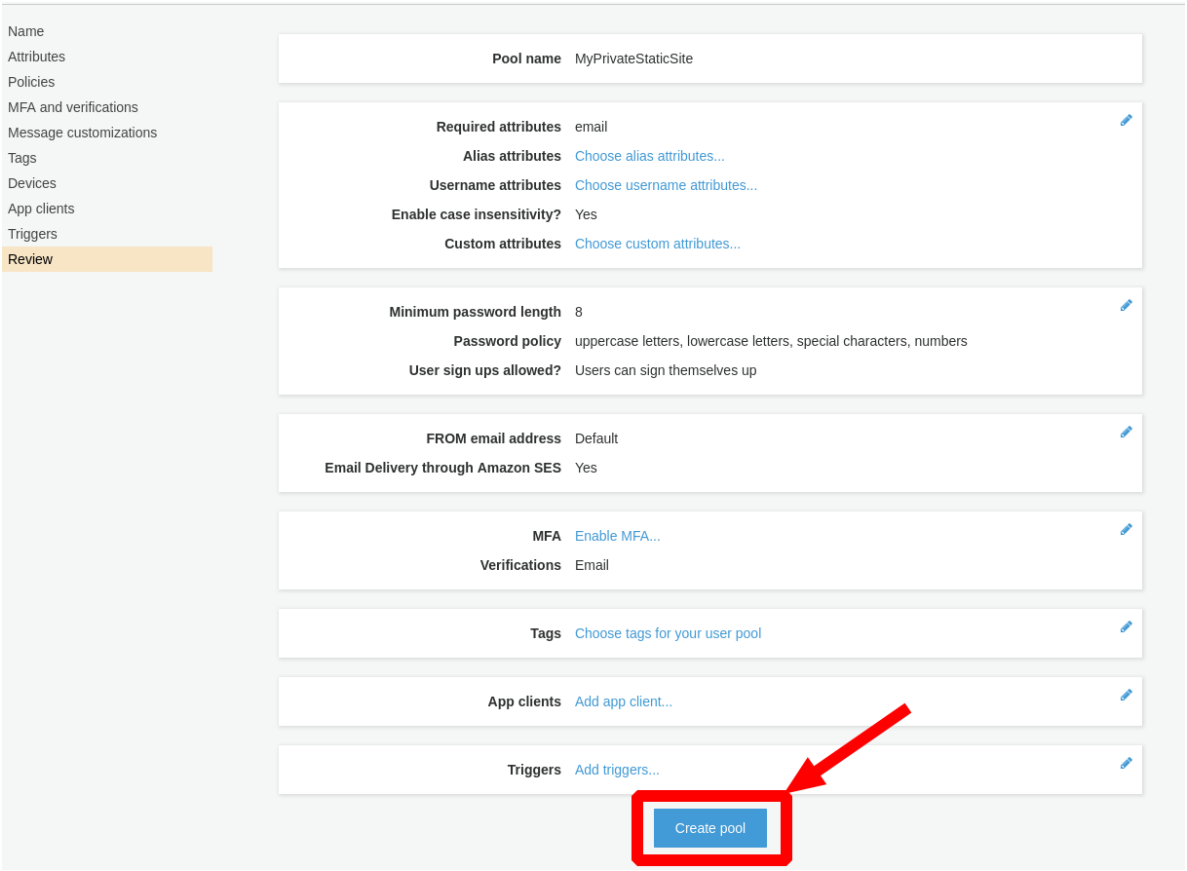
3. Click `Create a user pool`



4. You will be asked to provide a name for your User Pool. For our example we will be using a default Cognito User Pool, but you can `Step through settings` to customize your pool if you so choose. After entering your Pool name click the `Review defaults` button.

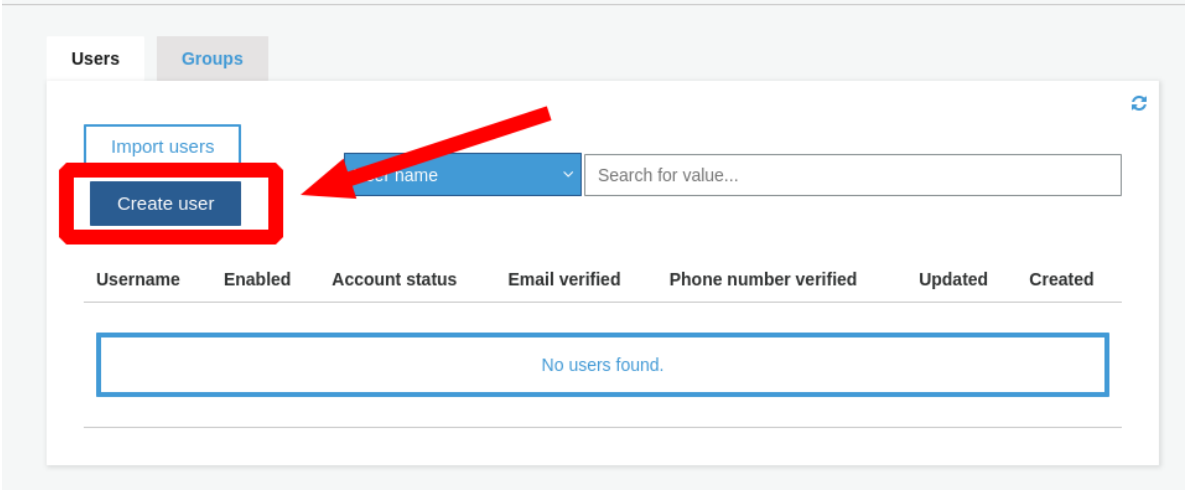


5. Review all the settings are accurate prior to clicking `Create pool`.

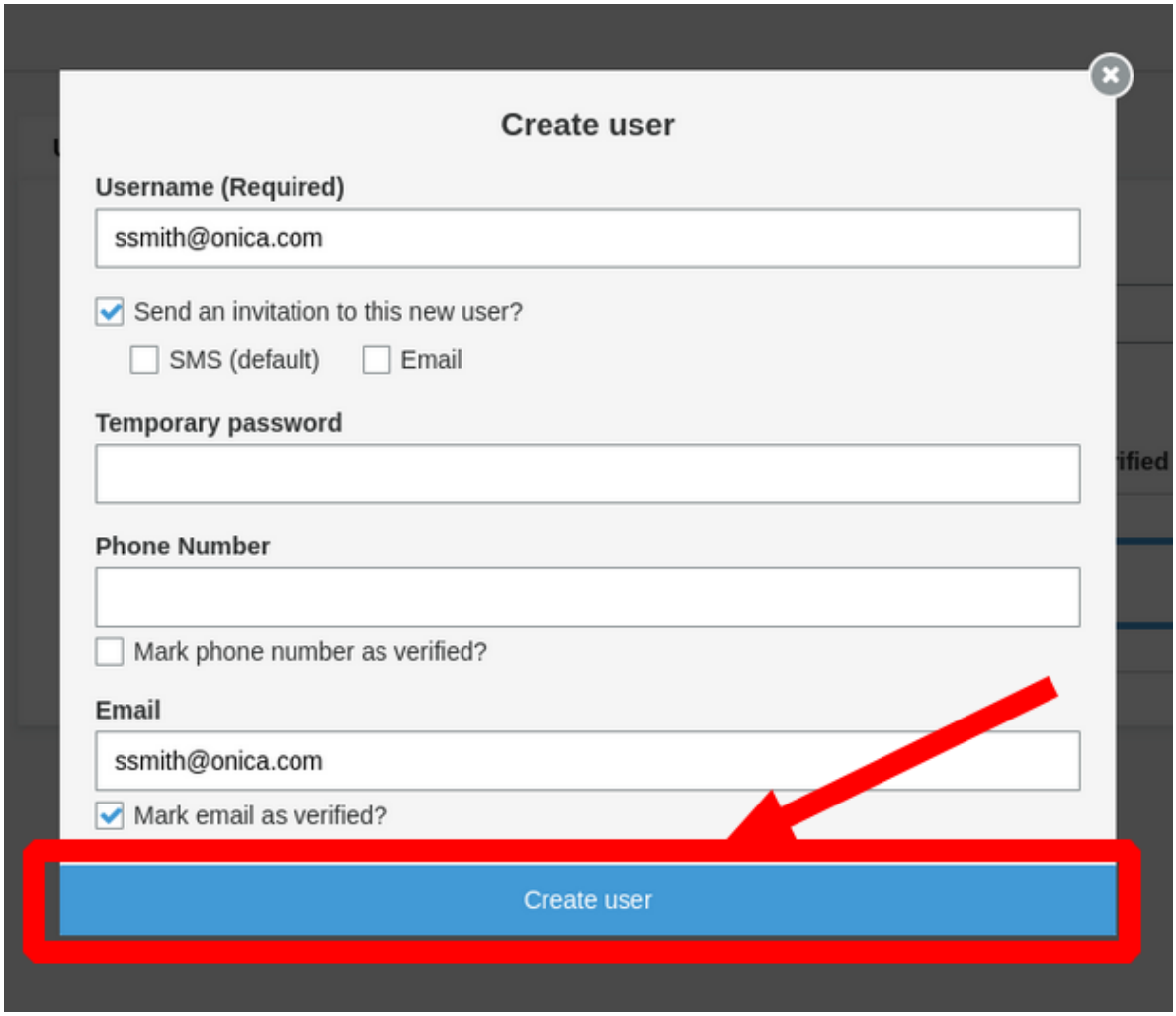


6. Next let's create a test user to verify our authentication functionality after deployment. Click the `Users and groups` list link.

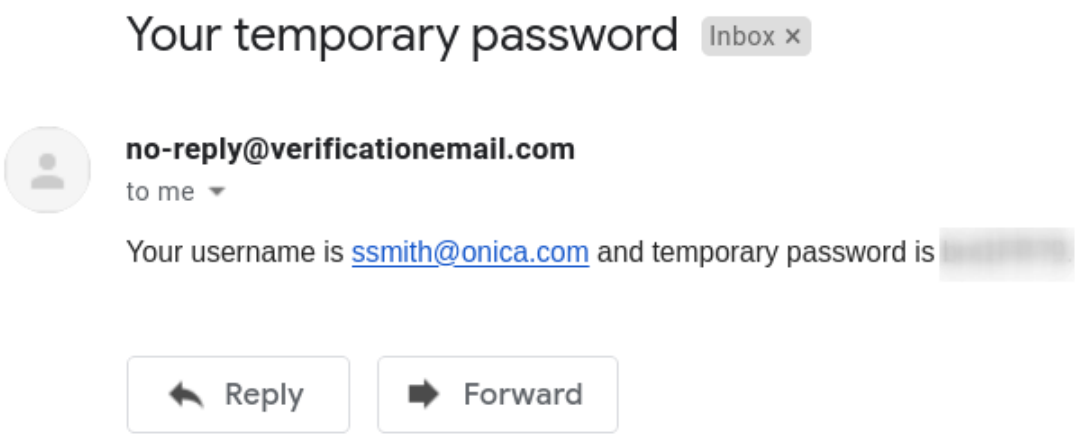
7. Click `Create user`



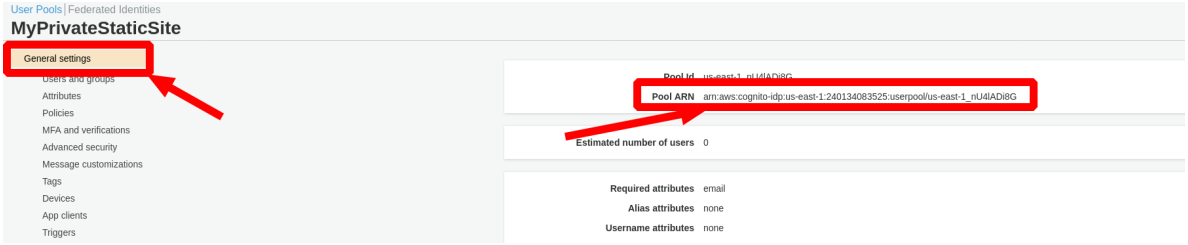
8. In the form provided give a valid email address for the `Username (Required)` and `Email` entries. Ensure `Send an invitation to this new user?` is checked so you can receive the temporary password to access the site. Click the `Create user` button.



9. Check the email address provided, you should receive a notification email from Cognito with the username and password that will need to be used for initial authentication.



10. Now we need to retrieve the ARN for the User Pool we just created and add it to the `deployments -> modules -> environments -> dev` section of our `runway.yml` document. Click the `General Settings` list link to retrieve the ARN.

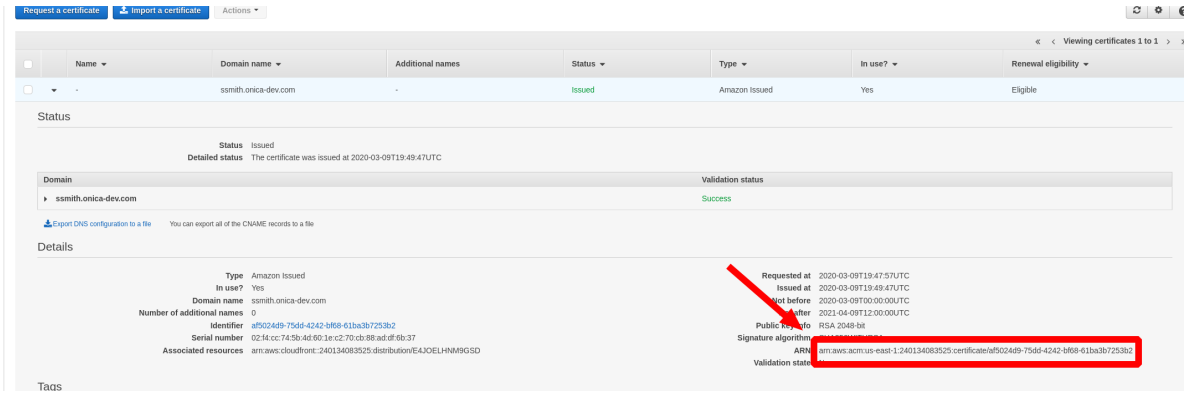



```
staticsite_user_pool_arn: YOUR_USER_POOL_ARN
```

Domain Aliases with ACM Certificate

In this example we are going to be using an alias custom domain name to identify the CloudFront Distribution. This series of steps is **optional**, a domain will still be provided with the Distribution if you choose not to use a custom domain. This guide assumes that you have already purchased and registered a custom domain and created and validated an ACM certificate.

1. The ARN of the ACM certificate is required when providing an alias domain name. From the search bar of the AWS console locate `certificate manager`. In this screen dropdown the details of your issued and validated certificate and locate the ARN.



2. Create two entries in the `runway.yml` configuration file under the `deployments -> modules -> environments -> dev` heading. One for the alias we're looking to provide, and the other for it's ARN.

```
staticsite_aliases: YOUR_CUSTOM_DOMAIN_NAMES_COMMA_SEPARATED
staticsite_acmcert_arn: YOUR_ACM_ARN
```

Cleanup

By default the `gen-sample static-react` sample `runway.yaml` document comes with `staticsite_cf_disable: true` added. Due to the nature of the authorization a Distribution is required. Remove this line from your config file.

4.4.3 Deploying

Execute `runway deploy`. Deployment will take some time (mostly waiting for the CloudFront distribution to stabilize).

The CloudFront domain at which the site can be reached will be displayed near the last lines of output once deployment is complete, e.g.:

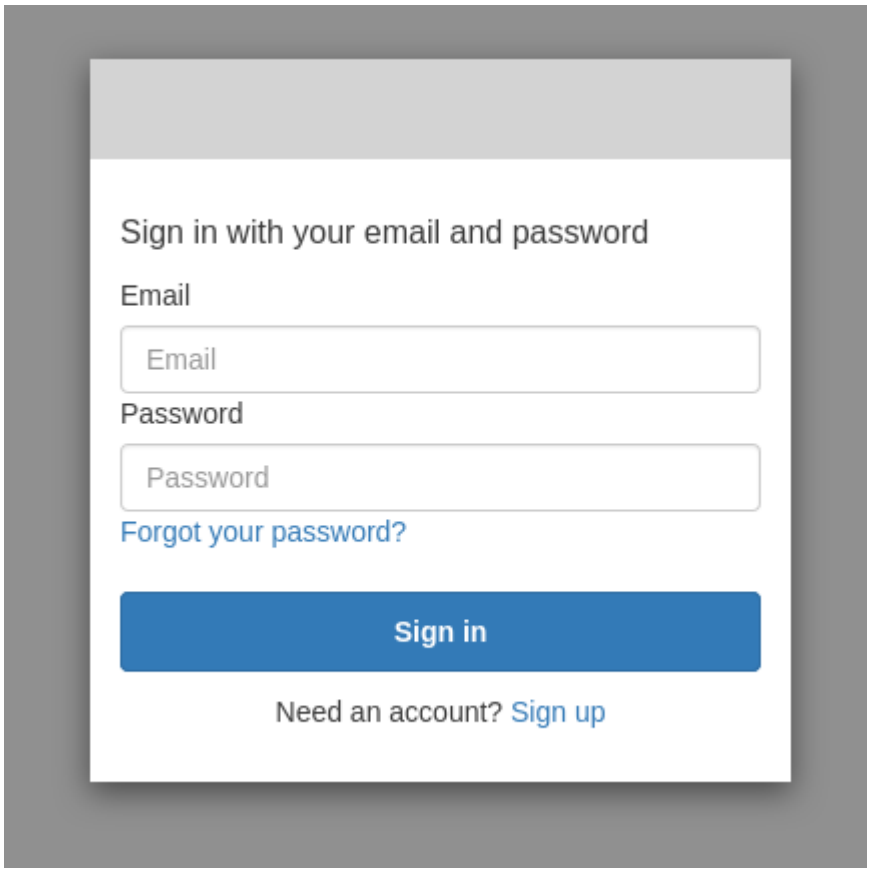
```
staticsite: sync & CF invalidation of E17B5JWPMTX5Z8 (domain ddy1q4je03d7u.cloudfront.
↪net) complete
```

Since we're using a custom domain alias the Distribution will also be accessible by that domain.

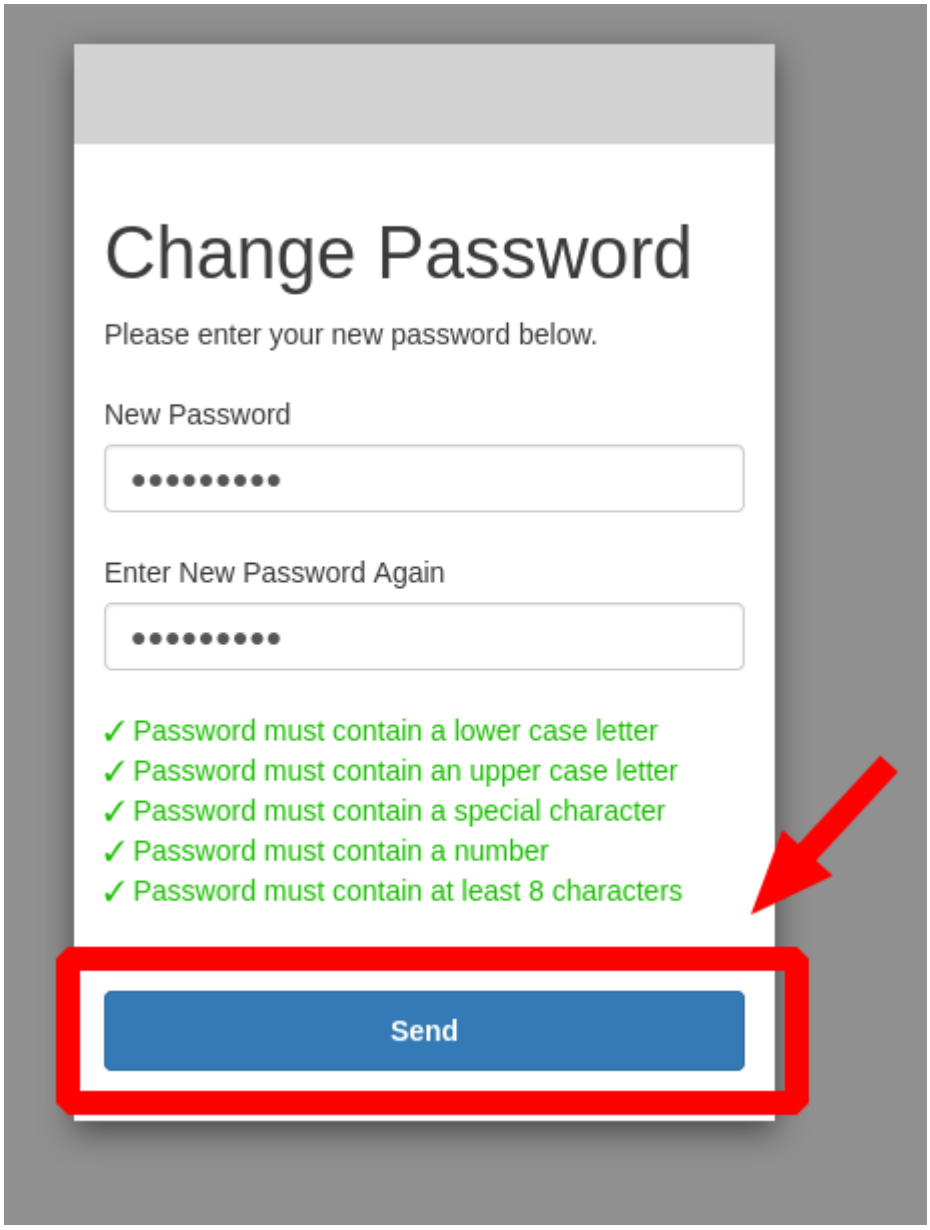
4.4.4 Accessing and Authorizing

Authorizing

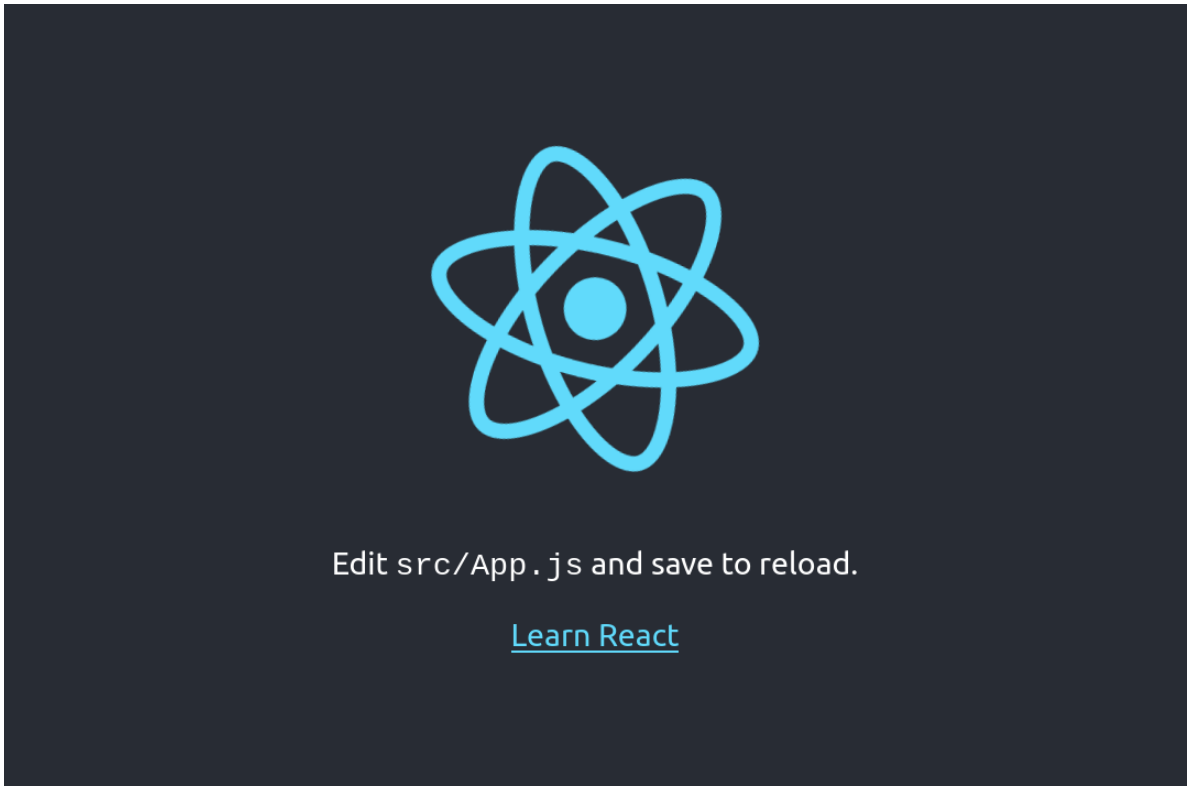
1. From your browser enter either the CloudFront Distribution domain or the alias you provided. You will be greeted with the Cognito login screen. Enter the username and temporary password you received in step 9 of *User Pool Setup*:



2. You will be asked to change your password based on the validation requirements you specified when creating the User Pool. Once you have satisfied the requirements click `Send`



3. You will be greeted with the default React App home page:



Sign-Out

By default a `/sign-out` path is provided to sign out of Cognito.

4.4.5 Teardown

Execute `runway destroy`.

CHAPTER

FIVE

COMMANDS

```
$ runway --help
Usage: runway [OPTIONS] COMMAND [ARGS]...

  Runway CLI.

  Full documentation available at https://docs.onica.com/projects/runway/

Options:
  --version   Show the version and exit.
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
...
```

5.1 deploy

```
$ runway deploy --help
Usage: runway deploy [OPTIONS]

  Deploy infrastructure as code.

  Process
  -------
  1. Determines the deploy environment.
      - "-e, --deploy-environment" option
      - "DEPLOY_ENVIRONMENT" environment variable
      - git branch name
          - strips "ENV-" prefix, master is converted to common
          - ignored if "ignore_git_branch: true"
      - name of the current working directory
  2. Selects deployments & modules to deploy.
      - (default) prompts
      - (tags) module contains all tags
      - (non-interactive) all
  3. Deploys selected deployments/modules in the order defined.
```

(continues on next page)

(continued from previous page)

```
Options:
  --ci                            Run in non-interactive mode.
  --debug                         Supply once to display Runway debug logs.
                                  Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                  Manually specify the name of the deploy
                                  environment.
  --no-color                      Disable color in Runway's logs.
  --tag <tag>...                  Select modules by tag or tags. This option
                                  can be specified more than once to build a
                                  list of tags that are treated as "AND".
                                  (e.g. "--tag <tag1> --tag <tag2>" would
                                  select all modules with BOTH tags).
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway deploy
$ runway deploy --ci --deploy-environment example
$ runway deploy --tag tag1 --tag tag2
```

5.2 destroy

```
$ runway destroy --help
Usage: runway destroy [OPTIONS]

  Destroy infrastructure as code.

  Process
  -------
  1. Determines the deploy environment.
      - "-e, --deploy-environment" option
      - "DEPLOY_ENVIRONMENT" environment variable
      - git branch name
          - strips "ENV-" prefix, master is converted to common
          - ignored if "ignore_git_branch: true"
      - name of the current working directory
  2. Selects deployments & modules to deploy.
      - (default) prompts
      - (tags) module contains all tags
      - (non-interactive) all
  3. Destroys selected deployments/modules in reverse the order defined.

Options:
  --ci                            Run in non-interactive mode.
  --debug                         Supply once to display Runway debug logs.
```

(continues on next page)

(continued from previous page)

```
                                Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                Manually specify the name of the deploy
                                environment.
  --no-color                    Disable color in Runway's logs.
  --tag <tag>...                Select modules by tag or tags. This option
                                can be specified more than once to build a
                                list of tags that are treated as "AND".
                                (e.g. "--tag <tag1> --tag <tag2>" would
                                select all modules with BOTH tags).
  --verbose                     Display Runway verbose logs.
  -h, --help                    Show this message and exit.
```

Example

```
$ runway destroy
$ runway destroy --ci --deploy-environment example
$ runway destroy --tag tag1 --tag tag2
```

5.3 dismantle

```
$ runway dismantle --help
Usage: runway dismantle [OPTIONS]

  Alias of "runway destroy".

  For more information, refer to the output of "runway destroy --help".

Options:
  --ci                          Run in non-interactive mode.
  --debug                       Supply once to display Runway debug logs.
                                Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                Manually specify the name of the deploy
                                environment.
  --no-color                    Disable color in Runway's logs.
  --tag <tag>...                Select modules by tag or tags. This option
                                can be specified more than once to build a
                                list of tags that are treated as "AND".
                                (e.g. "--tag <tag1> --tag <tag2>" would
                                select all modules with BOTH tags).
  --verbose                     Display Runway verbose logs.
  -h, --help                    Show this message and exit.
```

Example

```
$ runway dismantle
$ runway dismantle --ci --deploy-environment example
$ runway dismantle --tag tag1 --tag tag2
```

5.4 docs

```
$ runway docs --help
Usage: runway docs [OPTIONS]

  Open the Runway documentation web site using the default web browser.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway docs
```

5.5 envvars

```
$ runway envvars --help
Usage: runway envvars [OPTIONS]

  Output "env_vars" defined in the Runway config file.

  OS environment variables can be set in the Runway config file for different
  Runway environments (e.g. dev & prod KUBECONFIG values). This command allows
  access to these values for use outside of Runway.

  NOTE: Only outputs "env_vars" defined in deployments, not modules.

Options:
  --debug                          Supply once to display Runway debug logs.
                                   Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                   Manually specify the name of the deploy
                                   environment.
  --no-color                       Disable color in Runway's logs.
```

(continues on next page)

(continued from previous page)

```
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway envvars
$ eval "$(runway envvars)"
$ runway envvars --deploy-environment example
```

5.6 gen-sample

```
$ runway gen-sample --help
Usage: runway gen-sample [OPTIONS] COMMAND [ARGS]...

  Generate a sample Runway module module/project.

  The sample is created in the current directory. If a directory already
  exists with the name Runway tries to use, the sample will not be created.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.

Commands:
  cdk-csharp      cdk + c# (sampleapp.cdk)
  cdk-py          cdk + py (sampleapp.cdk)
  cdk-tsc         cdk + tsc (sampleapp.cdk)
  cfn             cfngin + cfn (sampleapp.cfn)
  cfngin          cfngin + troposphere (sampleapp.cfn)
  k8s-cfn-repo    k8s + cfn (k8s-cfn-infrastructure)
  k8s-flux-repo   k8s + flux + tf (k8s-tf-infrastructure)
  k8s-tf-repo     k8s + tf (k8s-tf-infrastructure)
  sls-py          sls + python (sampleapp.sls)
  sls-tsc         sls + tsc (sampleapp.sls)
  static-angular  angular static site (static-angular)
  static-react    react static site (static-react)
  tf              tf (sampleapp.tf)
```

Example

```
$ runway gen-sample cfngin
$ runway gen-sample static-react
```

5.7 init

```
$ runway init --help
Usage: runway init [OPTIONS]

  Run initialization/bootstrap steps.

  Process
  -------
  1. Determines the deploy environment.
      - "-e, --deploy-environment" option
      - "DEPLOY_ENVIRONMENT" environment variable
      - git branch name
          - strips "ENV-" prefix, master is converted to common
          - ignored if "ignore_git_branch: true"
      - name of the current working directory
  2. Selects deployments & modules to deploy.
      - (default) prompts
      - (tags) module contains all tags
      - (non-interactive) all
  3. Initializes/bootstraps selected deployments/modules in the order defined.
     (e.g. "cdk bootstrap", "terraform init")

  Steps By Module Type
  --------------------
  - AWS CDK: Runs "cdk bootstrap".
  - CFNgin: Creates the "cfngin_bucket" if needed.
  - Terraform: Runs "terraform init", changes the workspace if needed, runs
    "terraform init" again if the workspace was changed, and finally
    downloads/updates Terraform modules.

Options:
  --ci                            Run in non-interactive mode.
  --debug                         Supply once to display Runway debug logs.
                                  Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                  Manually specify the name of the deploy
                                  environment.
  --no-color                      Disable color in Runway's logs.
  --tag <tag>...                  Select modules by tag or tags. This option
                                  can be specified more than once to build a
                                  list of tags that are treated as "AND".
                                  (e.g. "--tag <tag1> --tag <tag2>" would
                                  select all modules with BOTH tags).
```

(continues on next page)

(continued from previous page)

```
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway init
$ runway init --ci --deploy-environment example
$ runway init --tag tag1 --tag tag2
```

5.8 kbenv install

```
$ runway kbenv install --help
Usage: runway kbenv install [OPTIONS] [<version>]

  Install the specified <version> of kubectl (e.g. v1.14.0).

  If no version is specified, Runway will attempt to find and read a
  ".kubectl-version" file in the current directory. If this file doesn't
  exist, nothing will be installed.

  Compatible with https://github.com/alexppg/kbenv.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway kbenv install
$ runway kbenv install v1.14.0
```

5.9 kbenv list

```
$ runway kbenv list --help
Usage: runway kbenv list [OPTIONS]

  List the versions of kubectl that have been installed by Runway and/or
  kbenv.
```

(continues on next page)

(continued from previous page)

```
Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway kbenv list
```

5.10 kbenv run

```
$ runway kbenv run --help
Usage: runway kbenv run [OPTIONS] <args>

  Run a kubectl command.

  Uses the version of kubectl specified in the ".kubectl-version" file in the
  current directory.

  IMPORTANT: When using options shared with Runway "--" must be placed before
  the kubectl command.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway kbenv run version --client
$ runway kbenv run -- --help
```

5.11 kbenv uninstall

```
$ runway kbenv uninstall --help
Usage: runway kbenv uninstall [OPTIONS] [<version>]

  Uninstall the specified <version> of kubectl (e.g. v1.14.0) or all installed
  versions.

  If no version is specified, Runway will attempt to find and read a
  ".kubectl-version" file in the current directory.

Options:
  --all       Uninstall all versions of kubectl.
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway kbenv uninstall v1.21.0
$ runway kbenv uninstall --all
```

5.12 new

```
$ runway new --help
Usage: runway new [OPTIONS]

  Create an example runway.yml file in the correct directory.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway new
$ runway new --debug
```

5.13 plan

Note: Currently only supported for *AWS Cloud Development Kit (CDK)*, *CloudFormation & Troposphere*, and *Terraform*.

```
$ runway new --help
Usage: runway new [OPTIONS]

  Create an example runway.yml file in the correct directory.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway plan
$ runway plan --ci --deploy-environment example
$ runway plan --tag tag1 --tag tag2
```

5.14 preflight

```
$ runway preflight --help
Usage: runway preflight [OPTIONS]

  Alias of "runway test".

  For more information, refer to the output of "runway test --help".

Options:
  --debug                          Supply once to display Runway debug logs.
                                   Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                   Manually specify the name of the deploy
```

(continues on next page)

(continued from previous page)

```
                                environment.
  --no-color                    Disable color in Runway's logs.
  --verbose                     Display Runway verbose logs.
  -h, --help                    Show this message and exit.
```

Example

```
$ runway preflight
```

5.15 run-python

```
$ runway run-python --help
Usage: runway run-python [OPTIONS] <filename>

  Execute a python script using a bundled copy of python.

  This command can execute actions using python without requiring python to be
  installed on a system. This is only applicable when installing a binary
  release of Runway (not installed from PyPi). When installed from PyPI, the
  current interpreter is used.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway run-python my_script.py
```

5.16 schema cfngin

```
$ runway schema cfngin --help
Usage: runway schema cfngin [OPTIONS]

  Output JSON schema for CFNgin configuration files.

  The schema that is output can be used to validate configuration files. It
  can also be added to text editors to provide autocompletion, tool tips, and
```

(continues on next page)

(continued from previous page)

```
  suggestions within configuration files.

Options:
  --debug                   Supply once to display Runway debug logs. Supply
                            twice to display all debug logs.
  --indent <int>            Number of spaces to use per indentation level when
                            output JSON.  [default: 4]
  --no-color                Disable color in Runway's logs.
  -o, --output <file-name>  If provided, schema will be saved to a file
                            instead of being output to stdout.
  --verbose                 Display Runway verbose logs.
  -h, --help                Show this message and exit.
```

Example

```
$ runway schema cfngin --output cfngin-schema.json
```

5.17 schema runway

```
$ runway schema runway --help
Usage: runway schema runway [OPTIONS]

  Output JSON schema Runway configuration files.

  The schema that is output can be used to validate configuration files. It
  can also be added to text editors to provide autocompletion, tool tips, and
  suggestions within configuration files.

Options:
  --debug                   Supply once to display Runway debug logs. Supply
                            twice to display all debug logs.
  --indent <int>            Number of spaces to use per indentation level when
                            output JSON.  [default: 4]
  --no-color                Disable color in Runway's logs.
  -o, --output <file-name>  If provided, schema will be saved to a file
                            instead of being output to stdout.
  --verbose                 Display Runway verbose logs.
  -h, --help                Show this message and exit.
```

Example

```
$ runway schema runway --output runway-schema.json
```

5.18 takeoff

```
$ runway takeoff --help
Usage: runway takeoff [OPTIONS]

  Alias of "runway deploy".

  For more information, refer to the output of "runway deploy --help".

Options:
  --ci                            Run in non-interactive mode.
  --debug                         Supply once to display Runway debug logs.
                                  Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                  Manually specify the name of the deploy
                                  environment.
  --no-color                      Disable color in Runway's logs.
  --tag <tag>...                  Select modules by tag or tags. This option
                                  can be specified more than once to build a
                                  list of tags that are treated as "AND".
                                  (e.g. "--tag <tag1> --tag <tag2>" would
                                  select all modules with BOTH tags).
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway takeoff
$ runway takeoff --ci --deploy-environment example
$ runway takeoff --tag tag1 --tag tag2
```

5.19 taxi

```
$ runway taxi --help
Usage: runway taxi [OPTIONS]

  Alias of "runway plan".

  For more information, refer to the output of "runway plan --help".
```

(continues on next page)

(continued from previous page)

```
Options:
  --ci                            Run in non-interactive mode.
  --debug                         Supply once to display Runway debug logs.
                                  Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                  Manually specify the name of the deploy
                                  environment.
  --no-color                      Disable color in Runway's logs.
  --tag <tag>...                  Select modules by tag or tags. This option
                                  can be specified more than once to build a
                                  list of tags that are treated as "AND".
                                  (e.g. "--tag <tag1> --tag <tag2>" would
                                  select all modules with BOTH tags).
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway taxi
$ runway taxi --ci --deploy-environment example
$ runway taxi --tag tag1 --tag tag2
```

5.20 test

```
$ runway test --help
Usage: runway test [OPTIONS]

  Execute tests as defined in the Runway config file.

  If one of the tests fail, the command will exit immediately unless
  "required: false" is set on the failing test.

  If the failing test is not required, the next test will be executed.

  If any of the tests fail, the command will exit with a non-zero exit code.

Options:
  --debug                         Supply once to display Runway debug logs.
                                  Supply twice to display all debug logs.
  -e, --deploy-environment <env-name>
                                  Manually specify the name of the deploy
                                  environment.
  --no-color                      Disable color in Runway's logs.
  --verbose                       Display Runway verbose logs.
  -h, --help                      Show this message and exit.
```

Example

```
$ runway test
```

5.21 tfenv install

```
$ runway tfenv install --help
Usage: runway tfenv install [OPTIONS] [<version>]

  Install the specified <version> of Terraform (e.g. 0.12.0).

  If no version is specified, Runway will attempt to find and read a
  ".terraform-version" file in the current directory. If this file doesn't
  exist, nothing will be installed.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway tfenv install 0.12.0
```

5.22 tfenv list

```
$ runway tfenv list --help
Usage: runway tfenv list [OPTIONS]

  List the versions of Terraform that have been installed by Runway and/or
  tfenv.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway tfenv list
```

5.23 tfenv run

```
$ runway tfenv run --help
Usage: runway tfenv run [OPTIONS] <args>

  Run a Terraform command.

  Uses the version of Terraform specified in the ".terraform-version" file in
  the current directory.

  IMPORTANT: When using options shared with Runway "--" must be placed before
  the Terraform command.

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway tfenv run --version
$ runway tfenv run -- --help
```

5.24 tfenv uninstall

```
$ runway tfenv uninstall --help
Usage: runway tfenv uninstall [OPTIONS] [<version>]

  Uninstall the specified <version> of Terraform (e.g. 0.12.0) or all
  installed versions.

  If no version is specified, Runway will attempt to find and read a
  ".terraform-version" file in the current directory.

Options:
  --all       Uninstall all versions of Terraform.
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
```

(continues on next page)

(continued from previous page)

```
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway tfenv uninstall 1.0.0
$ runway tfenv uninstall --all
```

5.25 whichenv

```
$ runway whichenv --help
Usage: runway whichenv [OPTIONS]

  Print the current deploy environment name to stdout.

  When run, the deploy environment will be determined from one of the
  following (in order of precedence):

    - "DEPLOY_ENVIRONMENT" environment variable
    - git branch name (strips "ENV-" prefix, master => common)
    - current working directory

Options:
  --debug     Supply once to display Runway debug logs. Supply twice to
              display all debug logs.
  --no-color  Disable color in Runway's logs.
  --verbose   Display Runway verbose logs.
  -h, --help  Show this message and exit.
```

Example

```
$ runway whichenv
```

CHAPTER

SIX

RUNWAY CONFIG FILE

The Runway config file is where all options are defined. It contains definitions for deployments, tests, and some global options that impact core functionality.

The Runway config file can have two possible names, `runway.yml` or `runway.yaml`. It must be stored at the root of the directory containing the modules to be deployed.

6.1 Top-Level Configuration

deployments: `List[`*deployment*`]`

A list of deployments that will be processed in the order they are defined. See *Deployment* for detailed information about defining this value.

Example

```yaml
deployments:
  - name: example
    modules:
      - sampleapp-01.cfn
      - path: sampleapp-02.cfn
    regions:
      - us-east-1
```

ignore_git_branch: `bool = false`

Optionally exclude the git branch name when determining the current *deploy environment*.

This can be useful when using the directory name or environment variable to set the *deploy environment* to ensure the correct value is used.

Example

```yaml
ignore_git_branch: true
```

Note: The existence of `DEPLOY_ENVIRONMENT` in the environment will automatically ignore the git branch.

runway_version: `str = ">=1.10.0"`

Define the versions of Runway that can be used with this configuration file.

The value should be a PEP 440 compliant version specifier set.

Example

Listing 1: greater than or equal to 1.14.0

```
runway_version: ">=1.14.0"
```

Listing 2: explicit version

```
runway_version: "==14.0.0"
```

Listing 3: greater than or equal to 1.14.0 but less than 2.0.0

```
runway_version: ">=1.14.0,<2.0.0"  # or ~=1.14.0
```

New in version 1.11.0.

tests: `Optional[List[test]] = []`

List of Runway test definitions that are executed with the *test command* command. See *Test* for detailed information about defining this value.

Example

```
tests:
  - name: Hello World
    type: script
    args:
      commands:
        - echo "Hello World"
```

variables: `Optional[Dict[str, Any]] = {}`

Runway variables are used to fill values that could change based on any number of circumstances. They can also be used to simplify the Runway config file by pulling lengthy definitions into another YAML file. Variables can be consumed in the config file by using the *var lookup* in any field that supports *Lookups*.

By default, Runway will look for and load a `runway.variables.yml` or `runway.variables.yaml` file that is in the same directory as the Runway config file. The file path and name of the file can optionally be defined in the config file. If the file path is explicitly provided and the file can't be found, an error will be raised.

Variables can also be defined in the Runway config file directly. This can either be in place of a dedicated variables file, extend an existing file, or override values from the file.

Important: The `variables` and the variables file cannot contain lookups. If there is a lookup string in either of these locations, they will not be resolved.

Example

```yaml
deployments:
  - modules:
      - path: sampleapp.cfn
    env_vars: ${var env_vars}  # exists in example-file.yml
    parameters:
      namespace: ${var namespace}-${env DEPLOY_ENVIRONMENT}
    regions: ${var regions.${env DEPLOY_ENVIRONMENT}}

variables:
  file_path: example-file.yml
  namespace: example
  regions:
    dev:
      - us-east-1
      - us-west-2
```

variables.**file_path**: Optional[str]

Explicit path to a variables file that will be loaded and merged with the variables defined here.

Example

```yaml
variables:
  file_path: some-file.yml
```

variables.**sys_path**: Optional[str] = ./

Directory to use as the root of a relative *variables.file_path*. If not provided, the current working directory is used.

Example

```yaml
variables:
  sys_path: ./../variables
```

6.2 Deployment

class deployment

A deployment defines modules and options that affect the modules.

Deployments are processed during a *deploy*/*destroy*/*plan* action. If the processing of one deployment fails, the action will end.

During a *deploy*/*destroy* action, the user has the option to select which deployment will run unless the `CI` environment variable (`--ci` cli option) is set, the `--tag <tag>...` cli option was provided, or only one deployment is defined.

Lookup Support

Important: Due to how a deployment is processed, some values are resolved twice. Once before processing and once during processing.

Because of this, the fields that are resolved before processing begins will not have access to values set during processing like `AWS_REGION`, `AWS_DEFAULT_REGION`, and `DEPLOY_ENVIRONMENT` for the pre-processing resolution which can result in a `FailedLookup` error. To avoid errors during the first resolution due to the value not existing, provide a default value for the *Lookup*.

The values mentioned will be set before the second resolution when processing begins. This ensures that the correct values are passed to the module.

Impacted fields are marked with an asterisk (*).

The following fields support lookups:

- *account_alias* *
- *account_id* *
- *assume_role* *
- *env_vars* *
- *environments*
- *module_options*
- *parallel_regions* *
- *parameters*
- *regions* *

account_alias: Optional[str] = None

An AWS account alias use to verify the currently assumed role or credentials. Verification is performed by listing the account's alias and comparing the result to what is defined. This requires the credentials being used to have `iam:ListAccountAliases` permissions.

Example

Listing 4: using a literal value

```yaml
deployments:
  - account_alias: example-dev
```

Listing 5: using a lookup

```yaml
deployments:
  - account_alias: example-${env DEPLOY_ENVIRONMENT}
  - account_alias: ${var account_alias.${env DEPLOY_ENVIRONMENT}}

variables:
  account_alias:
    dev: example-dev
```

Changed in version 2.0.0: No longer accepts a `typing.Dict`.

account_id: Optional[str] = None

An AWS account ID use to verify the currently assumed role or credentials. Verification is performed by getting the caller identity. This does not required any added permissions as it is allowed by default. However, it does require that `sts:GetCallerIdentity` is not explicitly denied.

Example

Listing 6: using a literal value

```yaml
deployments:
  - account_id: 123456789012
```

Listing 7: using a lookup

```yaml
deployments:
  - account_id: ${var account_id.${env DEPLOY_ENVIRONMENT}}

variables:
  account_id:
    dev: 123456789012
```

Changed in version 2.0.0: No longer accepts a `typing.Dict`.

assume_role: Optional[*assume_role_definition*, str] = {}

Assume an AWS IAM role when processing the deployment. The credentials being used prior to assuming the role must to `iam:AssumeRole` permissions for the role provided.

Example

Listing 8: using a literal value

```yaml
deployments:
  - assume_role: arn:aws:iam::123456789012:role/name
```

Listing 9: using a lookup in a detailed definition

```yaml
deployments:
  - assume_role:
      arn: ${var assume_role.${env DEPLOY_ENVIRONMENT}}
      post_deploy_env_revert: True

variables:
  assume_role:
    dev:
      arn:aws:iam::123456789012:role/name
```

Changed in version 2.0.0: No longer accepts a `typing.Dict` defining a value per deploy environment.

class assume_role_definition

arn: str

The ARN of the AWS IAM role to be assumed.

duration: `int = 3600`

The duration, in seconds, of the session.

post_deploy_env_revert: `bool = false`

Revert the credentials stored in environment variables to what they were prior to execution after the deployment finished processing.

session_name: `str = runway`

An identifier for the assumed role session.

env_vars: `Optional[Dict[str, Union[List[str], str]]] = {}`

Additional variables to add to the environment when processing the deployment.

Anything defined here is merged with the value of *module.env_vars*.

Example

Listing 10: using a lookup as the value

```yaml
deployments:
  - env_vars:
      NAME: value
      KUBECONFIG:
        - .kube
        - ${env DEPLOY_ENVIRONMENT}
        - config
```

Listing 11: using a lookup in the value

```yaml
deployments:
  - env_vars: ${var env_vars.${env DEPLOY_ENVIRONMENT}}

variables:
  env_vars:
    dev:
      NAME: value
```

Changed in version 2.0.0: No longer accepts a `typing.Dict` defining a value per deploy environment. The entire value of the field is used for all environments.

environments: Optional[Dict[str, Union[bool, List[str], str]]] = {}

Explicitly enable/disable the deployment for a specific deploy environment, AWS Account ID, and AWS Region combination. Can also be set as a static boolean value.

Anything defined here is merged with the value of *module.environments*.

Example

```yaml
deployments:
  - environments:
      dev: True
      test: 123456789012
      qa: us-east-1
      prod:
        - 123456789012/ca-central-1
        - us-west-2
        - 234567890123
```

Listing 12: using a lookup as the value

```yaml
deployments:
  - environments: ${var environments}

variables:
  environments:
    dev: True
```

Changed in version 1.4.0: Now acts as an explicit toggle for deploying modules to a set AWS Account/AWS Region. For passing values to a module, *deployment.parameters*/*module.parameters* should be used instead.

Changed in version 2.0.0: If defined and the current deploy environment is missing from the definition, processing will be skipped.

modules: List[Union[*module*, str]]

A list of modules to process as part of a deployment.

Example

```yaml
deployments:
  - modules:
      - sampleapp-01.cfn
      - path: sampleapp-02.cfn
```

module_options: Optional[Union[Dict[str, Any], str]] = {}

Options that are passed directly to the modules within this deployment.

Anything defined here is merged with the value of *module.options*.

Example

```yaml
deployments:
  - module_options:
      example: value
```

Listing 13: using a lookup as the value

```yaml
deployments:
  - module_options:
      example: ${var example}

variables:
  example: value
```

Listing 14: using a lookup in the value

```yaml
deployments:
  - module_options: ${var parameters}

variables:
  parameters:
    example: value
```

name: Optional[str] = None

The name of the deployment to be displayed in logs and the interactive selection menu.

Example

```yaml
deployments:
  - name: networking
```

parallel_regions: Optional[Union[List[str], str]] = []

A list of AWS Regions to process asynchronously.

Only one of *parallel_regions* or *regions* can be defined.

Asynchronous deployment only takes effect when running non-interactively. Otherwise processing will occur synchronously.

assume_role.post_deploy_env_revert will always be `true` when run in parallel.

Can be used in tandem with *module.parallel*.

Example

Listing 15: using a lookup as the value

```yaml
deployments:
  - parallel_regions:
      - us-east-1
      - us-west-2
      - ${var third_region.${env DEPLOY_ENVIRONMENT}}

variables:
  third_region:
    dev: ca-central-1
```

Listing 16: using a lookup in the value

```yaml
deployments:
    - parallel_regions: ${var regions.${env DEPLOY_ENVIRONMENT}}

  variables:
    regions:
      - us-east-1
      - us-west-2
```

New in version 1.3.0.

parameters: Optional[Union[Dict[str, Any], str]] = {}

Used to pass variable values to modules in place of an environment configuration file.

Anything defined here is merged with the value of *module.parameters*.

Example

Listing 17: using a lookup as the value

```yaml
deployments:
  - parameters:
      namespace: example-${env DEPLOY_ENVIRONMENT}
```

Listing 18: using a lookup in the value

```yaml
deployments:
  - parameters: ${var parameters.${env DEPLOY_ENVIRONMENT}}

variables:
  parameters:
    dev:
      namespace: example-dev
```

New in version 1.4.0.

regions: Optional[Union[Dict[str, Union[List[str], str], List[str], str]] = []

A list of AWS Regions to process this deployment in.

Only one of *parallel_regions* or *regions* can be defined.

Can be used to define asynchronous processing similar to *parallel_regions*.

Example

Listing 19: synchronous

```yaml
deployments:
  - regions:
      - us-east-1
      - us-west-2
```

Listing 20: asynchronous

```yaml
deployments:
  - regions:
      parallel:
        - us-east-1
        - us-west-2
        - ${var third_region.${env DEPLOY_ENVIRONMENT}}

variables:
  third_region:
    dev: ca-central-1
```

Listing 21: using a lookup in the value

```yaml
deployments:
    - regions: ${var regions.${env DEPLOY_ENVIRONMENT}}

  variables:
    regions:
      - us-east-1
      - us-west-2
```

6.3 Module

class module

A module defines the directory to be processed and applicable options.

It can consist of *CloudFormation*, *Terraform*, *Serverless Framework*, *AWS CDK*, *Kubernetes*, or a *Static Site*. It is recommended to place the appropriate extension on each directory for identification (but it is not required). See *Repo Structure* for examples of a module directory structure.

Suffix/Extension	IaC Tool/Framework
`.cdk`	*AWS CDK*
`.cfn`	*CloudFormation*
`.k8s`	*Kubernetes*
`.sls`	*Serverless Framework*
`.tf`	*Terraform*
`.web`	*Static Site*

A module is only deployed if there is a corresponding environment file present, it is explicitly enabled via *deployment.environments*/*module.environments*, or *deployment.parameters*/*module.parameters* is defined. The naming format of an environment file varies per module type. See *Module Configurations* for acceptable environment file name formats.

Modules can be defined as a string or a mapping. The minimum requirement for a module is a string that is equal to the name of the module directory. Providing a string is the same as providing a value for *path* in a mapping definition.

Using a mapping to define a module provides the ability to specify all the fields listed here.

Lookup Support

The following fields support lookups:

- *class_path*
- *env_vars*
- *environments*
- *options*
- *parameters*
- *path*

class_path: Optional[str] = null

> **Note:** Most users will never need to use this. It is only used for custom module type handlers.

Import path to a custom Runway module handler class. See *Module Configurations* for detailed usage.

Example

```yaml
deployments:
  - modules:
    - class_path: runway.module.cloudformation.CloudFormation
```

env_vars: Optional[Dict[str, Union[List[str], str]]] = {}

Additional variables to add to the environment when processing the deployment.

Anything defined here is merged with the value of *deployment.env_vars*. Values defined here take precedence.

Example

Listing 22: using a lookup as the value

```yaml
deployments:
  - modules:
    - env_vars:
        NAME: VALUE
        KUBECONFIG:
          - .kube
          - ${env DEPLOY_ENVIRONMENT}
          - config
```

Listing 23: using a lookup in the value

```yaml
deployments:
  - modules:
      - env_vars: ${var env_vars.${env DEPLOY_ENVIRONMENT}}

variables:
  env_vars:
    dev:
      NAME: value
```

Changed in version 2.0.0: No longer accepts a `typing.Dict` defining a value per deploy environment. The entire value of the field is used for all environments.

environments: Optional[Dict[str, Union[bool, List[str], str]]] = {}

Explicitly enable/disable the deployment for a specific deploy environment, AWS Account ID, and AWS Region combination. Can also be set as a static boolean value.

Anything defined here is merged with the value of *deployment.environments*. Values defined here take precedence.

Example

```yaml
deployments:
  - modules:
    - environments:
      dev: True
      test: 123456789012
      qa: us-east-1
      prod:
        - 123456789012/ca-central-1
        - us-west-2
        - 234567890123
```

Listing 24: using a lookup as the value

```yaml
deployments:
  - modules:
    - environments: ${var environments}
```

(continues on next page)

(continued from previous page)

```
variables:
  environments:
    dev: True
```

Changed in version 1.4.0: Now acts as an explicit toggle for deploying modules to a set AWS Account/AWS Region. For passing values to a module, *deployment.parameters*/*module.parameters* should be used instead.

Changed in version 2.0.0: If defined and the current deploy environment is missing from the definition, processing will be skipped.

name: `Optional[str]`

The name of the module to be displayed in logs and the interactive selection menu.

If a name is not provided, the *path* value is used.

Example

```
deployments:
  - modules:
    - name: networking
```

options: `Optional[Union[Dict[str, Any], str]] = {}`

Options that are passed directly to the module type handler class.

The options that can be used with each module vary. For detailed information about options for each type of module, see *Module Configurations*.

Anything defined here is merged with the value of *deployment.module_options*. Values defined here take precedence.

Example

```
deployments:
  - module:
    - options:
        example: value
```

Listing 25: using a lookup as the value

```
deployments:
  - module:
    - options:
        example: ${var example}

variables:
  example: value
```

Listing 26: using a lookup in the value

```yaml
deployments:
  - module:
    - options: ${var parameters}

variables:
  parameters:
    example: value
```

parallel: Optional[List[*module*]] = []

List of *module* definitions that can be executed asynchronously.

Incompatible with *class_path*, *path*, and *type*.

Asynchronous deployment only takes effect when running non-interactively. Otherwise processing will occur synchronously.

Example

```yaml
deployments:
  - modules:
    - parallel:
      - path: sampleapp-01.cfn
      - path: sampleapp-02.cfn
```

parameters: Optional[Union[Dict[str, Any], str]] = {}

Used to pass variable values to modules in place of an environment configuration file.

Anything defined here is merged with the value of *deployment.parameters*. Values defined here take precedence.

Example

Listing 27: using a lookup as the value

```yaml
deployments:
  - modules:
    - parameters:
        namespace: example-${env DEPLOY_ENVIRONMENT}
```

Listing 28: using a lookup in the value

```yaml
deployments:
  - modules:
    - parameters: ${var parameters.${env DEPLOY_ENVIRONMENT}}

variables:
  parameters:
    dev:
      namespace: example-dev
```

New in version 1.4.0.

path: `Optional[Union[str, Path]]`

Directory (relative to the Runway config file) containing IaC. The directory can either be on the local file system or a network accessible location.

See *path* for more detailed information.

Example

Listing 29: using a lookup

```yaml
deployments:
  - modules:
    - path: sampleapp-${env DEPLOY_ENVIRONMENT}.cfn
```

New in version 1.4.0.

tags: `Optional[List[str]] = []`

A list of strings to categorize the module which can be used with the CLI to quickly select a group of modules.

This field is only used by the `--tag` CLI option.

Example

```yaml
deployments:
  - modules:
    - tags:
      - app:sampleapp
      - type:network
```

type: `Optional[str]`

Explicitly define the type of IaC contained within the directory. This can be useful when Runway fails to automatically determine the correct module type.

Accepted Values

- cdk
- cloudformation
- kubernetes
- serverless
- terraform
- static

Example

```yaml
deployments:
  - modules:
    - type: static
```

New in version 1.4.0.

6.3.1 path

path can either be defined as a local path relative to the Runway config file or a network accessible (remote) location.

When the value is identified as a remote location, Runway is responsible for retrieving resources from the location and caching them locally for processing. This allows the remote resources to be handled automatically by Runway rather than needing to manually retrieve them or employ another mechanism to retrieve them.

Remote Location Syntax

The syntax is based on that of Terraform module sources.

```
${source}::${uri}//${location}?${arguments}
```

source Combined with the following `::` separator, it is used to identify the location as remote. The value determines how Runway with handle retrieving resources from the remote location.

uri The uniform resource identifier when targeting a remote resource. This instructs runway on where to retrieve your module.

location An optional location within the remote location (assessed after the resources have been retrieve) relative to the root of the retrieve resources.

This field is preceded by a `//`. If not defining a location, this separator does not need to be provided.

arguments An optional ampersand (`&`) delimited list of `key=value` pairs that are unique to each remote location source. These are used to provide granular control over how Runway retrieves resources from the remote location.

This field is preceded by a `?`. If not defining a location, this separator does not need to be provided.

Remote Location Sources

Git Repository

Runway can retrieve a git repository to process modules contained within it. Below is an example of using a module in a git repository as well as a breakdown of the values being provided to each field.

```yaml
deployments:
    - modules:
        # ${source}::${uri}//${location}?${arguments}
        - path: git::git://github.com/foo/bar.git//my/path?branch=develop
```

Field	Value	Description
source	`git`	The *type* of remote location source.
uri	`git://github.com/foo/bar.git`	The protocol and URI address of the git repository.
location	`my/path`	The relative path from the root of the repo where the module is located. *(optional)*
arguments	`branch=develop`	After cloning the repository, checkout the develop branch. *(optional)*

Arguments

branch Name of a branch to checkout after cloning the git repository.

Only one of *branch*, *commit*, or *tag* can be defined. If none are defined, *HEAD* is used.

commit After cloning the git repository, reset *HEAD* to the given commit hash.

Only one of *branch*, *commit*, or *tag* can be defined. If none are defined, *HEAD* is used.

tag After cloning the git repository, reset *HEAD* to the given tag.

Only one of *branch*, *commit*, or *tag* can be defined. If none are defined, *HEAD* is used.

6.4 Test

class test

Tests can be defined as part of the Runway config file. This is to remove the need for complex Makefiles or scripts to initiate test runners. Simply define all tests for a project in the Runway config file and use the *test command* to execute them.

Lookup Support

Note: Runway does not set `AWS_REGION` or `AWS_DEFAULT_REGION` environment variables when using the *test command*.

The following fields support lookups:

- *test.args*
- *test.required*

args: `Optional[Union[Dict[str, Any], str]] = {}`

Arguments to be passed to the test. Supported arguments vary by test type. See *Build-in Test Types* for the arguments supported by each test type.

Example

```yaml
tests:
  - args:
      commands:
        - echo "Hello world"
```

name: `Optional[str]`

Name of the test. Used to more easily identify where different tests begin/end in the logs and to identify which tests failed.

Example

```yaml
tests:
  - name: example-test
```

required: `bool = false`

Whether the test must pass for subsequent tests to be run. If `false`, testing will continue if the test fails.

If the test fails, the *test command* will always return a non-zero exit code regardless of this value.

Example

Listing 30: using a literal value

```yaml
tests:
  - required: false
```

Listing 31: using a lookup

```yaml
tests:
  - required: ${var test.required}

variables:
  test:
    required: false
```

type: str

The type of test to run.

Accepted Values

- *cfn-lint*
- *script*
- *yamllint*

Example

```yaml
tests:
  - type: script
```

CHAPTER

SEVEN

LOOKUPS

Runway Lookups allow the use of variables within the Runway config file. These variables can then be passed along to *deployments*, *modules* and *tests*.

The syntax for a lookup is `${<lookup-name> <query>::<arg>=<arg-val>}`.

Component	Description
`${`	Signifies the opening of the lookup.
`<lookup-name>`	The name of the lookup you wish to use (e.g. `env`). This signifies the *source* of the data to be retrieved by the lookup.
	The separator between lookup name a query.
`<query>`	The value the lookup will be looking for (e.g. `AWS_REGION`). When using a lookup on a dictionary/mapping, like for the *var*, you can get nested values by providing the full path to the value (e.g. `ami.dev`).
`::`	The separator between a query and optional arguments.
`<arg>=<arg-val>`	An argument passed to a lookup. Multiple arguments can be passed to a lookup by separating them with a comma (`,`). Arguments are optional. Supported arguments depend on the lookup being used.

Lookups can be nested (e.g. `${var ami_id.${var AWS_REGION}}`).

Lookups can't resolve other lookups. For example, if i use `${var region}` in my Runway config file to resolve the `region` from my variables file, the value in the variables file can't be `${env AWS_REGION}`. Well, it can but it will resolve to the literal value provided, not an AWS region like you may expect.

Contents

7.1 Lookup Arguments

Arguments can be passed to Lookups to effect how they function.

To provide arguments to a Lookup, use a double-colon (`::`) after the query. Each argument is then defined as a **key** and **value** separated with equals (=) and the arguments themselves are separated with a comma (`,`). The arguments can have an optional space after the comma and before the next key to make them easier to read but this is not required. The value of all arguments are read as strings.

Example

```
${var my_query::default=true, transform=bool}
${env MY_QUERY::default=1,transform=bool}
```

Each Lookup may have their own, specific arguments that it uses to modify its functionality or the value it returns. There is also a common set of arguments that all Lookups accept.

7.1.1 Common Lookup Arguments

`default: Any`

If the Lookup is unable to find a value for the provided query, this value will be returned instead of raising an exception.

`get: str`

Can be used on a dictionary type object to retrieve a specific piece of data. This is executed after the optional `load` step.

`indent: int`

Number of spaces to use per indent level when transforming a dictionary type object to a string.

`load: Literal['json', 'troposphere', 'yaml']`

Load the data to be processed by a Lookup using a specific parser. This is the first action taking on the data after it has been retrieved from it's source. The data must be in a format that is supported by the parser in order for it to be used.

json Loads a JSON serializable string into a dictionary like object.

troposphere Loads the `properties` of a subclass of `troposphere.BaseAWSObject` into a dictionary.

yaml Loads a YAML serializable string into a dictionary like object.

region: `str`

AWS region used when creating a `boto3.Session` to retrieve data. If not provided, the region currently being processed will be used. This can be specified to always get data from one region regardless of region is being deployed to.

transform: `Literal['bool', 'str']`

Transform the data that will be returned by a Lookup into a different data type. This is the last action taking on the data before it is returned.

Supports the following:

bool Converts a string or boolean value into a boolean.

str Converts any value to a string. The original data type determines the end result.

`list`, `set`, and `tuple` will become a comma delimited list

`dict` and anything else will become an escaped JSON string.

Example

```
deployments:
  - parameters:
      some_variable: ${var some_value::default=my_value}
      comma_list: ${var my_list::default=undefined, transform=str}
```

7.2 Built-in Lookups

7.2.1 cfn

Important: The Stack must exist in CloudFormation before the module using this Lookup begins processing to successfully get a value. This means that the Stack must have been deployed by another module, run before the one using this Lookup, or it must have been created external to Runway.

Query Syntax `<stack-name>.<output-name>[::<arg>=<arg-val>, ...]`

Retrieve a value from CloudFormation Stack Outputs.

The query syntax for this lookup is `<stack-name>.<output-name>`. When specifying the output name, be sure to use the *Logical ID* of the output; not the *Export.Name*.

If the Lookup is unable to find a CloudFormation Stack Output matching the provided query, the default value is returned or an exception is raised to show why the value could be be resolved (e.g. Stack does not exist or output does not exist on the Stack).

New in version 1.11.0.

See also:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html

Arguments

This Lookup supports all *Common Lookup Arguments*.

Example

```
deployments:
  - modules:
      path: sampleapp.tf
      options:
        terraform_backend_config:
          bucket: ${cfn common-tf-state.TerraformStateBucketName::region=us-east-1}
          dynamodb_table: ${cfn common-tf-state.TerraformStateTableName::region=us-east-
↪1}
          region: us-east-1
```

7.2.2 ecr

Retrieve a value from AWS Elastic Container Registry (ECR).

This Lookup only supports very specific queries.

New in version 1.18.0.

Supported Queries

login-password

Get a password to login to ECR registry.

The returned value can be passed to the login command of the container client of your preference, such as the CFNgin *docker.login*. After you have authenticated to an Amazon ECR registry with this Lookup, you can use the client to push and pull images from that registry as long as your IAM principal has access to do so until the token expires. The authorization token is valid for **12 hours**.

Arguments

This Lookup does not support any arguments.

Example

```
deployments:
  - modules:
    - path: example.cfn
      parameters:
        ecr_password: ${ecr login-password}
  ...
```

7.2.3 env

Query Syntax `<variable-name>[::<arg>=<arg-val>, ...]`

Retrieve a value from an environment variable.

The value is retrieved from a copy of the current environment variables that is saved to the context object. These environment variables are manipulated at runtime by Runway to fill in additional values such as `DEPLOY_ENVIRONMENT` and `AWS_REGION` to match the current execution.

Note: `DEPLOY_ENVIRONMENT` and `AWS_REGION` can only be resolved during the processing of a module. To ensure no error occurs when trying to resolve one of these in a *Deployment* definition, provide a default value.

If the Lookup is unable to find an environment variable matching the provided query, the default value is returned or a `ValueError` is raised if a default value was not provided.

New in version 1.4.0.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- region

Example

```
deployment:
  - modules:
      - path: sampleapp.cfn
        parameters:
          creator: ${env USER}
    env_vars:
      ENVIRONMENT: ${env DEPLOY_ENVIRONMENT::default=default}
```

7.2.4 random.string

Query Syntax `<desired-length>[::<arg>=<arg-val>, ...]`

Generate a random string of the given length.

New in version 2.2.0.

Arguments

`digits: bool = True`

When generating the random string, the string may contain digits (`[0-9]`). If the string can contain digits, it will always contain at least one.

`lowercase: bool = True`

When generating the random string, the string may contain lowercase letters (`[a-z]`). If the string can contain lowercase letters, it will always contain at least one.

punctuation: `bool = False`

When generating the random string, the string may contain ASCII punctuation (`[!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]`). If the string can contain ASCII punctuation, it will always contain at least one.

uppercase: `bool = True`

When generating the random string, the string may contain uppercase letters (`[A-Z]`). If the string can contain uppercase letters, it will always contain at least one.

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- default
- get
- indent
- load
- region

Example

This example shows the use of this lookup to create an SSM parameter that will retain value generated during the first deployment. Even through subsequent deployments generate a new value that is passed to the hook, the hook does not overwrite the value of an existing parameter.

```yaml
pre_deploy: &hooks
  - path: runway.cfngin.hooks.ssm.parameter.SecureString
    args:
      name: /${namespace}/password
      overwrite: false
      value: ${random.string 12::punctuation=true}

post_destroy: *hooks
```

7.2.5 ssm

Query Syntax `<parameter>[::<arg>=<arg-val>, ...]`

Retrieve a value from SSM Parameter Store.

If the Lookup is unable to find an SSM Parameter matching the provided query, the default value is returned or `ParameterNotFound` is raised if a default value is not provided.

Parameters of type `SecureString` are automatically decrypted.

Parameters of type `StringList` are returned as a list.

New in version 1.5.0.

Arguments

This Lookup supports all *Common Lookup Arguments*.

Example

```
deployment:
  - modules:
    - path: sampleapp.cfn
      parameters:
        secret_value: ${ssm /example/secret}
        conf_file: ${ssm /example/config/json::load=json, get=value}
        toggle: ${ssm toggle::load=yaml, get=val, transform=bool}
    env_vars:
      SOME_VARIABLE: ${ssm /example/param::region=us-east-1}
      DEFAULT_VARIABLE: ${ssm /example/default::default=default}
```

7.2.6 var

Query Syntax `<variable-name>[::<arg>=<arg-val>, ...]`

Retrieve a variable from the variables file or definition.

If the Lookup is unable to find an defined variable matching the provided query, the default value is returned or a `ValueError` is raised if a default value was not provided.

Nested values can be used by providing the full path to the value but, it will not select a list element.

The returned value can contain any YAML support data type (dictionaries/mappings/hashes, lists/arrays/sequences, strings, numbers, and boolean).

New in version 1.4.0.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- region

Example

```
deployment:
  - modules:
      - path: sampleapp.cfn
        parameters:
          ami_id: ${var ami_id.${env AWS_REGION}}
    env_vars:
      SOME_VARIABLE: ${var some_variable::default=default}
```

CHAPTER

EIGHT

DEFINING TESTS

Tests can be defined in the *runway config file* to test your modules in any way you desire before deploying. They are run by using the `runway test` *command*. Tests are run in the order they are defined.

Example:

```yaml
tests:
  - name: example-test
    type: script
    args:
      commands:
        - echo "Success!"
```

Contents

8.1 Test Failures

The default behavior if a test failed is to continue running the rest of the tests and return a non-zero exit code at the end. This behavior can be modified to stop testing on failure by setting `required: true` in the test definition. This will terminate execution if a test fails; no further tests will be run.

Example

```yaml
tests:
  - name: hello-world
    type: script
    required: true
    args:
      commands:
        - echo "Hello World!"  && exit 1
```

8.2 Built-in Test Types

8.2.1 cfn-lint

Source Tool https://github.com/aws-cloudformation/cfn-python-lint

Description *Validate CloudFormation yaml/json templates against the CloudFormation spec and additional checks. Includes checking valid values for resource properties and best practices.*

In order to use this test, there must be a `.cfnlintrc` file in the same directory as the *Runway config file*.

Example:

```yaml
tests:
  - name: cfn-lint-example
    type: cfn-lint
```

8.2.2 script

Executes a list of provided commands. Each command is run in its own subprocess.

Commands are passed into the test using the `commands` argument.

Example:

```yaml
tests:
  - name: hello-world
    type: script
    args:
      commands:
        - echo "Hello World!"
```

8.2.3 yamllint

Source Tool https://github.com/adrienverge/yamllint

Description *A linter for YAML files. yamllint does not only check for syntax validity, but for weirdnesses like key repetition and cosmetic problems such as lines length, trailing spaces, indentation, etc.*

A `.yamllint` file can be placed at in the same directory as the *Runway config file* to customize the linter or, the Runway provided template will be used.

Example:

```yaml
tests:
  - name: yamllint-example
    type: yamllint
```

CHAPTER

NINE

REPO STRUCTURE

Projects deployed via Runway can be structured in a few ways.

Contents

9.1 Git Branches as Environments

This example shows two modules using environment git branches (these same files would be present in each environment branch, with changes to any environment promoted through branches).

```
.
├── myapp.cfn
│   ├── dev-us-west-2.env
│   ├── prod-us-west-2.env
│   ├── myapp.yaml
│   └── templates
│       └── foo.json
├── myapp.tf
│   ├── backend.tfvars
│   ├── dev-us-east-1.tfvars
│   ├── prod-us-east-1.tfvars
│   └── main.tf
└── runway.yml
```

9.2 Directories as Environments

The same two modules from the above *Git Branches as Environments* structure can instead be stored in a normal single-branch git repo. Each directory correlates with an environment (dev and prod in this example).

Environment changes are done by copying the environments' contents between each other. E.g., promotion from dev to prod could be as simple as `diff -u dev/ prod/` followed by `rsync -r --delete dev/ prod/`

Enabling that automated promotion is one of the reasons this example below has prod config files in the dev folder and vice versa. When promotions between environments are more hand managed, this is not technically required.

```
.
├── dev
│   ├── myapp.cfn
│   │   ├── dev-us-west-2.env
│   │   ├── prod-us-west-2.env
│   │   ├── myapp.yaml
│   │   └── templates
│   │       └── myapp_cf_template.json
│   ├── myapp.tf
│   │   ├── backend.tfvars
│   │   ├── dev-us-east-1.tfvars
│   │   ├── prod-us-east-1.tfvars
│   │   └── main.tf
│   └── runway.yml
└── prod
    ├── myapp.cfn
    │   ├── dev-us-west-2.env
    │   ├── prod-us-west-2.env
    │   ├── myapp.yaml
    │   └── templates
    │       └── myapp_cf_template.json
    ├── myapp.tf
    │   ├── backend.tfvars
    │   ├── dev-us-east-1.tfvars
    │   ├── prod-us-east-1.tfvars
    │   └── main.tf
    └── runway.yml
```

9.3 Directories as Environments with a Single Module

Another sample repo structure, showing environment folders containing a single CloudFormation modules at their root (using the `ignore_git_branch` Runway config file field and a single declared module of `./` to merge the Environment & Module folders).

See the *Directories as Environments* example above for more information on why this shows prod config files in the dev folder and vice versa.

```
.
├── dev
│   ├── dev-us-west-2.env
│   ├── prod-us-west-2.env
```

(continues on next page)

(continued from previous page)

```
│   ├── myapp.yaml
│   ├── runway.yml
│   └── templates
│       └── myapp_cf_template.json
└── prod
    ├── dev-us-west-2.env
    ├── prod-us-west-2.env
    ├── myapp.yaml
    ├── runway.yml
    └── templates
        └── myapp_cf_template.json
```

CHAPTER

TEN

MODULE CONFIGURATION

10.1 AWS Cloud Development Kit (CDK)

The CDK module type is deployed using the AWS Cloud Development Kit (CDK). Runway uses system installed npm to install the CDK per-module. This means that the CDK must be included as a dev dependency in the **package.json** of the module.

- *Configuration*
- *Directory Structure*
- *Advanced Features*

10.1.1 Configuration

Standard CDK rules apply but, we have some added prerequisites, recommendations, and caveats.

Contents

Prerequisites

- npm installed on the system
- CDK must be a dev dependency of the module (e.g. `npm install --save-dev aws-cdk`)

We strongly recommend you commit the package-lock.json that is generated after running `npm install`.

Recommendations

Feature Flags

The AWS CDK uses feature flags to enable potentially breaking behaviors prior to the next major release that makes them default behaviors. Flags are stored as Runtime context values in `cdk.json` (or `~/.cdk.json`).

aws-cdk:enableDiffNoFail

This feature flag is available in version `^1.0.0`.

If this is set to `true` (recommend), `cdk diff` will always exit with `0`. With this set to `false`, `cdk diff` will exit with a non-zero exit code if there is a diff. This will result in Runway exiting before all stacks/modules/deployments are processed.

Example

```
{
  "context": {
    "aws-cdk:enableDiffNoFail": true
  },
}
```

Environments

Unlike some other module types, CDK does not have a file that can be used to configure an environment. It can only be configured using the *deployment.environments*/*module.environments* field.

Example

```
deployments:
  - modules:
      - path: mycdkmodule.cdl
        environments:
          dev: true
          prod: true
  - modules:
      - path: myothercdkmodule.cdk
    environments:
      dev: true
      prod: true
```

Options

build_steps: Optional[List[str]] = None

Shell commands to be run before processing the module.

See *Build Steps* for more details.

Example

```yaml
options:
  build_steps:
    - npx tsc
```

skip_npm_ci: bool = False

Skip running `npm ci` in the module directory prior to processing the module. See *Disable NPM CI* for more details.

Example

```yaml
options:
  skip_npm_ci: true
```

10.1.2 Directory Structure

Example directory structures for a CDK module.

Contents

C# Example

```
.
├── add-project.hook.d.ts
├── cdk.json
├── package.json
├── package-lock.json
└── src
    ├── HelloCdk
    │   ├── HelloCdk.csproj
    │   ├── HelloConstruct.cs
    │   ├── HelloStack.cs
```

(continues on next page)

(continued from previous page)

```
    │   └── Program.cs
    └── HelloCdk.sln
```

Python Example

```
.
├── app.py
├── cdk.json
├── hello
│   ├── __init__.py
│   ├── hello_construct.py
│   └── hello_stack.py
├── package.json
├── package-lock.json
├── poetry.lock
└── pyproject.toml
```

TypeScript Example

```
.
├── bin
│   └── sample.ts
├── cdk.json
├── lib
│   └── sample-stack.ts
├── package.json
├── package.json
└── tsconfig.json
```

10.1.3 Advanced Features

Advanced features and detailed information for using CDK with Runway.

Contents

Build Steps

Build steps (e.g. for compiling TypeScript) can be specified in *deployment.module_options*/*module.options*. These steps will be run before each diff, deploy, or destroy.

Example

```yaml
deployments:
  - modules:
      - path: mycdkmodule.cdk
        environments:
          dev: true
        options:
          build_steps:
            - npx tsc
```

Disabling NPM CI

At the start of each module execution, Runway will execute `npm ci` to ensure that CDK is installed in the project (so Runway can execute it via `npx cdk`). This can be disabled (e.g. for use when the `node_modules` directory is pre-compiled) via the `skip_npm_ci` field of *deployment.module_options*/*module.options*.

Example

```yaml
deployments:
  - modules:
      - path: mycdkmodule.cdk
        options:
          skip_npm_ci: true
```

10.2 CloudFormation & Troposphere

The CloudFormation module type is deployed using Runway's CloudFormation engine (CFNgin). It is able to deploy raw CloudFormation templates (JSON & YAML) and Troposphere templates that are written in the form of a *Blueprints*.

- *Configuration*
- *Directory Structure*
- *Advanced Features*

10.2.1 Configuration

In addition to the *Runway Config File*, there are two files that can be used for configuration:

- a YAML *configuration file* **[REQUIRED]**
- a key/value *environment file*

Changed in version 1.5.0: Stacker is no longer used for handling CloudFormation/Troposphere. It has been replaced with an internal CloudFormation engin (CFNgin).

Contents

runway.yml

Example

```yaml
deployments:
  - modules:
      - path: sampleapp.cfn
        type: cloudformation  # not required; implied from ".cfn" directory extension
        environments:
          dev: true
        parameters:
          namespace: example-${env DEPLOY_ENVIRONMENT}
          cfngin_bucket: example-${env DEPLOY_ENVIRONMENT}-${env AWS_REGION}
    regions:
      - us-east-1
```

Options

CloudFormation modules do not have any module-specific options.

Parameters

Runway can pass *Parameters* to a CloudFormation module in place of or in addition to values in an *environment file*. When *Parameters* are passed to the module, the data type is retained (e.g. `array`, `boolean`, `mapping`).

A typical usage pattern would be to use *Runway Lookups* in combination with *Parameters* to pass *deploy environment* and/or region specific values to the module from the *Runway Config File*.

Example

```
deployments:
  - modules:
      - sampleapp-01.cfn
      - path: sampleapp-02.cfn
        parameters:
          instance_count: ${var instance_count.${env DEPLOY_ENVIRONMENT}}
    parameters:
      image_id: ${var image_id.%{env AWS_REGION}}
```

Common Parameters

Runway automatically makes the following commonly used *Parameters* available to CloudFormation modules.

Note: If these parameters are already being explicitly defined in `deployment.parameters`/`module.parameters` the value provided will be used instead of what would be automatically added.

`environment:` `str`

Taken from the `DEPLOY_ENVIRONMENT` environment variable. This will the be current *deploy environment*.

`region:` `str`

Taken from the `AWS_REGION` environment variable. This will be the current region being processed.

CFNgin Config File

The CFNgin config file has full support for YAML features like anchors & references for a DRY config file (See *YAML anchors & references* for details).

Top-Level Fields

class cfngin.**config**

Runway's CFNgin makes use of a YAML formatted config file to define the different CloudFormation stacks that make up a given environment.

cfngin_bucket: Optional[str] = None

By default, CloudFormation templates are pushed into an S3 bucket and CloudFormation is pointed to the template in that bucket when launching or updating stacks. By default it uses a bucket named `cfngin-${namespace}-${region}`, where the namespace is *namespace* and region is the current AWS region.

To change this, define a value for this field.

If the bucket does not exists, CFNgin will try to create it in the same region that the stacks will be launched in. The bucket will be created by deploying a CloudFormation stack named `${namespace}-cfngin-bucket` where the namespace is *namespace*. If there is a stack named `cfngin-bucket` found defined in the *stacks* field, it will be used in place of default *stack* & Blueprint (*runway.cfngin.blueprints.cfngin_bucket.CfnginBucket*) provided by CFNgin. When using a custom stack, it is the user's responsibility to ensure that a bucket with the correct name is created by this stack.

If you want CFNgin to upload templates directly to CloudFormation instead of first uploading to S3, you can set this field to an empty string. However, the template size is greatly limited when uploading directly. See the CloudFormation Limits Reference.

Tip: Defining a *stack* that uses the Blueprint provided by CFNgin allows for easy customization of stack fields such as *tags*. It also allows the stack to be deleted as part of the normal deletion process. If it is not defined as a stack, CFNgin won't delete the stack or bucket.

```yaml
namespace: ${namespace}
cfngin_bucket: cfngin-${namespace}-${region}

stacks:
  - name: cfngin-bucket
    class_path: runway.cfngin.blueprints.cfngin_bucket.CfnginBucket
    variables:
      BucketName: cfngin-${namespace}-${region}

pre_destroy:
  - path: runway.cfngin.hooks.cleanup_s3.purge_bucket
    args:
      bucket_name: cfngin-${namespace}-${region}
```

Example

```
cfngin_bucket: example-${region}
```

Changed in version 2.0.0: The format of the default value is now `cfngin-${namespace}-${region}`.

cfngin_bucket_region: Optional[str] = None

AWS Region where *cfngin_bucket* is located. This can be different than the region currently being deployed to but, ensure to account for all AWS limitations before manually setting this value.

If not provided, the current region is used.

Example

```
cfngin_bucket_region: us-east-1
```

cfngin_cache_dir: Optional[str] = ./.runway/

Path to a local directory that CFNgin will use for local caching.

If provided, the cache location is relative to the CFNgin configuration file.

If NOT provided, the cache location is relative to the `runway.yaml`/`runway.yml` file and is shared between all Runway modules.

Example

```
cfngin_cache_dir: ./.runway
```

log_formats: Optional[Dict[str, str]] = {}

Customize log message formatting by log level.

Any of the standard Python logging module format attributes can be used when writing a new log format string.

Example

```
log_formats:
  info: "[%(asctime)s] %(message)s"
  debug: "[%(asctime)s] %(levelname)s %(threadName)s %(name)s:%(lineno)d(
↪%(funcName)s): %(message)s"
```

lookups: Optional[Dict[str, str]] = {}

Lookups allow you to create custom methods which take a value and are resolved at runtime time. The resolved values are passed to the *Blueprint* before it is rendered. For more information, see the *Lookups* documentation.

CFNgin provides some common *Lookups*, but it is sometimes useful to have your own custom lookup that doesn't get shipped with Runway. You can register your own lookups here.

The *key* of each item in the mapping will be used as the name of the lookup type when registering the lookup. The *value* should be the path to a valid lookup handler.

Example

```
lookups:
  custom: path.to.lookup.handler

conf_value: ${custom query}
```

mappings: `Optional[Dict[str, Dict[str, Dict[str, Any]]]] = {}`

Mappings are dictionaries that are provided as Mappings to each CloudFormation stack that CFNgin produces.

These can be useful for providing things like different AMIs for different instance types in different regions.

These can be used in each *Blueprint*/template as usual.

Example

```
mappings:
  AmiMap:
    us-east-1:
      NAT: ami-ad227cc4
      ubuntu1404: ami-74e27e1c
      bastion: ami-74e27e1c
    us-west-2:
      NAT: ami-290f4119
      ubuntu1404: ami-5189a661
      bastion: ami-5189a661
```

namespace: `str`

A *namespace* to create all stacks within. The value will be used as a prefix for the name of any stack that is created.

In addition, this value can be used to create an S3 bucket that will be used to upload and store all CloudFormation templates. See *cfngin_bucket* for more detailed information.

In general, this is paired with the concept of *deploy environments* to create a namespace per environment.

Example

```
namespace: ${namespace}-${environment}
```

namespace_delimiter: `Optional[str] = "-"`

By default, - will be used as a delimiter between the *namespace* and the declared stack name to deploy the actual CloudFormation stack name that gets created.

If you prefer to not use a delimiter, an empty string can be used as the value of this field.

See the CloudFormation API Reference for allowed stack name characters

Example

```
namespace_delimiter: ""
```

package_sources: Optional[*cfngin.package_sources*] = {}

See *Remote Sources* for detailed information.

```
package_sources:
  git:
    ...
  local:
    ...
  s3:
    ...
```

persistent_graph_key: Optional[str] = None

Used to track the *state* of stacks defined in configuration file. This can result in stacks being destroyed when they are removed from the configuration file removing the need to manually delete the stacks.

See *Persistent Graph* for detailed information.

Example

```
persistent_graph_key: unique-key.json
```

post_deploy: Optional[List[*cfngin.hook*]] = []

Python functions/methods that are executed after processing the stacks in the config while using the *deploy command*.

See *Hooks* for more detailed information.

Example

```
post_deploy:
  - path: do.something
```

Changed in version 2.0.0: *post_build* renamed to *post_deploy*.

Changed in version 2.2.0: The CFNgin bucket is now created using a CloudFormation stack.

post_destroy: Optional[List[*cfngin.hook*]] = []

Python functions/methods that are executed after processing the stacks in the config while using the *destroy command*.

See *Hooks* for more detailed information.

Example

```
post_destroy:
  - path: do.something
```

pre_deploy: Optional[List[*cfngin.hook*]] = []

Python functions/methods that are executed before processing the stacks in the config while using the *deploy command*.

See *Hooks* for more detailed information.

Example

```
pre_deploy:
  - path: do.something
```

Changed in version 2.0.0: *pre_build* renamed to *pre_deploy*.

pre_destroy: Optional[List[*cfngin.hook*]] = []

Python functions/methods that are executed before processing the stacks in the config while using the *destroy command*.

See *Hooks* for more detailed information.

Example

```
pre_destroy:
  - path: do.something
```

service_role: Optional[str] = None

By default CFNgin doesn't specify a service role when executing changes to CloudFormation stacks. If you would prefer that it do so, you define the IAM Role ARN that CFNgin should use when executing CloudFormation changes.

This is the equivalent of setting `RoleARN` on a call to the following CloudFormation API calls: `CreateStack`, `UpdateStack`, `CreateChangeSet`.

See the AWS CloudFormation service role for more information.

Example

```
service_role: arn:aws:iam::123456789012:role/name
```

stacks: Optional[List[*cfngin.stack*]] = []

This is the core part of the config where the CloudFormations stacks that will be deployed in the environment are defined.

See *Stack* for more information.

sys_path: Optional[str] = None

A path to be added to `$PATH` while processing the configuration file. This will allow modules from the provided path location to be used.

When setting *class_path* for a *Blueprint* or *path* for a *hook* , it is sometimes desirable to load modules from outside the default `$PATH` (e.g. to include modules inside the same repo as config files).

Example

```
sys_path: ./  # most common value to use
```

tags: Optional[Dict[str, str]] = {"cfngin_namespace": namespace}

A dictionary of tags to add to all stacks. These tags are propagated to all resources that AWS CloudFormation supports. See CloudFormation - Resource Tag for more information.

If this field is undefined, a **cfngin_namespace** tag is applied to your stack with the value of *namespace* as the tag-value. Alternatively, this field can be set to a value of `{}` (an empty dictionary) to disable the default tag.

Example

```
tags:
  namespace: ${namespace}
  example: value
```

Listing 1: disable default tag

```
tags: {}
```

template_indent: Optional[int] = 4

Number of spaces per indentation level to use when rendering/outputting CloudFormation templates.

Example

```
template_indent: 2
```

Stack

class cfngin.**stack**

Defines a CloudFormation stack.

Lookup Support

The following fields support lookups:

- *variables*

Example

```yaml
stacks:
  - name: vpc-example
    class_path: blueprints.vpc.VPC
    variables:
      InstanceType: t2.small
      SshKeyName: default
      ImageName: NAT
      AZCount: 2
      PublicSubnets:
        - 10.128.0.0/24
        - 10.128.1.0/24
        - 10.128.2.0/24
        - 10.128.3.0/24
      PrivateSubnets:
        - 10.128.8.0/22
        - 10.128.12.0/22
        - 10.128.16.0/22
        - 10.128.20.0/22
      CidrBlock: 10.128.0.0/16
```

class_path: Optional[str] = None

A python importable path to the *Blueprint* class to be used.

Exactly one of *class_path* or *template_path* must be defined.

Example

```yaml
stacks:
  - name: example-stack
    class_path: example.BlueprintClass
```

description: Optional[str] = None

A short description to apply to the stack. This overwrites any description defined in the *Blueprint*. See Cloudformation - Template Description for more information.

Example

```yaml
stacks:
  - name: example-stack
    description: An Example Stack
```

enabled: Optional[bool] = True

Whether to deploy/update the stack. This enables the ability to disable stacks in different environments.

Important: This field is ignored when destroying stacks.

Example

```
stacks:
  - name: example-stack
    enabled: false
  - name: another-stack
    enabled: ${enable_another_stack}
```

in_progress_behavior: Optional[Literal['wait']] = None

Specifies the behavior for when a stack is in `CREATE_IN_PROGRESS` or `UPDATE_IN_PROGRESS`. By default, CFNgin will raise an exception if the stack is in an `IN_PROGRESS` state when processing begins.

If the value of this field is *wait*, CFNgin will wait for the previous update to complete before attempting to update the stack instead of raising an exception.

Example

```
stacks:
  - name: example-stack
    in_progress_behavior: wait
```

locked: Optional[bool] = False

Whether the stack should be updated after initial deployment. This is useful for *risky* stacks that you don't want to take the risk of allowing CloudFormation to update but still want to deploy it using CFNgin.

Example

```
stacks:
  - name: example-stack
    locked: true
  - name: another-stack
    locked: ${locked_another_stack}
```

name: str

Name of the CFNgin Stack. The value of this field is used by CFNgin when referring to a Stack. It will also be used as the name of the Stack when created in CloudFormation unless overridden by `stack_name`.

Note: *namespace* will be appended to this value when used as the name of the CloudFormation Stack.

Example

```
stacks:
  - name: example-stack
```

protected: Optional[bool] = False

Whether to force all updates to be performed interactively.

When true and running in non-interactive mode, CFNgin will switch to interactive mode for this stack to require manual review and approval of any changes.

Example

```yaml
stacks:
  - name: example-stack
    protected: true
  - name: another-stack
    protected: ${protected_another_stack}
```

required_by: Optional[List[str]] = []

A list of other stacks that require this stack. All stacks must be defined in the same configuration file.

Inverse of *requires*.

Example

```yaml
stacks:
  - name: example-stack:  # deployed first
    required_by:
      - another-stack
  - name: another-stack:  # deployed after example-stack
    ...
```

requires: Optional[List[str]] = []

A list of other stacks that this stack requires. All stacks must be defined in the same configuration file.

Inverse of *required_by*.

Example

```yaml
stacks:
  - name: example-stack# deployed after another-stack
    requires:
      - another-stack
  - name: another-stack  # deployed first
    ...
```

stack_name: Optional[str] = None

The name used when creating the CloudFormation stack. If not provided, `name` will be used.

Note: *namespace* will be appended to this value.

Example

```
stacks:
  - name: example-stack
    stack_name: another-name
```

stack_policy_path: Optional[str] = None

Path to a JSON formatted stack policy that will be applied when the CloudFormation stack is created and/or updated.

See CloudFormation - Prevent updates to stack resources for examples and more information.

Example

```
stacks:
  - name: example-stack
    stack_policy_path: ./stack_policies/example-stack.json
```

tags: Optional[Dict[str, str]] = {}

A dictionary of tags to add to the Stack. These tags are propagated to all resources that AWS CloudFormation supports. See CloudFormation - Resource Tag for more information.

This will be combined with the global *tags*. Values defined here take precedence over those defined globally.

Example

```
stacks:
  - name: example-stack
    tags:
      namespace: ${namespace}
      example: value
```

template_path: Optional[str]

Path to a raw CloudFormation template (JSON or YAML). Can be relative to the working directory (e.g. templates stored alongside the configuration file), or relative to a directory in the *$PATH* (i.e. for loading templates retrieved via *package_sources*).

Exactly one of *class_path* or *template_path* must be provided.

Example

```
stacks:
  - name: example-stack
    template_path: ./templates/example-stack.yml
  - name: another-stack
    template_path: remote/path/templates/another-stack.json
```

termination_protection: Optional[bool] = False

Whether the stack will be protected from termination by CloudFormation.

Any attempts to destroy the stack (using Runway, the AWS console, AWS API, etc) will be prevented unless manually disabled.

When updating a stack and the value has been changed, termination protection on the CloudFormation stack sill also change. This is useful when needing to destroy a stack by first changing the value in the configuration file, updating the stack, then proceeding to destroy it.

Example

```yaml
stacks:
  - name: example-stack
    termination_protection: true
  - name: another-stack
    termination_protection: ${termination_protection_another_stack}
```

timeout: Optional[int] = None

Specifies the amount of time, in minutes, that CloudFormation should allot before timing out stack creation operations. If CloudFormation can't create the entire stack in the time allotted, it fails the stack creation due to timeout and rolls back the stack.

By default, there is no timeout for stack creation. However, individual resources may have their own timeouts based on the nature of the service they implement. For example, if an individual resource in your stack times out, stack creation also times out even if the timeout you specified for stack creation hasn't yet been reached.

Example

```yaml
stacks:
  - name: example-stack
    timeout: 120
```

variables: Optional[Dict[str, Any]] = {}

A dictionary of *Variables* to pass to the *Blueprint* when rendering the CloudFormation template. Can be any valid YAML data structure.

When using a raw CloudFormation template, these are the values provided for it's *Parameters*.

Example

```yaml
stacks:
  - name: example-stack
    variables:
      StackVariable: value
```

Variables

Variables are values that will be passed into a *Blueprint* before it is rendered. Variables can be any valid YAML data structure and can leverage *Lookups* to expand values at runtime.

YAML anchors & references

If you have a common set of variables that you need to pass around in many places, it can be annoying to have to copy and paste them in multiple places. Instead, using a feature of YAML known as anchors & references, you can define common values in a single place and then refer to them with a simple syntax.

For example, say you pass a common domain name to each of your stacks, each of them taking it as a Variable. Rather than having to enter the domain into each stack you could do the following to have an anchor called **domain** that you can use in place of any value in the config to provide the value **mydomain.com**.

```
stacks:
- name: example-stack
  class_path: blueprints.Example
  variables:
    DomainName: &domain mydomain.com
  - name: vpc
    class_path: blueprints.VPC
    variables:
      DomainName: *domain
```

Even more powerful is the ability to anchor entire dictionaries, and then reference them in another dictionary, effectively providing it with default values. Now, rather than having to provide each of those variables to every stack that could use them, you can just do this instead.

```
stacks:
  - name: example-stack
    class_path: blueprints.Example
    variables: &variables
      DomainName: mydomain.com
      InstanceType: m3.medium
      AMI: ami-12345abc
  - name: vpc
    class_path: blueprints.VPC
    variables:
      << : *variables
      InstanceType: c4.xlarge # override the InstanceType in this stack
```

Using Outputs as Variables

Since CFNgin encourages the breaking up of your CloudFormation stacks into entirely separate stacks, sometimes you'll need to pass values from one stack to another. The way this is handled in CFNgin is by having one stack provide *Outputs* for all the values that another stack may need, and then using those as the inputs for another stack's *variables*. CFNgin makes this easier for you by providing a syntax for *variables* that will cause CFNgin to automatically look up the values of *Outputs* from another stack in its config.

To do so, use the *output* in the *variables* on the target stack.

```
MyParameter: ${output OtherStack.OutputName}
```

For more information see *Lookups*.

In this example config, when deploying things inside a VPC, you will need to pass the **VpcId** of the VPC that you want the resources to be located in. If the **vpc** stack provides an Output called **VpcId**, you can reference it easily.

```
domain_name: my_domain &domain

stacks:
  - name: vpc
    class_path: blueprints.vpc.VPC
    variables:
      DomainName: *domain
  - name: webservers
    class_path: blueprints.asg.AutoscalingGroup
    variables:
      DomainName: *domain
      VpcId: ${output vpc.VpcId} # gets the VpcId Output from the vpc stack
```

Doing this creates an implicit dependency from the **webservers** stack to the **vpc** stack, which will cause CFNgin to submit the **vpc** stack, and then wait until it is complete until it submits the **webservers** stack. This would be the same as adding **vpc** to the *requires* field of the **webservers** stack.

Environment File

When using CFNgin, you can optionally provide an "environment" file. The CFNgin config file will be interpolated as a string.Template using the key-value pairs from the environment file as *parameters*. The format of the file is a single key-value per line, separated by a colon (**:**).

File Naming

Environment files must follow a specific naming format in order to be recognized by Runway. The files must also be stored at the root of the module's directory.

${DEPLOY_ENVIRONMENT}-${AWS_REGION}.env The typical naming format that will be used for these files specifies the name of the `DEPLOY_ENVIRONMENT` and `AWS_REGION` in which to use the file.

${DEPLOY_ENVIRONMENT}.env The region can optionally be omitted to apply a single file to all regions.

Files following both naming schemes may be used. The file with the most specific name takes precedence. Values passed in as `parameters` from the *Runway Config File* take precedence over those provided in an environment file.

Usage

A pretty common use case is to have separate environments that you want to look mostly the same, though with some slight modifications. For example, you might want a **production** and a **staging** environment.

The production environment likely needs more instances, and often those instances will be of a larger instance type. The parameters defined in an environment file, *deployment.parameters*, and/or *module.parameters* allow you to use your existing CFNgin config, but provide different values based on the current *deploy environment*.

Example

```
vpcID: vpc-12345678
```

Provided the key-value pair above, you will now be able to use this in your configs for a *deploy environment*. They act as keys that can be used in your config file, providing a sort of templating ability. This allows you to change the values of your config based on the current *deploy environment*.

For example, if you have a **webserver** stack, and you need to provide it a variable for the instance size it should use, you would have something like this in your config file.

```
stacks:
  - name: webservers
    class_path: blueprints.asg.AutoscalingGroup
    variables:
      InstanceType: m3.medium
```

But what if you needed more CPU in your production environment, but not in your staging? Without parameters, you'd need a separate config for each. With parameters, you can simply define two different values for **InstanceType** in an an environment file, *deployment.parameters*, and/or *module.parameters* then use the parameter's name to reference the value in a config file.

Listing 2: sampleapp.cfn/cfngin.yml

```
# in your config file:
stacks:
  - name: webservers:
    class_path: blueprints.asg.AutoscalingGroup
    variables:
      InstanceType: ${web_instance_type}
```

Using Environment Files

Both files would be required.

Listing 3: sampleapp.cfn/stage.env

```
web_instance_type: m5.medium
```

Listing 4: sampleapp.cfn/prod.env

```
web_instance_type: c5.xlarge
```

Using Runway

This option would not required the use of environment files to define the values.

Listing 5: runway.yaml

```yaml
deployments:
  - modules:
    - name: Sample Application
      path: sampleapp.cfn
      parameters:
        web_instance_type: ${var web_instance_type.${env DEPLOY_ENVIRONMENT}}

variables:
  web_instance_type:
    stage: m5.medium
    prod: c5.xlarge
```

10.2.2 Directory Structure

Example directory structures for a CloudFormation module.

Contents

Blueprints

Important: Blueprints must be importable by python. (e.g. directory contains `__init__.py`)

```
.
├── poetry.lock
├── pyproject.toml
├── runway.variables.yml
├── runway.yml
└── sampleapp.cfn
    ├── blueprints
    │   ├── __init__.py
    │   └── my_blueprint.py
    ├── dev-us-east-1.env
    └── cfngin.yml
```

Cloudformation Templates

Important: CloudFormation templates can't be stored in the root of the module directory. They must be in a subdirectory or external to the module.

```
.
├── poetry.lock
├── pyproject.toml
├── runway.variables.yml
├── runway.yml
└── sampleapp.cfn
    ├── templates
    │   ├── template-01.yml
    │   └── template-02.json
    ├── dev-us-east-1.env
    ├── 01-cfngin.yml
    └── 02-cfngin.yml
```

10.2.3 Advanced Features

Advanced features and detailed information for using Runway's CFNgin for CloudFormation modules.

Blueprints

A *Blueprint* is a python classes that dynamically builds CloudFormation templates. Where you would specify a raw Cloudformation template in a *stack* using the *template_path* key, you instead specify a *Blueprint* subclass using the *class_path* key.

Traditionally Blueprints are built using troposphere, but that is not absolutely necessary.

Making your own should be easy, and you can take a lot of examples from Runway blueprints. In the end, all that is required is that the *Blueprint* is a subclass of *runway.cfngin.blueprints.base.Blueprint* and it has the following method overridden:

```
# Updates self.template to create the actual template
def create_template(self) -> None:
    """Create a template from the blueprint.

    Main method called by CFNgin when rendering a Blueprint into a template
    that is expected to be overridden.

    """
```

Contents

Variables

A *Blueprint* can define a *VARIABLES* ClassVar that defines the variables it accepts from the *Config Variables*.

VARIABLES should be a Dict of <variable name>: <variable definition>. The variable definition should be a *BlueprintVariableTypeDef*.

Example

```
from __future__ import annotations

from typing import TYPE_CHECKING, ClassVar, Dict

from runway.cfngin.blueprints.base import Blueprint

if TYPE_CHECKING:
    from runway.cfngin.blueprints.type_defs import BlueprintVariableTypeDef


class ExampleClass(Blueprint):
    """Example Blueprint."""
    VARIABLES: ClassVar[Dict[str, BlueprintVariableTypeDef]] = {
        "ExampleVariable": {
            "default": "",
            "description": "Example variable.",
            "type": str,
        }
    }
```

See also:

runway.cfngin.blueprints.type_defs.BlueprintVariableTypeDef Documentation for the contents of a *Blueprint* variable definition.

Variable Types

Any native python type can be specified as the *type* for a variable. You can also use the following custom types:

TroposphereType

The *TroposphereType* can be used to generate resources for use in the *Blueprint* directly from user-specified configuration. Which of the below case applies depends on what `defined_type` was chosen, and how it would be normally used in the *Blueprint* (and CloudFormation in general).

Resource Types

When `defined_type` is a Resource Type, the value specified by the user in the configuration file must be a dictionary, but with two possible structures.

When `many` is disabled, the top-level dictionary keys correspond to parameters of the `defined_type` constructor. The key-value pairs will be used directly, and one object will be created and stored in the variable.

When `many` is enabled, the top-level dictionary *keys* are resource titles, and the corresponding *values* are themselves dictionaries, to be used as parameters for creating each of multiple `defined_type` objects. A list of those objects will be stored in the variable.

Property Types

When `defined_type` is a property type the value specified by the user in the configuration file must be a dictionary or a list of dictionaries.

When `many` is disabled, the top-level dictionary keys correspond to parameters of the `defined_type` constructor. The key-value pairs will be used directly, and one object will be created and stored in the variable.

When `many` is enabled, a list of dictionaries is expected. For each element, one corresponding call will be made to the `defined_type` constructor, and all the objects produced will be stored (also as a list) in the variable.

Optional variables

In either case, when `optional` is enabled, the variable may have no value assigned, or be explicitly assigned a null value. When that happens the variable's final value will be `None`.

Example

Below is an annotated example:

```
"""Example Blueprint."""
from __future__ import annotations

from typing import TYPE_CHECKING, ClassVar, Dict

from troposphere import s3, sns

from runway.cfngin.blueprints.base import Blueprint
```

(continues on next page)

(continued from previous page)

```
from runway.cfngin.blueprints.variables.types import TroposphereType

if TYPE_CHECKING:
    from runway.cfngin.blueprints.type_defs import BlueprintVariableTypeDef


class Buckets(Blueprint):
    """S3 Buckets."""

    VARIABLES: ClassVar[Dict[str, BlueprintVariableTypeDef]] = {
        # Specify that Buckets will be a list of s3.Bucket types.
        # This means the config should a dictionary of dictionaries
        # which will be converted into troposphere buckets.
        "Buckets": {
            "type": TroposphereType(s3.Bucket, many=True),
            "description": "S3 Buckets to create.",
        },
        # Specify that only a single bucket can be passed.
        "SingleBucket": {
            "type": TroposphereType(s3.Bucket),
            "description": "A single S3 bucket",
        },
        # Specify that Subscriptions will be a list of sns.Subscription types.
        # Note: sns.Subscription is the property type, not the standalone
        # sns.SubscriptionResource.
        "Subscriptions": {
            "type": TroposphereType(sns.Subscription, many=True),
            "description": "Multiple SNS subscription designations",
        },
        # Specify that only a single subscription can be passed, and that it
        # is made optional.
        "SingleOptionalSubscription": {
            "type": TroposphereType(sns.Subscription, optional=True),
            "description": "A single, optional SNS subscription designation",
        },
    }

    def create_template(self) -> None:
        """Create a template from the blueprint."""
        # The Troposphere s3 buckets have already been created when we
        # access self.variables["Buckets"], we just need to add them as
        # resources to the template.
        for bucket in self.variables["Buckets"]:
            self.template.add_resource(bucket)

        # Add the single bucket to the template. You can use
        # `Ref(single_bucket)` to pass CloudFormation references to the
        # bucket just as you would with any other Troposphere type.
        self.template.add_resource(self.variables["SingleBucket"])

        subscriptions = self.variables["Subscriptions"]
        optional_subscription = self.variables["SingleOptionalSubscription"]
```

(continues on next page)

(continued from previous page)

```
        # Handle it in some special way...
        if optional_subscription is not None:
            subscriptions.append(optional_subscription)

        self.template.add_resource(
            sns.Topic("ExampleTopic", TopicName="Example", Subscriptions=subscriptions)
        )
```

A sample config for the above:

```
stacks:
  - name: buckets
    class_path: path.to.above.Buckets
    variables:
      Buckets:
        # resource name (title) that will be added to CloudFormation.
        FirstBucket:
          # name of the s3 bucket
          BucketName: my-first-bucket
        SecondBucket:
          BucketName: my-second-bucket
      SingleBucket:
        # resource name (title) that will be added to CloudFormation.
        MySingleBucket:
          BucketName: my-single-bucket
      Subscriptions:
        - Endpoint: one-lambda
          Protocol: lambda
        - Endpoint: another-lambda
          Protocol: lambda
      # The following could be omitted entirely
      SingleOptionalSubscription:
        Endpoint: a-third-lambda
        Protocol: lambda
```

CFNType

The *CFNType* can be used to signal that a variable should be submitted to CloudFormation as a Parameter instead of only available to the *Blueprint* when rendering. This is useful if you want to leverage AWS-Specific Parameter types (e.g. `List<AWS::EC2::Image::Id>`) or Systems Manager Parameter Store values (e.g. `AWS::SSM::Parameter::Value<String>`).

See *runway.cfngin.blueprints.variables.types* for available subclasses of the *CFNType*.

Example

```python
"""Example Blueprint."""
from __future__ import annotations

from typing import TYPE_CHECKING, ClassVar, Dict

from runway.cfngin.blueprints.base import Blueprint
from runway.cfngin.blueprints.variables.types import (
    CFNString,
    EC2AvailabilityZoneNameList,
)

if TYPE_CHECKING:
    from runway.cfngin.blueprints.type_defs import BlueprintVariableTypeDef


class ExampleBlueprint(Blueprint):
    """Example Blueprint."""

    VARIABLES: ClassVar[Dict[str, BlueprintVariableTypeDef]] = {
        "String": {"type": str, "description": "Simple string variable"},
        "List": {"type": list, "description": "Simple list variable"},
        "CloudFormationString": {
            "type": CFNString,
            "description": "A variable which will create a CloudFormation "
            "Parameter of type String",
        },
        "CloudFormationSpecificType": {
            "type": EC2AvailabilityZoneNameList,
            "description": "A variable which will create a CloudFormation "
            "Parameter of type List<AWS::EC2::AvailabilityZone::Name>",
        },
    }

    def create_template(self) -> None:
        """Create a template from the blueprint."""
        # `self.variables` returns a dictionary of <variable name>: <variable value>.
        # For the subclasses of `CFNType`, the values are
        # instances of `CFNParameter` which have a `ref` helper property
        # which will return a troposphere `Ref` to the parameter name.
        self.add_output("StringOutput", self.variables["String"])

        # self.variables["List"] is a native list
        for index, value in enumerate(self.variables["List"]):
            self.add_output("ListOutput:{}".format(index), value)

        # `CFNParameter` values (which wrap variables with a `type`
        # that is a `CFNType` subclass) can be converted to troposphere
        # `Ref` objects with the `ref` property
        self.add_output(
            "CloudFormationStringOutput", self.variables["CloudFormationString"].ref
        )
```

(continues on next page)

(continued from previous page)

```
        self.add_output(
            "CloudFormationSpecificTypeOutput",
            self.variables["CloudFormationSpecificType"].ref,
        )
```

Utilizing Stack name within your Blueprint

Sometimes your *Blueprint* might want to utilize the already existing *stack.name* within your *Blueprint*. Runway's CFNgin provides access to both the fully qualified stack name matching what's shown in the CloudFormation console, in addition to the stack's short name you have set in your YAML config.

Referencing Fully Qualified Stack name

The fully qualified name is a combination of the CFNgin namespace + the short name (what you set as `name` in your YAML config file). If your CFNgin *namespace* is `CFNginIsCool` and the stack's short name is `myAwesomeEC2Instance`, the fully qualified name would be `CFNginIsCool-myAwesomeEC2Instance`.

To use this in your *Blueprint*, you can get the name from context using `self.context.get_fqn(self.name)`.

Referencing the Stack short name

The *Stack* short name is the name you specified for the *stack* within your YAML config. It does not include the *namespace*. If your CFNgin namespace is `CFNginIsCool` and the stack's short name is `myAwesomeEC2Instance`, the short name would be `myAwesomeEC2Instance`.

To use this in your *Blueprint*, you can get the name from the *name* attribute.

Example

```
"""Example Blueprint."""
from __future__ import annotations

from typing import TYPE_CHECKING, ClassVar, Dict

from troposphere import Tags, ec2

from runway.cfngin.blueprints.base import Blueprint
from runway.cfngin.blueprints.variables.types import CFNString

if TYPE_CHECKING:
    from runway.cfngin.blueprints.type_defs import BlueprintVariableTypeDef


class ExampleBlueprint(Blueprint):
    """Example Blueprint."""

    # VpcId set here to allow for Blueprint to be reused
    VARIABLES: ClassVar[Dict[str, BlueprintVariableTypeDef]] = {
```

(continues on next page)

(continued from previous page)

```
        "VpcId": {
            "type": CFNString,
            "description": "The VPC to create the Security group in",
        }
    }

    def create_template(self) -> None:
        """Create a template from the blueprint."""
        # now adding a SecurityGroup resource named `SecurityGroup` to the CFN template
        self.template.add_resource(
            ec2.SecurityGroup(
                "SecurityGroup",
                # Referencing the VpcId set as the variable
                VpcId=self.variables["VpcId"].ref,
                # Setting the group description as the fully qualified name
                GroupDescription=self.context.get_fqn(self.name),
                # setting the Name tag to be the stack short name
                Tags=Tags(Name=self.name),
            )
        )
```

Testing Blueprints

When writing your own *Blueprint* it is useful to write tests for them in order to make sure they behave the way you expect they would, especially if there is any complex logic inside.

To this end, a sub-class of the `unittest.TestCase` class has been provided: *runway.cfngin.blueprints.testutil.BlueprintTestCase*. You use it like the regular TestCase class, but it comes with an addition assertion: `assertRenderedBlueprint`. This assertion takes a *Blueprint* object and renders it, then compares it to an expected output, usually in `tests/fixtures/blueprints`.

Yaml (CFNgin) format tests

In order to wrap the *BlueprintTestCase* tests in a format similar to CFNgin's stack format, the *YamlDirTestGenerator* class is provided. When subclassed in a directory, it will search for yaml files in that directory with certain structure and execute a test case for it.

Example

```
namespace: test

stacks:
  - name: test_stack
    class_path: cfngin_blueprints.s3.Buckets
    variables:
      var1: val1
```

When run from tests, this will create a template fixture file called `test_stack.json` containing the output from the `cfngin_blueprints.s3.Buckets` template.

Hooks

A *hook* is a python function, class, or class method that is executed before or after an action is taken for the entire config.

Only the following actions allow pre/post hooks:

deploy using fields *pre_deploy* and *post_deploy*

destroy using fields *pre_destroy* and *post_destroy*

class cfngin.**hook**

When defining a hook in one of the supported fields, the follow fields can be used.

Lookup Support

The following fields support lookups:

- *args*

args: Optional[Dict[str, Any]] = {}

A dictionary of arguments to pass to the hook.

This field supports the use of *lookups*.

Important: *Lookups* that change the order of execution, like *output*, can only be used in a *post* hook but hooks like *rxref* are able to be used with either *pre* or *post* hooks.

Example

```
pre_deploy:
  - args:
      key: ${val}
```

data_key: Optional[str] = None

If set, and the hook returns data (a dictionary or `pydantic.BaseModel`), the results will be stored in *CfnginContext.hook_data* with the `data_key` as its key.

Example

```
pre_deploy:
  - data_key: example-key
```

enabled: Optional[bool] = True

Whether to execute the hook every CFNgin run. This field provides the ability to execute a hook per environment when combined with a variable.

Example

```
pre_deploy:
  - enabled: ${enable_example_hook}
```

path: `str`

Python importable path to the hook.

Example

```
pre_deploy:
  - path: runway.cfngin.hooks.command.run_command
```

required: `Optional[bool] = True`

Whether to stop execution if the hook fails.

Contents

Built-in Hooks

acm.Certificate

Hook Path *runway.cfngin.hooks.acm.Certificate*

Manage a DNS validated certificate in AWS Certificate Manager.

When used in the *pre_deploy* or *post_deploy* stage this hook will create a CloudFormation stack containing a DNS validated certificate. It will automatically create a record in Route 53 to validate the certificate and wait for the stack to complete before returning the `CertificateArn` as hook data. The CloudFormation stack also outputs the ARN of the certificate as `CertificateArn` so that it can be referenced from other stacks.

When used in the *pre_destroy* or *post_destroy* stage this hook will delete the validation record from Route 53 then destroy the stack created during a deploy stage.

If the hook fails during a deploy stage (e.g. stack rolls back or Route 53 can't be updated) all resources managed by this hook will be destroyed. This is done to avoid orphaning resources/record sets which would cause errors during subsequent runs. Resources effected include the CloudFormation stack it creates, ACM certificate, and Route 53 validation record.

New in version 1.6.0.

Requirements

- Route 53 hosted zone
  - authoritative for the domain the certificate is being created for
  - in the same AWS account as the certificate being created

Args

alt_names: Optional[List[str]] = []

Additional FQDNs to be included in the Subject Alternative Name extension of the ACM certificate. For example, you can add *www.example.net* to a certificate for which the `domain` field is *www.example.com* if users can reach your site by using either name.

domain: str

The fully qualified domain name (FQDN), such as *www.example.com*, with which you want to secure an ACM certificate. Use an asterisk (*) to create a wildcard certificate that protects several sites in the same domain. For example, **.example.com* protects *www.example.com*, *site.example.com*, and *images.example.com*.

hosted_zone_id: str

The ID of the Route 53 Hosted Zone that contains the resource record sets that you want to change. This must exist in the same account that the certificate will be created in.

stack_name: Optional[str] = None

Provide a name for the stack used to create the certificate. If not provided, the domain is used (replacing `.` with `-`). If the is provided in a deploy stage, its needs to be provided in the matching destroy stage.

ttl: Optional[int] = None

The resource record cache time to live (TTL), in seconds. (*default:* `300`)

Example

```
namespace: example
cfngin_bucket: ''

sys_path: ./

pre_deploy:
  acm-cert:
    path: runway.cfngin.hooks.acm.Certificate
    required: true
    args:
      domain: www.example.com
      hosted_zone_id: ${rxref example-com::HostedZone}

stack:
  sampleapp:
    class_path: blueprints.sampleapp.BlueprintClass
    variables:
      cert_arn: ${rxref www-example-com::CertificateArn}
```

(continues on next page)

(continued from previous page)

```
post_destroy:
  acm-cert:
    path: runway.cfngin.hooks.acm.Certificate
    required: true
    args:
      domain: www.example.com
      hosted_zone_id: ${rxref example-com::HostedZone}
```

aws_lambda.upload_lambda_functions

Deprecated since version 2.5.0: Replaced by *awslambda.PythonFunction*.

Hook Path `runway.cfngin.hooks.aws_lambda.upload_lambda_functions`

Build Lambda payloads from user configuration and upload them to S3 using the following process:

1. Constructs ZIP archives containing files matching specified patterns for each function.
2. Uploads the result to Amazon S3
3. Returns a `Dict` of "*function name*: `troposphere.awslambda.Code`".

The returned value can retrieved using the *hook_data Lookup* or by interacting with the `CfnginContext` object passed to the `Blueprint`.

Configuration consists of some global options, and a dictionary of function specifications. In the specifications, each key indicating the name of the function (used for generating names for artifacts), and the value determines what files to include in the ZIP (see more details below).

If a `requirements.txt` file or `Pipfile/Pipfile.lock` files are found at the root of the provided `path`, the hook will use the appropriate method to package dependencies with your source code automatically. If you want to explicitly use `pipenv` over `pip`, provide `use_pipenv: true` for the function.

Docker can be used to collect python dependencies instead of using system python to build appropriate binaries for Lambda. This can be done by including the `dockerize_pip` configuration option which can have a value of `true` or `non-linux`.

Payloads are uploaded to either the `cfngin_bucket` or an explicitly specified bucket, with the object key containing it's checksum to allow repeated uploads to be skipped in subsequent runs.

Args

`bucket: Optional[str] = None`
Custom bucket to upload functions to. If not provided, `cfngin_bucket` will be used.

`bucket_region: Optional[str] = None`
The region in which the bucket should exist. If not provided, `cfngin_bucket_region` will be used.

`prefix: Optional[str] = None`
S3 key prefix to prepend to the uploaded zip name.

`follow_symlinks: Optional[bool] = False`
Whether symlinks should be followed and included with the zip artifact.

`payload_acl: runway.cfngin.hooks.aws_lambda.PayloadAclTypeDef = private`
The canned S3 object ACL to be applied to the uploaded payload.

functions: Dict[str, Any]

Configurations of desired payloads to build. Keys correspond to function names, used to derive key names for the payload. Each value should itself be a dictionary, with the following data:

docker_file: Optional[str] = None

Path to a local DockerFile that will be built and used for `dockerize_pip`. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

docker_image: Optional[str] = None

Custom Docker image to use with `dockerize_pip`. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

dockerize_pip: Optional[Union[bool, Literal['non-linux']]] = None

Whether to use Docker when restoring dependencies with pip. Can be set to `true`/`false` or the special string `non-linux` which will only run on non Linux systems. To use this option Docker must be installed.

exclude: Optional[Union[List[str], str]] = None

Pattern or list of patterns of files to exclude from the payload. If provided, any files that match will be ignored, regardless of whether they match an inclusion pattern.

Commonly ignored files are already excluded by default, such as `.git`, `.svn`, `__pycache__`, `*.pyc`, `.gitignore`, etc.

include: Optional[List[str], str] = None

Pattern or list of patterns of files to include in the payload. If provided, only files that match these patterns will be included in the payload.

Omitting it is equivalent to accepting all files that are not otherwise excluded.

path: str

Base directory of the Lambda function payload content. If it not an absolute path, it will be considered relative to the directory containing the CFNgin configuration file in use.

Files in this directory will be added to the payload ZIP, according to the include and exclude patterns. If no patterns are provided, all files in this directory (respecting default exclusions) will be used.

Files are stored in the archive with path names relative to this directory. So, for example, all the files contained directly under this directory will be added to the root of the ZIP file.

python_path: Optional[str] = None

Absolute path to a python interpreter to use for `pip`/`pipenv` actions. If not provided, the current python interpreter will be used for `pip` and `pipenv` will be used from the current `$PATH`.

runtime: Optional[str] = None

Runtime of the AWS Lambda Function being uploaded. Used with `dockerize_pip` to automatically select the appropriate Docker image to use. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

use_pipenv: Optional[bool] = False

Will determine if pipenv will be used to generate requirements.txt from an existing Pipfile. To use this option pipenv must be installed.

Example

Listing 6: Hook Configuration

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.aws_lambda.upload_lambda_functions
    required: true
    enabled: true
    data_key: lambda
    args:
      bucket: custom-bucket
      follow_symlinks: true
      prefix: cloudformation-custom-resources/
      payload_acl: authenticated-read
      functions:
        MyFunction:
          path: ./lambda_functions
          dockerize_pip: non-linux
          use_pipenv: true
          runtime: python3.9
          include:
            - '*.py'
            - '*.txt'
          exclude:
            - '*.pyc'
            - test/
```

Listing 7: Blueprint Usage

```python
"""Example Blueprint."""
from __future__ import annotations

from typing import cast

from troposphere.awslambda import Code, Function

from runway.cfngin.blueprints.base import Blueprint


class LambdaBlueprint(Blueprint):
    """Example Blueprint."""

    def create_template(self) -> None:
        """Create a template from the blueprint."""
        code = cast(Code, self.context.hook_data["lambda"]["MyFunction"])

        self.template.add_resource(
            Function(
                "MyFunction",
                Code=code,
                Handler="my_function.handler",
                Role="...",
                Runtime="python3.9",
```

(continues on next page)

(continued from previous page)

```
        )
    )
```

awslambda.PythonFunction

Hook Path *runway.cfngin.hooks.awslambda.PythonFunction*

This hook creates deployment packages for Python Lambda Functions, uploads them to S3, and returns data about the deployment package.

The return value can be retrieved using the *hook_data* or by interacting with the *CfnginContext* object passed to the *Blueprint*.

To use this hook to install dependencies, it must be able to find project metadata files. This can include `Pipefile` & `Pipfile.lock` files (pipenv), a `pyproject.toml` & `poetry.lock` files (poetry), or a `requirements.txt` file (pip). The project metadata files can exist either in the source code directory (value of `source_code` arg) or in the same directory as the CFNgin configuration file. If metadata files are not found, dependencies will not be included in the deployment package.

This hook will always use Docker to install/compile dependencies unless explicitly configured not to. It is recommended to always use Docker to ensure a clean and consistent build. It also ensures that binary files built during the install process are compatible with AWS Lambda.

New in version 2.5.0.

Table of Contents

Args

Arguments that can be passed to the hook in the *args* field.

Note: Documentation for each field is automatically generated from class attributes in the source code. When specifying the field, exclude the class name.

PythonHookArgs.**bucket_name:** str

Name of the S3 Bucket where deployment package is/will be stored. The Bucket must be in the same region the Lambda Function is being deployed in.

PythonHookArgs.**cache_dir:** Optional[pathlib.Path]

Explicitly define the directory location. Must be an absolute path or it will be relative to the CFNgin module directory.

If not provided, the cache directory is `.runway/awslambda/pip_cache` within the current working directory.

PythonHookArgs.**docker:** *runway.cfngin.hooks.awslambda.models.args.DockerOptions*

Docker options.

DockerOptions.**disabled:** bool

Explicitly disable the use of docker (default `False`).

If not disabled and Docker is unreachable, the hook will result in an error.

Example

```
args:
  docker:
    disabled: true
```

DockerOptions.**extra_files:** List[str]

List of absolute file paths within the Docker container to copy into the deployment package.

Some Python packages require extra OS libraries (`*.so`) files at runtime. These files need to be included in the deployment package for the Lambda Function to run. List the files here and the hook will handle copying them into the deployment package.

The file name may end in a wildcard (*) to accommodate `.so` files that end in an variable number (see example below).

If the file does not exist, it will result in an error.

Example

```
args:
  docker:
    extra_files:
      - /usr/lib64/mysql/libmysqlclient.so.*
      - /usr/lib64/libxmlsec1-openssl.so
```

DockerOptions.**file:** Optional[pathlib.Path]

Dockerfile to use to build an image for use in this process.

This, `image` , or `runtime` must be provided. If not provided, `image` will be used.

Example

```
args:
  docker:
    file: Dockerfile
```

DockerOptions.**image:** Optional[str]

Docker image to use. If the image does not exist locally, it will be pulled.

This, `file` (takes precedence), or `runtime` must be provided. If only `runtime` is provided, it will be used to determine the appropriate image to use.

Example

```yaml
args:
  docker:
    image: public.ecr.aws/sam/build-python3.9:latest
```

DockerOptions.**name: Optional[str]**

When providing a Dockerfile, this will be the name applied to the resulting image. It is the equivalent to `name` in the `name:tag` syntax of the `docker build [--tag, -t]` command option.

If not provided, a default image name is used.

This field is ignore unless `file` is provided.

Example

```yaml
args:
  docker:
    file: Dockerfile
    name: ${namespace}.runway.awslambda
```

DockerOptions.**pull: bool**

Always download updates to the specified image before use. When building an image, the `FROM` image will be updated during the build process (default `True`).

Example

```yaml
args:
  docker:
    pull: false
```

PythonHookArgs.**extend_gitignore: List[str]**

gitignore rules that should be added to the rules already defined in a `.gitignore` file in the source code directory. This can be used with or without an existing file. Files that match a gitignore rule will not be included in the deployment package.

`.git/` & `.gitignore` will always be added.

Important: This only applies to files in the `source_code` directory.

Example

```yaml
args:
  extend_gitignore:
    - cfngin.yml
    - poetry.lock
    - poetry.toml
    - pyproject.toml
```

PythonHookArgs.**extend_pip_args:** Optional[List[str]]

Additional arguments that should be passed to `pip install`.

Important: When providing this field, be careful not to duplicate any of the arguments passed by this hook (e.g. `--requirements`, `--target`, `--no-input`). Providing duplicate arguments will result in an error.

Example

```yaml
args:
  extend_pip_args:
    - '--proxy'
    - '[user:passwd@]proxy.server:port'
```

PythonHookArgs.**object_prefix:** Optional[str]

Prefix to add to the S3 Object key.

The object will always be prefixed with `awslambda/functions`. If provided, the value will be added to the end of the static prefix (e.g. `awslambda/<functions|layers>/<object_prefix>/<file name>`).

PythonHookArgs.**runtime:** Optional[str]

Runtime of the Lambda Function (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

This, `docker.file`, or `docker.image` must be provided. If `docker.disabled`, this field is required.

When provided, the runtime available on the build system (Docker container or localhost) will be checked against this value. If they do not match, an error will be raised.

If the defined or detected runtime ever changes so that it no longer matches the deployment package in S3, the deployment package in S3 will be deleted and a new one will be built and uploaded.

PythonHookArgs.**slim:** bool

Automatically remove information and caches from dependencies (default `True`). This is done by applying the following gitignore rules to the dependencies:

- `**/*.dist-info*`
- `**/*.py[c|d|i|o]`
- `**/*.so`
- `**/__pycache__*`

PythonHookArgs.**source_code:** pathlib.Path

Path to the Lambda Function source code.

Example

```yaml
args:
  source_code: ./my/package
```

PythonHookArgs.**strip:** bool

Whether or not to strip binary files from the dependencies (default `True`). This only takes effect if `slim: true`.

If false, the gitignore rule `**/*.so` is not used.

PythonHookArgs.**use_cache:** bool

Whether to use a cache directory with pip that will persist builds (default True).

PythonHookArgs.**use_pipenv:** bool

Whether pipenv should be used if determined appropriate.

PythonHookArgs.**use_poetry:** bool

Whether poetry should be used if determined appropriate.

Return Value

class runway.cfngin.hooks.awslambda.models.responses.**AwsLambdaHookDeployResponse**

Data model for AwsLambdaHook deploy response.

When returned by the hook as hook_data, this model is dumped to a standard Dict using the field's aliases as the key in place of the attribute names. This is done so that the key is a direct match to a CloudFormation Property where the value should be used.

bucket_name: str

Name of the S3 Bucket where the deployment package is located. (alias S3Bucket)

code_sha256: str

SHA256 of the deployment package. This can be used by CloudFormation as the value of AWS::Lambda::Version.CodeSha256. (alias CodeSha256)

compatible_architectures: Optional[List[str]]

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html) (alias CompatibleArchitectures)

compatible_runtimes: Optional[List[str]]

A list of compatible function runtimes. Used for filtering with ListLayers and ListLayerVersions. (alias CompatibleRuntimes)

license: Optional[str]

The layer's software license (alias License). Can be any of the following:

- A SPDX license identifier (e.g. MIT).
- The URL of a license hosted on the internet (e.g. https://opensource.org/licenses/MIT).
- The full text of the license.

object_key: str

Key (file path) of the deployment package S3 Object. (alias S3Key)

object_version_id: Optional[str]

The version ID of the deployment package S3 Object. This will only have a value if the S3 Bucket has versioning enabled. (alias S3ObjectVersion)

runtime: str

Runtime of the Lambda Function. (alias Runtime)

Example

Listing 8: Dockerfile

```
FROM public.ecr.aws/sam/build-python3.9:latest

RUN yum install libxml2-devel xmlsec1-devel xmlsec1-openssl-devel libtool-ltdl-devel -y
```

Listing 9: cfngin.yml

```
namespace: ${namespace}
cfngin_bucket: ${cfngin_bucket}
src_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: awslambda.example-function-no-docker
    args:
      bucket_name: ${bucket_name}
      docker:
        disabled: true
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      extend_pip_args:
        - '--proxy'
        - '[user:passwd@]proxy.server:port'
      runtime: python3.9
      slim: false
      source_code: ./src/example-function
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: awslambda.example-function
    args:
      bucket_name: ${bucket_name}
      # docker:  # example of default & inferred values
      #   disabled: false  # default value
      #   image: public.ecr.aws/sam/build-python3.9:latest  # inferred from runtime
      #   pull: true  # default value
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      extend_pip_args:
        - '--proxy'
        - '[user:passwd@]proxy.server:port'
      runtime: python3.9
      source_code: ./src/example-function
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: awslambda.xmlsec
    args:
```

(continues on next page)

(continued from previous page)

```
      bucket_name: ${bucket_name}
      docker:
        extra_files:
          - /usr/lib64/libltdl.so.*
          - /usr/lib64/libxml2.so.*
          - /usr/lib64/libxmlsec1-openssl.so
          - /usr/lib64/libxmlsec1.so.*
          - /usr/lib64/libxslt.so.*
        file: ./Dockerfile
        pull: false
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      source_code: ./src/xmlsec-function
      strip: false

stacks:
  - name: example-stack
    class_path: blueprints.ExampleBlueprint
    parameters:
      XmlCodeSha256: ${awslambda.CodeSha256 awslambda.xmlsec}
      XmlRuntime: ${awslambda.Runtime awslambda.xmlsec}
      XmlS3Bucket: ${awslambda.S3Bucket awslambda.xmlsec}
      XmlS3Key: ${awslambda.S3Key awslambda.xmlsec}
  ...
```

awslambda.PythonLayer

Hook Path *runway.cfngin.hooks.awslambda.PythonLayer*

This hook creates deployment packages for Python Lambda Layers, uploads them to S3, and returns data about the deployment package.

The return value can be retrieved using the *hook_data* or by interacting with the *CfnginContext* object passed to the *Blueprint*.

To use this hook to install dependencies, it must be able to find project metadata files. This can include `Pipefile` & `Pipfile.lock` files (pipenv), a `pyproject.toml` & `poetry.lock` files (poetry), or a `requirements.txt` file (pip). The project metadata files can exist either in the source code directory (value of `source_code` arg) or in the same directory as the CFNgin configuration file. If metadata files are not found, dependencies will not be included in the deployment package.

This hook will always use Docker to install/compile dependencies unless explicitly configured not to. It is recommended to always use Docker to ensure a clean and consistent build. It also ensures that binary files built during the install process are compatible with AWS Lambda.

New in version 2.5.0.

Table of Contents

Args

Arguments that can be passed to the hook in the *args* field.

Documentation for each field is automatically generated from class attributes in the source code. When specifying the field, exclude the class name.

PythonHookArgs.**bucket_name**: str

Name of the S3 Bucket where deployment package is/will be stored. The Bucket must be in the same region the Lambda Function is being deployed in.

PythonHookArgs.**cache_dir**: Optional[pathlib.Path]

Explicitly define the directory location. Must be an absolute path or it will be relative to the CFNgin module directory.

If not provided, the cache directory is `.runway/awslambda/pip_cache` within the current working directory.

PythonHookArgs.**compatible_architectures**: Optional[List[str]]

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html)

Only used by Lambda Layers.

Example

```
args:
  compatible_architectures:
    - x86_64
    - arm64
```

PythonHookArgs.**compatible_runtimes**: Optional[List[str]]

A list of compatible function runtimes. When provided, the `runtime` being used to build the deployment package must be included in the list or an error will be raised.

Only used by Lambda Layers.

Example

```
args:
  compatible_runtimes:
    - python3.9
    - python3.10
```

PythonHookArgs.**docker**: *runway.cfngin.hooks.awslambda.models.args.DockerOptions*

Docker options.

DockerOptions.**disabled:** bool

Explicitly disable the use of docker (default `False`).

If not disabled and Docker is unreachable, the hook will result in an error.

Example

```
args:
  docker:
    disabled: true
```

DockerOptions.**extra_files:** List[str]

List of absolute file paths within the Docker container to copy into the deployment package.

Some Python packages require extra OS libraries (`*.so`) files at runtime. These files need to be included in the deployment package for the Lambda Function to run. List the files here and the hook will handle copying them into the deployment package.

The file name may end in a wildcard (*) to accommodate `.so` files that end in an variable number (see example below).

If the file does not exist, it will result in an error.

Example

```
args:
  docker:
    extra_files:
      - /usr/lib64/mysql/libmysqlclient.so.*
      - /usr/lib64/libxmlsec1-openssl.so
```

DockerOptions.**file:** Optional[pathlib.Path]

Dockerfile to use to build an image for use in this process.

This, `image` , or `runtime` must be provided. If not provided, `image` will be used.

Example

```
args:
  docker:
    file: Dockerfile
```

DockerOptions.**image:** Optional[str]

Docker image to use. If the image does not exist locally, it will be pulled.

This, `file` (takes precedence), or `runtime` must be provided. If only `runtime` is provided, it will be used to determine the appropriate image to use.

Example

```yaml
args:
  docker:
    image: public.ecr.aws/sam/build-python3.9:latest
```

DockerOptions.**name: Optional[str]**

When providing a Dockerfile, this will be the name applied to the resulting image. It is the equivalent to `name` in the `name:tag` syntax of the `docker build [--tag, -t]` command option.

If not provided, a default image name is used.

This field is ignore unless `file` is provided.

Example

```yaml
args:
  docker:
    file: Dockerfile
    name: ${namespace}.runway.awslambda
```

DockerOptions.**pull: bool**

Always download updates to the specified image before use. When building an image, the `FROM` image will be updated during the build process (default `True`).

Example

```yaml
args:
  docker:
    pull: false
```

PythonHookArgs.**extend_gitignore: List[str]**

gitignore rules that should be added to the rules already defined in a `.gitignore` file in the source code directory. This can be used with or without an existing file. Files that match a gitignore rule will not be included in the deployment package.

`.git/` & `.gitignore` will always be added.

Important: This only applies to files in the `source_code` directory.

Example

```yaml
args:
  extend_gitignore:
    - cfngin.yml
    - poetry.lock
    - poetry.toml
    - pyproject.toml
```

PythonHookArgs.**extend_pip_args: Optional[List[str]]**

Additional arguments that should be passed to `pip install`.

Important: When providing this field, be careful not to duplicate any of the arguments passed by this hook (e.g. `--requirements`, `--target`, `--no-input`). Providing duplicate arguments will result in an error.

Example

```yaml
args:
  extend_pip_args:
    - '--proxy'
    - '[user:passwd@]proxy.server:port'
```

PythonHookArgs.**license: Optional[str]**

The layer's software license. Can be any of the following:

- A SPDX license identifier (e.g. `Apache-2.0`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/Apache-2.0`).
- The full text of the license.

Only used by Lambda Layers.

Example

```yaml
args:
  license: Apache-2.0
```

PythonHookArgs.**object_prefix: Optional[str]**

Prefix to add to the S3 Object key.

The object will always be prefixed with `awslambda/functions`. If provided, the value will be added to the end of the static prefix (e.g. `awslambda/<functions|layers>/<object_prefix>/<file name>`).

PythonHookArgs.**runtime: Optional[str]**

Runtime of the Lambda Function (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

This, `docker.file`, or `docker.image` must be provided. If `docker.disabled`, this field is required.

When provided, the runtime available on the build system (Docker container or localhost) will be checked against this value. If they do not match, an error will be raised.

If the defined or detected runtime ever changes so that it no longer matches the deployment package in S3, the deployment package in S3 will be deleted and a new one will be built and uploaded.

PythonHookArgs.**slim: bool**

Automatically remove information and caches from dependencies (default `True`). This is done by applying the following gitignore rules to the dependencies:

- `**/*.dist-info*`
- `**/*.py[c|d|i|o]`
- `**/*.so`

- `**/__pycache__*`

`PythonHookArgs.`**`source_code:`** `pathlib.Path`

Path to the Lambda Function source code.

Example

```
args:
  source_code: ./my/package
```

`PythonHookArgs.`**`strip:`** `bool`

Whether or not to strip binary files from the dependencies (default `True`). This only takes effect if `slim: true`.

If false, the gitignore rule `**/*.so` is not used.

`PythonHookArgs.`**`use_cache:`** `bool`

Whether to use a cache directory with pip that will persist builds (default `True`).

`PythonHookArgs.`**`use_pipenv:`** `bool`

Whether pipenv should be used if determined appropriate.

`PythonHookArgs.`**`use_poetry:`** `bool`

Whether poetry should be used if determined appropriate.

Return Value

class `runway.cfngin.hooks.awslambda.models.responses.`**`AwsLambdaHookDeployResponse`**

Data model for AwsLambdaHook deploy response.

When returned by the hook as `hook_data`, this model is dumped to a standard `Dict` using the field's aliases as the `key` in place of the attribute names. This is done so that the `key` is a direct match to a CloudFormation Property where the value should be used.

`bucket_name:` `str`

Name of the S3 Bucket where the deployment package is located. (alias `S3Bucket`)

`code_sha256:` `str`

SHA256 of the deployment package. This can be used by CloudFormation as the value of `AWS::Lambda::Version.CodeSha256`. (alias `CodeSha256`)

`compatible_architectures:` `Optional[List[str]]`

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html) (alias `CompatibleArchitectures`)

`compatible_runtimes:` `Optional[List[str]]`

A list of compatible function runtimes. Used for filtering with `ListLayers` and `ListLayerVersions`. (alias `CompatibleRuntimes`)

`license:` `Optional[str]`

The layer's software license (alias `License`). Can be any of the following:

- A SPDX license identifier (e.g. `MIT`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/MIT`).
- The full text of the license.

object_key: *str*

Key (file path) of the deployment package S3 Object. (alias S3Key)

object_version_id: *Optional[str]*

The version ID of the deployment package S3 Object. This will only have a value if the S3 Bucket has versioning enabled. (alias S3ObjectVersion)

runtime: *str*

Runtime of the Lambda Function. (alias Runtime)

Example

Listing 10: Dockerfile

```
FROM public.ecr.aws/sam/build-python3.9:latest

RUN yum install libxml2-devel xmlsec1-devel xmlsec1-openssl-devel libtool-ltdl-devel -y
```

Listing 11: cfngin.yml

```
namespace: ${namespace}
cfngin_bucket: ${cfngin_bucket}
src_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: awslambda.example-function-no-docker
    args:
      bucket_name: ${bucket_name}
      compatible_runtimes:
        - python3.9
        - python3.10
      docker:
        disabled: true
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      extend_pip_args:
        - '--proxy'
        - '[user:passwd@]proxy.server:port'
      runtime: python3.9
      slim: false
      source_code: ./src/example-function
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: awslambda.example-function
    args:
      bucket_name: ${bucket_name}
      # docker:  # example of default & inferred values
      #   disabled: false  # default value
      #   image: public.ecr.aws/sam/build-python3.9:latest  # inferred from runtime
```

(continues on next page)

(continued from previous page)

```
      #   pull: true  # default value
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      extend_pip_args:
        - '--proxy'
        - '[user:passwd@]proxy.server:port'
      runtime: python3.9
      source_code: ./src/example-function
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: awslambda.xmlsec
    args:
      bucket_name: ${bucket_name}
      docker:
        extra_files:
          - /usr/lib64/libltdl.so.*
          - /usr/lib64/libxml2.so.*
          - /usr/lib64/libxmlsec1-openssl.so
          - /usr/lib64/libxmlsec1.so.*
          - /usr/lib64/libxslt.so.*
        file: ./Dockerfile
        pull: false
      extend_gitignore:
        - "*.lock"
        - '*.md'
        - '*.toml'
        - tests/
      source_code: ./src/xmlsec-function
      strip: false

stacks:
  - name: example-stack
    class_path: blueprints.ExampleBlueprint
    parameters:
      XmlCompatibleRuntimes: ${awslambda.CompatibleRuntimes awslambda.xmlsec}
      XmlS3Bucket: ${awslambda.S3Bucket awslambda.xmlsec}
      XmlS3Key: ${awslambda.S3Key awslambda.xmlsec}
  ...
```

build_staticsite.build

Hook Path `runway.cfngin.hooks.staticsite.build_staticsite.build`

Build static site. Used by the *Static Site* module type.

Changed in version 2.0.0: Moved from `runway.hooks` to `runway.cfngin.hooks`.

Args

See *Static Site* module documentation for details.

cleanup_s3.purge_bucket

Hook Path `runway.cfngin.hooks.cleanup_s3.purge_bucket`

Delete objects in a Bucket. Primarily used as a *pre_destroy* hook before deleting an S3 bucket.

Changed in version 2.0.0: Moved from `runway.hooks` to `runway.cfngin.hooks`.

Args

bucket_name: *str*
Name of the S3 bucket.

cleanup_ssm.delete_param

Hook Path `runway.cfngin.hooks.cleanup_ssm.delete_param`

Delete SSM parameter. Primarily used when an SSM parameter is created by a hook rather than CloudFormation.

Changed in version 2.0.0: Moved from `runway.hooks` to `runway.cfngin.hooks`.

Args

parameter_name: *str*
Name of an SSM parameter.

command.run_command

Hook Path `runway.cfngin.hooks.command.run_command`

Run a shell custom command as a hook.

Args

command: Union[List[str], str]
Command(s) to run.

capture: Optional[bool] = False
If enabled, capture the command's stdout and stderr, and return them in the hook result.

interactive: Optional[bool] = False
If enabled, allow the command to interact with stdin. Otherwise, stdin will be set to the null device.

ignore_status: Optional[bool] = False
Don't fail the hook if the command returns a non-zero status.

quiet: Optional[bool] = False
Redirect the command's stdout and stderr to the null device, silencing all output. Should not be enabled if `capture` is also enabled.

stdin: Optional[str] = None
String to send to the stdin of the command. Implicitly disables `interactive`.

env: Optional[Dict[str, str]] = None
Dictionary of environment variable overrides for the command context. Will be merged with the current environment.

****kwargs**
Any other arguments will be forwarded to the `subprocess.Popen` function. Interesting ones include: `cwd` and `shell`.

Example

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.command.run_command
    required: true
    enabled: true
    data_key: copy_env
    args:
      command: ['cp', 'environment.template', 'environment']
  - path: runway.cfngin.hooks.command.run_command
    required: true
    enabled: true
    data_key: get_git_commit
    args:
      command: ['git', 'rev-parse', 'HEAD']
      cwd: ./my-git-repo
      capture: true
  - path: runway.cfngin.hooks.command.run_command
    args:
      command: '`cd $PROJECT_DIR/project; npm install`'
      env:
        PROJECT_DIR: ./my-project
        shell: true
```

docker.image.build

Hook Path `runway.cfngin.hooks.docker.image.build`

Docker image build hook.

Replicates the functionality of the `docker image build` CLI command.

New in version 1.18.0.

Args

docker: `Optional[Dict[str, Any]] = {}`

Options for `docker image build`.

buildargs: `Optional[Dict[str, str]] = None`

Dict of build-time variables.

custom_context: `bool = False`

Optional if providing a path to a zip file.

extra_hosts: `Optional[Dict[str, str]] = None`

Extra hosts to add to `/etc/hosts` in the building containers. Defined as a mapping of hostname to IP address.

forcerm: `bool = False`

Always remove intermediate containers, even after unsuccessful builds.

isolation: `Optional[str] = None`

Isolation technology used during build.

network_mode: `Optional[str] = None`

Network mode for the run commands during build.

nocache: `bool = False`

Don't use cache when set to `True`.

platform: `Optional[str] = None`

Set platform if server is multi-platform capable. Uses format `os[/arch[/variant]]`.

pull: `bool = False`

Download any updates to the FROM image in the Dockerfile.

rm: `bool = True`

Remove intermediate containers.

squash: `bool = False`

Squash the resulting image layers into a single layer.

tag: `Optional[str] = None`

Optional name and tag to apply to the base image when it is built.

target: `Optional[str] = None`

Name of the build-stage to build in a multi-stage Dockerfile.

timeout: `Optional[int] = None`

HTTP timeout.

use_config_proxy: `bool = False`

If `True` and if the docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being built.

dockerfile: `Optional[str] = "./Dockerfile"`

Path within the build context to the Dockerfile.

ecr_repo: `Optional[Dict[str, Optional[str]]] = None`

Information describing an ECR repository. This is used to construct the repository URL. If providing a value for this field, do not provide a value for `repo`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

account_id: `Optional[str] = None`

AWS account ID that owns the registry being logged into. If not provided, it will be acquired automatically if needed.

aws_region: `Optional[str] = None`

AWS region where the registry is located. If not provided, it will be acquired automatically if needed.

registry_alias: `Optional[str] = None`

If it is a public repository, provide the alias.

repo_name: `str`

The name of the repository.

path: `Optional[str]`

Path to the directory containing the Dockerfile.

repo: `Optional[str] = None`

URI of a non Docker Hub repository where the image will be stored. If providing one of the other repo values, leave this value empty.

tags: `Optional[List[str]] = ["latest"]`

List of tags to apply to the image.

Returns

type *DockerHookData*

description The value of item `image` in the returned object is set to the *DockerImage* that was just created.

The returned object is accessible with the *hook_data Lookup* under the `data_key` of `docker` (do not specify a `data_key` for the hook, this is handled automatically).

Important: Each execution of this hook overwrites any previous values stored in this attribute. It is advices to consume the resulting image object after it has been built, if it will be consumed by a later hook/stack.

Example

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.docker.login
    args:
      ecr: true
      password: ${ecr login-password}
  - path: runway.cfngin.hooks.docker.image.build
    args:
      ecr_repo:
        repo_name: ${cfn ${namespace}-test-ecr.Repository}
      tags:
        - latest
        - python3.9
  - path: runway.cfngin.hooks.docker.image.push
    args:
      image: ${hook_data docker.image}
```

docker.image.push

Hook Path `runway.cfngin.hooks.docker.image.push`

Docker image push hook.

Replicates the functionality of the `docker image push` CLI command.

New in version 1.18.0.

Args

ecr_repo: Optional[Dict[str, Optional[str]]] = None

Information describing an ECR repository. This is used to construct the repository URL. If providing a value for this field, do not provide a value for `repo` or `image`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

account_id: Optional[str] = None

AWS account ID that owns the registry being logged into. If not provided, it will be acquired automatically if needed.

aws_region: Optional[str] = None

AWS region where the registry is located. If not provided, it will be acquired automatically if needed.

registry_alias: Optional[str] = None

If it is a public repository, provide the alias.

repo_name: str

The name of the repository.

image: Optional[*DockerImage*] = None

A *DockerImage* object. This can be retrieved from `hook_data` for a preceding *docker.image.build* using the *hook_data Lookup*.

If providing a value for this field, do not provide a value for `ecr_repo` or `repo`.

repo: `Optional[str] = None`

URI of a non Docker Hub repository where the image will be stored. If providing one of the other repo values or `image`, leave this value empty.

tags: `Optional[List[str]] = ["latest"]`

List of tags to push.

Example

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.docker.login
    args:
      ecr: true
      password: ${ecr login-password}
  - path: runway.cfngin.hooks.docker.image.build
    args:
      ecr_repo:
        repo_name: ${cfn ${namespace}-test-ecr.Repository}
      tags:
        - latest
        - python3.9
  - path: runway.cfngin.hooks.docker.image.push
    args:
      image: ${hook_data docker.image}

stacks:
  ecr-lambda-function:
    class_path: blueprints.EcrFunction
    variables:
      ImageUri: ${hook_data docker.image.uri.latest}
```

docker.image.remove

Hook Path `runway.cfngin.hooks.docker.image.remove`

Docker image remove hook.

Replicates the functionality of the `docker image remove` CLI command.

New in version 1.18.0.

Args

ecr_repo: `Optional[Dict[str, Optional[str]]] = None`

Information describing an ECR repository. This is used to construct the repository URL. If providing a value for this field, do not provide a value for `repo` or `image`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

account_id: Optional[str] = None

AWS account ID that owns the registry being logged into. If not provided, it will be acquired automatically if needed.

aws_region: Optional[str] = None

AWS region where the registry is located. If not provided, it will be acquired automatically if needed.

registry_alias: Optional[str] = None

If it is a public repository, provide the alias.

repo_name: str

The name of the repository.

force: Optional[bool] = False

Whether to force the removal of the image.

image: Optional[*DockerImage*] = None

A *DockerImage* object. This can be retrieved from `hook_data` for a preceding *docker.image.build* using the *hook_data Lookup*.

If providing a value for this field, do not provide a value for `ecr_repo` or `repo`.

noprune: Optional[bool] = False

Whether to delete untagged parents.

repo: Optional[str] = None

URI of a non Docker Hub repository where the image will be stored. If providing one of the other repo values or `image`, leave this value empty.

tags: Optional[List[str]] = ["latest"]

List of tags to remove.

Example

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.docker.login
    args:
      ecr: true
      password: ${ecr login-password}
  - path: runway.cfngin.hooks.docker.image.build
    args:
      ecr_repo:
        repo_name: ${cfn ${namespace}-test-ecr.Repository}
      tags:
        - latest
        - python3.9
  - path: runway.cfngin.hooks.docker.image.push
    args:
      image: ${hook_data docker.image}
      tags:
        - latest
        - python3.9

stacks:
```

(continues on next page)

(continued from previous page)

```
  ...

post_deploy:
  - path: runway.cfngin.hooks.docker.image.remove
    args:
      image: ${hook_data docker.image}
      tags:
        - latest
        - python3.9
```

docker.login

Hook Path `runway.cfngin.hooks.docker.login`

Docker login hook.

Replicates the functionality of the `docker login` CLI command.

New in version 1.18.0.

Args

dockercfg_path: `Optional[str] = None`

Use a custom path for the Docker config file (`$HOME/.docker/config.json` if present, otherwise `$HOME/.dockercfg`).

ecr: `Optional[Dict[str, Optional[str]]] = None`

Information describing an ECR repository. This is used to construct the repository URL. If providing a value for this field, do not provide a value for `repo` or `image`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

account_id: `Optional[str] = None`

AWS account ID that owns the registry being logged into. If not provided, it will be acquired automatically if needed.

alias: `Optional[str] = None`

If it is a public registry, provide the alias.

aws_region: `Optional[str] = None`

AWS region where the registry is located. If not provided, it will be acquired automatically if needed.

email: `Optional[str] = None`

The email for the registry account.

password: `str`

The plaintext password for the registry account.

registry: `Optional[str] = None`

URL to the registry (e.g. `https://index.docker.io/v1/`).

If providing a value for this field, do not provide a value for `ecr`.

username: `Optional[str] = None`

The registry username. Defaults to `AWS` if supplying `ecr`.

Example

```
pre_deploy:
  - path: runway.cfngin.hooks.docker.login
    args:
      ecr: true
      password: ${ecr login-password}
```

ecr.purge_repository

Hook Path `runway.cfngin.hooks.ecr.purge_repository`

Purge all images from an ECR repository.

New in version 1.18.0.

Args

repository_name: `str`

The name of the ECR repository to purge.

Example

```
pre_destroy:
  - path: runway.cfngin.hooks.ecr.purge_repository
    args:
      repository_name: example-repo
```

ecs.create_clusters

Hook Path `runway.cfngin.hooks.ecs.create_clusters`

Create ECS clusters.

Args

clusters: `List[str]`

Names of clusters to create.

iam.create_ecs_service_role

Hook Path `runway.cfngin.hooks.iam.create_ecs_service_role`

Create ecsServiceRole IAM role.

See also:

AWS Documentation describing the Role

Args

role_name: `Optional[str] = "ecsServiceRole"`
Name of the role to create.

iam.ensure_server_cert_exists

Hook Path `runway.cfngin.hooks.iam.ensure_server_cert_exists`

Ensure server certificate exists.

Args

cert_name: `str`
Name of the certificate that should exist.

prompt: `bool = True`
Whether to prompt to upload a certificate if one does not exist.

keypair.ensure_keypair_exists

Hook Path `runway.cfngin.hooks.keypair.ensure_keypair_exists`

Ensure a specific keypair exists within AWS. If the key doesn't exist, upload it.

Args

keypair: `str`
Name of the key pair to create

public_key_path: `Optional[str] = None`
Path to a public key file to be imported instead of generating a new key. Incompatible with the SSM options, as the private key will not be available for storing.

ssm_key_id: `Optional[str] = None`
ID of a KMS key to encrypt the SSM parameter with. If omitted, the default key will be used.

ssm_parameter_name: `Optional[str] = None`
Path to an SSM store parameter to receive the generated private key, instead of importing it or storing it locally.

route53.create_domain

Hook Path `runway.cfngin.hooks.route53.create_domain`

Create a domain within route53.

Args

domain: *str*

Domain name for the Route 53 hosted zone to be created.

ssm.parameter.SecureString

Hook Path *runway.cfngin.hooks.ssm.parameter.SecureString*

Create, update, and delete a **SecureString** SSM parameter.

A SecureString parameter is any sensitive data that needs to be stored and referenced in a secure manner. If you have data that you don't want users to alter or reference in plaintext, such as passwords or license keys, create those parameters using the SecureString datatype.

When used in the *pre_deploy* or *post_deploy* stage this hook will create or update an SSM parameter.

When used in the *pre_destroy* or *post_destroy* stage this hook will delete an SSM parameter.

New in version 2.2.0.

Args

allowed_pattern: *Optional[str] = None*

A regular expression used to validate the parameter value.

data_type: *Optional[Literal['aws:ec2:image', 'text']] = None*

The data type for a String parameter. Supported data types include plain text and Amazon Machine Image IDs.

description: *Optional[str] = None*

Information about the parameter.

force: *bool = False*

Skip checking the current value of the parameter, just put it. Can be used alongside **overwrite** to always update a parameter.

key_id: *Optional[str] = None*

The KMS Key ID that you want to use to encrypt a parameter. Either the default AWS Key Management Service (AWS KMS) key automatically assigned to your AWS account or a custom key.

name: *str*

The fully qualified name of the parameter that you want to add to the system.

overwrite: *bool = True*

Allow overwriting an existing parameter. If this is set to `False` and the parameter already exists, the parameter will not be updated and a warning will be logged.

policies: *Optional[Union[List[Dict[str, Any]], str]] = None*

One or more policies to apply to a parameter. This field takes a JSON array.

tags: `Optional[Union[Dict[str, str], List[TagTypeDef]]] = None`

Tags to be applied to the parameter.

tier: `Literal['Advanced', 'Intelligent-Tiering', 'Standard'] = "Standard"`

The parameter tier to assign to a parameter.

value: `Optional[str] = None`

The parameter value that you want to add to the system. Standard parameters have a value limit of 4 KB. Advanced parameters have a value limit of 8 KB.

If the value of this field is falsy, the parameter will not be created or updated.

If the value of this field matches what is already in SSM Parameter Store, it will not be updated unless **force** is `True`.

Example

```yaml
pre_deploy: &hooks
  - path: runway.cfngin.hooks.ssm.parameter.SecureString
    args:
      name: /example/foo
      value: bar
  - path: runway.cfngin.hooks.ssm.parameter.SecureString
    args:
      name: /example/parameter1
      description: This is an example.
      tags:
        tag-key: tag-value
      tier: Advanced
      value: ${value_may_be_none}
  - path: runway.cfngin.hooks.ssm.parameter.SecureString
    args:
      name: /example/parameter2
      policies:
        - Type: Expiration
          Version: 1.0
          Attributes:
            Timestamp: 2018-12-02T21:34:33.000Z
      tags:
        - Key: tag-key
          Value: tag-value
      value: ${something_else}

post_destroy: *hooks
```

upload_staticsite.sync

Hook Path `runway.cfngin.hooks.staticsite.upload_staticsite.sync`

Sync static website to S3 bucket. Used by the *Static Site* module type.

Changed in version 2.0.0: Moved from `runway.hooks` to `runway.cfngin.hooks`.

Args

See *Static Site* module documentation for details.

Writing A Custom Hook

A custom hook must be in an executable, importable python package or standalone file. The hook must be importable using your current `sys.path`. This takes into account the *sys_path* defined in the *config* file as well as any `paths` of *package_sources*.

When executed, the hook will have various keyword arguments passed to it. The keyword arguments that will always be passed to the hook are `context` (*CfnginContext*) and `provider` (*Provider*). Anything defined in the *args* field will also be passed to hook as a keyword argument. For this reason, it is recommended to use an unpack operator (`**kwargs`) in addition to the keyword arguments the hook requires to ensure future compatibility and account for misconfigurations.

The hook must return `True` or a truthy object if it was successful. It must return `False` or a falsy object if it failed. This signifies to CFNgin whether or not to halt execution if the hook is *required*. If a `Dict`, *MutableMap*, or `pydantic.BaseModel` is returned, it can be accessed by subsequent hooks, lookups, or Blueprints from the context object. It will be stored as `context.hook_data[data_key]` where *data_key* is the value set in the hook definition. If *data_key* is not provided or the type of the returned data is not a `Dict`, *MutableMap*, or `pydantic.BaseModel`, it will not be added to the context object.

Important: When using a `pydantic.root_validator()` or `pydantic.validator()` `allow_reuse=True` must be passed to the decorator. This is because of how hooks are loaded/re-loaded for each usage. Failure to do so will result in an error if the hook is used more than once.

If using boto3 in a hook, use *context.get_session()* instead of creating a new session to ensure the correct credentials are used.

```
"""context.get_session() example."""
from __future__ import annotations

from typing import TYPE_CHECKING, Any

if TYPE_CHECKING:
    from runway.context import CfnginContext


def do_something(context: CfnginContext, **_kwargs: Any) -> None:
    """Do something."""
    s3_client = context.get_session().client("s3")
```

Example Hook Function

Listing 12: local_path/hooks/my_hook.py

```
"""My hook."""
from typing import Dict, Optional


def do_something(
    *, is_failure: bool = True, name: str = "Kevin", **_kwargs: str
) -> Optional[Dict[str, str]]:
    """Do something."""
    if is_failure:
        return None
    return {"result": f"You are not a failure {name}."}
```

Listing 13: local_path/cfngin.yaml

```
namespace: example
sys_path: ./

pre_deploy:
  - path: hooks.my_hook.do_something
    args:
      is_failure: false
```

Example Hook Class

Hook classes must implement the interface detailed by the *CfnginHookProtocol* `Protocol`. This can be done implicitly or explicitly (by creating a subclass of *CfnginHookProtocol*).

As shown in this example, *HookArgsBaseModel* or it's parent class *BaseModel* can be used to create self validating and sanitizing data models. These can then be used to parse the values provided in the *args* field to ensure they match what is expected.

Listing 14: local_path/hooks/my_hook.py

```
"""My hook."""
import logging
from typing import TYPE_CHECKING, Any, Dict, Optional

from runway.utils import BaseModel
from runway.cfngin.hooks.protocols import CfnginHookProtocol

if TYPE_CHECKING:
    from ...context import CfnginContext

LOGGER = logging.getLogger(__name__)


class MyClassArgs(BaseModel):
    """Arguments for MyClass hook.
```

(continues on next page)

(continued from previous page)

```
    Attributes:
        is_failure: Force the hook to fail if true.
        name: Name used in the response.

    """

    is_failure: bool = False
    name: str


class MyClass(CfnginHookProtocol):
    """My class does a thing.

    Keyword Args:
        is_failure (bool): Force the hook to fail if true.
        name (str): Name used in the response.

    Returns:
        Dict[str, str]: Response message is stored in ``result``.

    Example:
    .. code-block:: yaml

        pre_deploy:
          - path: hooks.my_hook.MyClass
            args:
            is_failure: False
            name: Karen

    """

    args: MyClassArgs

    def __init__(self, context: CfnginContext, **kwargs: Any) -> None:
        """Instantiate class.

        Args:
            context: Context instance. (passed in by CFNgin)
            provider: Provider instance. (passed in by CFNgin)

        """
        kwargs.setdefault("tags", {})

        self.args = self.ARGS_PARSER.parse_obj(kwargs)
        self.args.tags.update(context.tags)
        self.context = context

    def post_deploy(self) -> Optional[Dict[str, str]]:
        """Run during the **post_deploy** stage."""
        if self.args["is_failure"]:
            return None
```

(continues on next page)

(continued from previous page)

```
        return {"result": f"You are not a failure {self.args['name']}."}

    def post_destroy(self) -> None:
        """Run during the **post_destroy** stage."""
        LOGGER.error("post_destroy is not supported by this hook")

    def pre_deploy(self) -> None:
        """Run during the **pre_deploy** stage."""
        LOGGER.error("pre_deploy is not supported by this hook")

    def pre_destroy(self) -> None:
        """Run during the **pre_destroy** stage."""
        LOGGER.error("pre_destroy is not supported by this hook")
```

Listing 15: local_path/cfngin.yaml

```
namespace: example
sys_path: ./

pre_deploy:
  - path: hooks.my_hook.MyClass
    args:
      is_failure: False
      name: Karen
```

Lookups

Important: Runway lookups and CFNgin lookups are not interchangeable. While they do share a similar base class and syntax, they exist in two different registries. Runway config files can't use CFNgin lookups just as the CFNgin config cannot use Runway lookups.

Runway's CFNgin provides the ability to dynamically replace values in the config via a concept called lookups. A lookup is meant to take a value and convert it by calling out to another service or system.

A lookup is denoted in the config with the `${<lookup type> <lookup input>}` syntax.

Lookups are only resolved within *Variables*. They can be nested in any part of a YAML data structure and within another lookup itself.

Note: If a lookup has a non-string return value, it can be the only lookup within a field.

e.g. if `custom` returns a list, this would raise an exception:

```
Variable: ${custom something}, ${output otherStack.Output}
```

This is valid:

```
Variable: ${custom something}
```

For example, given the following:

```yaml
stacks:
  - name: sg
    class_path: some.stack.blueprint.Blueprint
    variables:
      Roles:
        - ${output otherStack.IAMRole}
      Values:
        Env:
          Custom: ${custom ${output otherStack.Output}}
          DBUrl: postgres://${output dbStack.User}@${output dbStack.HostName}
```

The *Blueprint* would have access to the following resolved variables dictionary:

```
{
    "Roles": ["other-stack-iam-role"],
    "Values": {
        "Env": {
            "Custom": "custom-output",
            "DBUrl": "postgres://user@hostname",
        },
    },
}
```

Built-in Lookups

ami

Query Syntax `[<region>@]<argument>:<argument-value> <argument>:<argument-value>, <argument-value>`

The *ami* lookup is meant to search for the most recent AMI created that matches the given filters.

Arguments

Any other arguments specified but not listed below are sent as filters to the AWS API. For example, `architecture:x86_64` would add a filter.

region: Optional[str]

AWS region to search (e.g. `us-east-1`). Defaults to the current region.

owners: Union[List[str], str]

At least one owner must be specified in the query (e.g. `amazon`, `self`, or an AWS account ID). Multiple owners can be provided by using a comma to delimitate the list.

name_regex: str

Regex pattern for the name of the AMI (e.g. `my-ubuntu-server-[0-9]+`).

executable_users: Optional[str]

`amazon`, `self`, or an AWS account ID.

Example

```
# Grabs the most recently created AMI that is owned by either this account,
# amazon, or the account id 888888888888 that has a name that matches
# the regex "server[0-9]+" and has "i386" as its architecture.

# Note: The region is optional, and defaults to the current CFNgin region
ImageId: ${ami [<region>@]owners:self,888888888888,amazon name_regex:server[0-9]+␣
↪architecture:i386}
```

awslambda

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

An *AwsLambdaHookDeployResponse* object is returned by this lookup. It is recommended to only use this lookup when passing the value into a *Blueprint* or hook as further processing will be required.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      code_metadata: ${awslambda example-function-01}
      ...
  - name: example-stack-02
    class_path: blueprints.BarStack
    variables:
```

(continues on next page)

(continued from previous page)

```
    code_metadata: ${awslambda example-function-02}
    ...
```

awslambda.Code

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A `troposphere.awslambda.Code` object is returned by this lookup. It is recommended to only use this lookup when passing the value into a *Blueprint* or hook as further processing will be required. However, it can be passed directly in the `Code` keyword argument of `troposphere.awslambda.Function`.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      code_object: ${awslambda.Code example-function-01}
      ...
  - name: example-stack-02
    class_path: blueprints.BarStack
    variables:
      code_object: ${awslambda.Code example-function-02}
      ...
```

awslambda.CodeSha256

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string is returned by this lookup. The returned value can be passed directly to AWS::Lambda::Version.CodeSha256.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      CodeSha256: ${awslambda.CodeSha256 example-function-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      CodeSha256: ${awslambda.CodeSha256 example-function-02}
      ...
```

awslambda.CompatibleArchitectures

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A list of strings or `None` is returned by this lookup. The returned value can be passed directly to AWS::Lambda::LayerVersion.CompatibleArchitectures.

New in version 2.5.0.

Arguments

This lookup only supports the `transform` argument which can be used to turn the list of strings into a comma delimited list.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      CompatibleArchitectures: ${awslambda.CompatibleArchitectures example-layer-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      CompatibleArchitectures: ${awslambda.CompatibleArchitectures example-layer-
↪02::transform=str}
      ...
```

awslambda.CompatibleRuntimes

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A list of strings or `None` is returned by this lookup. The returned value can be passed directly to AWS::Lambda::LayerVersion.CompatibleRuntimes.

New in version 2.5.0.

Arguments

This lookup only supports the `transform` argument which can be used to turn the list of strings into a comma delimited list.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      CompatibleRuntimes: ${awslambda.CompatibleRuntimes example-layer-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      CompatibleRuntimes: ${awslambda.CompatibleRuntimes example-layer-02::transform=str}
      ...
```

awslambda.Content

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A `troposphere.awslambda.Content` object is returned by this lookup. It is recommended to only use this lookup when passing the value into a *Blueprint* or hook as further processing will be required. However, it can be passed directly in the `Content` keyword argument of `troposphere.awslambda.LayerVersion`.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      layer_content: ${awslambda.Content example-layer-01}
      ...
  - name: example-stack-02
    class_path: blueprints.BarStack
    variables:
      layer_content: ${awslambda.Content example-layer-02}
      ...
```

awslambda.License

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string or `None` is returned by this lookup. The returned value can be passed directly to AWS::Lambda::LayerVersion.License.

New in version 2.5.0.

Example

```yaml
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonLayer
    data_key: example-layer-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      License: ${awslambda.License example-layer-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      License: ${awslambda.License example-layer-02}
      ...
```

awslambda.Runtime

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string is returned by this lookup. The returned value can be passed directly to AWS::Lambda::Function.Runtime. While not necessary, using this hook to fill in `Runtime` ensures compatibility with the deployment package and removes the need for duplicating configuration values.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      Runtime: ${awslambda.Runtime example-function-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      Runtime: ${awslambda.Runtime example-function-02}
      ...
```

awslambda.S3Bucket

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string is returned by this lookup. The returned value can be passed directly to AWS::Lambda::Function.Code.S3Bucket or AWS::Lambda::LayerVersion.Content.S3Bucket.

New in version 2.5.0.

Example

```yaml
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      S3Bucket: ${awslambda.S3Bucket example-function-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      S3Bucket: ${awslambda.S3Bucket example-function-02}
      ...
```

awslambda.S3Key

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string is returned by this lookup. The returned value can be passed directly to AWS::Lambda::Function.Code.S3Key or AWS::Lambda::LayerVersion.Content.S3Key.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      S3Key: ${awslambda.S3Key example-function-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      S3Key: ${awslambda.S3Key example-function-02}
      ...
```

awslambda.S3ObjectVersion

Query Syntax <hook.data_key>

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

A string is returned by this lookup. The returned value can be passed directly to AWS::Lambda::Function.Code.S3ObjectVersion or AWS::Lambda::LayerVersion.Content.S3ObjectVersion.

New in version 2.5.0.

Example

```
namespace: example
cfngin_bucket: ''
sys_path: ./

pre_deploy:
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-01
    args:
      ...
  - path: runway.cfngin.hooks.awslambda.PythonFunction
    data_key: example-function-02
    args:
      ...

stacks:
  - name: example-stack-01
    class_path: blueprints.FooStack
    variables:
      S3ObjectVersion: ${awslambda.S3ObjectVersion example-function-01}
      ...
  - name: example-stack-02
    template_path: ./templates/bar-stack.yml
    variables:
      S3ObjectVersion: ${awslambda.S3ObjectVersion example-function-02}
      ...
```

cfn

Important: The Stack must exist in CloudFormation before the config using this Lookup begins processing to successfully get a value. This means that it must have been deployed using another Runway module, deployed from a config that is run before the one using it, deployed manually, or deployed in the same config using `required`/*`required_by`* to specify a dependency between the Stacks.

Query Syntax `<stack-name>.<output-name>[::<arg>=<arg-val>, ...]`

Retrieve a value from CloudFormation Stack Outputs.

When specifying the output name, be sure to use the *Logical ID* of the output; not the *Export.Name*.

If the Lookup is unable to find a CloudFormation Stack Output matching the provided query, the default value is returned or an exception is raised to show why the value could be be resolved (e.g. Stack does not exist or output does not exist on the Stack).

New in version 1.11.0.

See also:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/outputs-section-structure.html

Arguments

This Lookup supports all *Common Lookup Arguments*.

Example

```
namespace: example

stacks:
  - ...
    variables:
      VpcId: ${cfn ${namespace}-vpc.Id}
```

Given the above config file, the lookup will get the value of the Output named **Id** from Stack **example-vpc**.

default

Query Syntax `<env_var>::<default value>`

The *default* lookup type will check if a value exists for the variable in the environment file, then fall back to a default defined in the CFNgin config if the environment file doesn't contain the variable. This allows defaults to be set at the config file level, while granting the user the ability to override that value per environment.

Note: The *default* lookup only supports checking if a variable is defined in an environment file. It does not support other embedded lookups to see if they exist. Only checking variables in the environment file are supported. If you attempt to have the default lookup perform any other lookup that fails, CFNgin will throw an exception for that lookup and will exit before it gets a chance to fall back to the default in your config.

Example

```
Groups: ${default app_security_groups::sg-12345,sg-67890}
```

If `app_security_groups` is defined in the environment file, its defined value will be returned. Otherwise, `sg-12345,sg-67890` will be the returned value.

dynamodb

Query Syntax `<region>:<table-name>@<partition-key>:<value>.<attribute>`

The *dynamodb* lookup type retrieves a value from a DynamoDb table.

As an example, if you have a Dynamo Table named `TestTable` and it has an Item with a Primary Partition key called `TestKey` and a value named `BucketName`, you can look it up by using CFNgin. The lookup key in this case is TestVal

Example

```
# We can reference that dynamo value
BucketName: ${dynamodb us-east-1:TestTable@TestKey:TestVal.BucketName}

# Which would resolve to:
BucketName: test-bucket
```

You can lookup other data types by putting the data type in the lookup. Valid values are `S` (String), `N` (Number), `M` (Map), `L` (List).

```
ServerCount: ${dynamodb us-east-1:TestTable@TestKey:TestVal.ServerCount[N]}
```

This would return an int value, rather than a string

You can lookup values inside of a map.

```
ServerCount: ${dynamodb us-east-1:TestTable@TestKey:TestVal.ServerInfo[M].ServerCount[N]}
```

ecr

Retrieve a value from AWS Elastic Container Registry (ECR).

This Lookup only supports very specific queries.

New in version 1.18.0.

Supported Queries

login-password

Get a password to login to ECR registry.

The returned value can be passed to the login command of the container client of your preference, such as the CFNgin *docker.login*. After you have authenticated to an Amazon ECR registry with this Lookup, you can use the client to push and pull images from that registry as long as your IAM principal has access to do so until the token expires. The authorization token is valid for **12 hours**.

Arguments

This Lookup does not support any arguments.

Example

```
pre_deploy:
  - path: runway.cfngin.hooks.docker.login
    args:
      password: ${ecr login-password}
      ...
```

env

Query Syntax `<variable-name>[::<arg>=<arg-val>, ...]`

Retrieve a value from an environment variable.

The value is retrieved from a copy of the current environment variables that is saved to the context object. These environment variables are manipulated at runtime by Runway to fill in additional values such as `DEPLOY_ENVIRONMENT` and `AWS_REGION` to match the current execution.

If the Lookup is unable to find an environment variable matching the provided query, the default value is returned or a `ValueError` is raised if a default value was not provided.

New in version 2.7.0.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- region

Example

```
stacks:
  - ...
    variables:
      Foo: ${env bar::default=foobar}
```

envvar

Deprecated since version 2.7.0: Replaced by *env*

Query Syntax `<variable-name>`

The *envvar* lookup type retrieves a value from a variable in the shell's environment.

Example

```
# Set an environment variable in the current shell.
$ export DATABASE_USER=root

# In the CFNgin config we could reference the value:
DBUser: ${envvar DATABASE_USER}

# Which would resolve to:
DBUser: root
```

You can also get the variable name from a file, by using the `file://` prefix in the lookup, like so:

```
DBUser: ${envvar file://dbuser_file.txt}
```

file

Query Syntax `<supported-codec>:<data>`

The *file* lookup type allows the loading of arbitrary data from files. The lookup additionally supports using a **codec** to parse and/or manipulate the file contents prior to returning it. The `parameterized-b64` codec is particularly useful to allow the interpolation of CloudFormation parameters in a UserData attribute of an instance or launch configuration.

If the file can be read locally, the `<data>` portion of the query should look something like `file://./path/to/file`. The `file://` prefix tells CFNgin that it needs to open the file located at the provide path and read it's contents before continuing. For a relative path, it will be relative to the current working directory (usually the root of the CFNgin module being processed).

If `<data>` is not prefixed with `file://`, it will be treated as the contents of the file. This enables lookups to be chained together to retrieve data and still take advantage of a codec to further parse and/or manipulate it as needed. For example, the value of an SSM Parameter can be parsed as `json-parameterized` before it is returned by the lookup with the following `${file json-parameterized:${ssm /parameter/name}}`.

Supported Codecs

- **plain** - Load the contents of the file untouched. This is the only codec that should be used with raw Cloudformation templates (the other codecs are intended for blueprints).
- **base64** - Encode the plain text file at the given path with base64 prior to returning it.
- **parameterized** - The same as plain, but additionally supports referencing CloudFormation parameters to create userdata that's supplemented with information from the template, as is commonly needed in EC2 UserData. For example, given a template parameter of BucketName, the file could contain the following text:

  ```
  #!/bin/sh
  aws s3 sync s3://{{BucketName}}/somepath /somepath
  ```

  and then you could use something like this in the YAML config file:

  ```
  UserData: ${file parameterized:file://path/to/file}
  ```

  resulting in the UserData parameter being defined as:

```
{
    "Fn::Join" : [
        "",
        [
            "#!/bin/sh\naws s3 sync s3://",
            {
                "Ref" : "BucketName"
            },
            "/somepath /somepath"
        ]
    ]
}
```

- **parameterized-b64** - The same as parameterized, with the results additionally wrapped in `{ "Fn::Base64": ... }`, which can be used as EC2 UserData.

  When using parameterized-b64 for UserData, you should use a parameter defined as such.

```
from troposphere import AWSHelperFn

"UserData": {
    "type": AWSHelperFn,
    "description": "Instance user data",
    "default": Ref("AWS::NoValue")
}
```

  Then set UserData in a LaunchConfiguration or Instance to `self.variables["UserData"]`. Note that we use AWSHelperFn as the type because the parameterized-b64 codec returns either a Base64 or a GenericHelperFn troposphere object.

- **json** - Decode the file as JSON and return the resulting object.
- **json-parameterized** - Same as `json`, but applying templating rules from `parameterized` to every object *value*. Note that object *keys* are not modified.

  Example (an external PolicyDocument):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "some:Action"
            ],
            "Resource": "{{MyResource}}"
        }
    ]
}
```

- **yaml** - Decode the file as YAML and return the resulting object.
- **yaml-parameterized** - Same as `json-parameterized`, but using YAML.

```
Version: 2012-10-17
Statement:
```

(continues on next page)

(continued from previous page)

```
  - Effect: Allow
    Action:
      - "some:Action"
    Resource: "{{MyResource}}"
```

Example

```
# We've written a file to /some/path:
$ echo "hello there" > /some/path

# In CFNgin we would reference the contents of this file with the following
conf_key: ${file plain:file://some/path}

# The above would resolve to
conf_key: hello there

# Or, if we used wanted a base64 encoded copy of the file data
conf_key: ${file base64:file://some/path}

# The above would resolve to
conf_key: aGVsbG8gdGhlcmUK
```

hook_data

Query Syntax `<hook.data_key>[::<arg>=<arg-val>, ...]`

When using hooks, you can have the hook store results in the *CfnginContext.hook_data* dictionary on the context by setting *data_key* in the *hook* config.

This lookup lets you look up values in that dictionary. A good example of this is when you use the *aws_lambda.upload_lambda_functions* to upload AWS Lambda code, then need to pass that code object as the **Code** variable in a Blueprint.

Changed in version 2.0.0: Support for the syntax deprecated in *1.5.0* has been removed.

Changed in version 1.5.0: The `<hook_name>::<key>` syntax was deprecated with support being added for the `key.nested_key` syntax for accessing data within a dictionary.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- region

Example

```
# If you set the ``data_key`` config on the aws_lambda hook to be "myfunction"
# and you name the function package "TheCode" you can get the troposphere
# awslambda.Code object with:

Code: ${hook_data myfunction.TheCode}

# If you need to pass the code location as individual strings for use in a
# CloudFormation template instead of a Blueprint, you can do so like this:

Bucket: ${hook_data myfunction.TheCode::load=troposphere, get=S3Bucket}
Key: ${hook_data myfunction.TheCode::load=troposphere, get=S3Key}
```

kms

Query Syntax `<encrypted-blob>[::region=<region>, ...]`

The *kms* lookup type decrypts its input value.

As an example, if you have a database and it has a parameter called `DBPassword` that you don't want to store in plain text in your config (maybe because you want to check it into your version control system to share with the team), you could instead en crypt the value using `kms`.

Changed in version 2.7.0: The `[<region>@]<encrypted-blob>` syntax is deprecated to comply with Runway's lookup syntax.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- default

Example

We use can use the aws cli to get the encrypted value for the string "PASSWORD" using the master key called 'myKey' in us-east-1.

```
$ aws --region us-east-1 kms encrypt --key-id alias/myKey \
    --plaintext "PASSWORD" --output text --query CiphertextBlob

CiD6bC8t2Y<...encrypted blob...>
```

```
namespace: example

stacks:
  - ...
    variables:
      # With CFNgin we would reference the encrypted value like:
      DBPassword: ${kms CiD6bC8t2Y<...encrypted blob...>::region=us-east-1}
```

(continues on next page)

(continued from previous page)

```
    # The above would resolve to:
    DBPassword: PASSWORD
```

This requires that the credentials used by CFNgin have access to the master key used to encrypt the value.

It is also possible to store the encrypted blob in a file (useful if the value is large) using the `file://` prefix, ie:

```
namespace: example

stacks:
  - ...
    variables:
      DockerConfig: ${kms file://dockercfg}
```

Note: Lookups resolve the path specified with `file://` relative to the location of the config file, not the current working directory.

output

Query Syntax `<relative-stack-name>.<output-name>[::<arg>=<arg-val>, ...]`

The *output* lookup retrieves an Output from the given Stack name within the current *namespace*.

CFNgin treats output lookups differently than other lookups by auto adding the referenced stack in the lookup as a requirement to the stack whose variable the output value is being passed to.

Changed in version 2.7.0: The `<relative-stack-name>::<output-name>` syntax is deprecated to comply with Runway's lookup syntax.

Arguments

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- region

Example

You can specify an output lookup with the following syntax:

```
namespace: example

stacks:
  - ...
    variables:
      ConfVariable: ${output stack-name.OutputName}
```

random.string

Query Syntax `<desired-length>`

Generate a random string of the given length.

New in version 2.2.0.

Arguments

digits: `bool = True`

When generating the random string, the string may contain digits (`[0-9]`). If the string can contain digits, it will always contain at least one.

lowercase: `bool = True`

When generating the random string, the string may contain lowercase letters (`[a-z]`). If the string can contain lowercase letters, it will always contain at least one.

punctuation: `bool = False`

When generating the random string, the string may contain ASCII punctuation (`[!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]`). If the string can contain ASCII punctuation, it will always contain at least one.

uppercase: `bool = True`

When generating the random string, the string may contain uppercase letters (`[A-Z]`). If the string can contain uppercase letters, it will always contain at least one.

This Lookup supports all *Common Lookup Arguments* but, the following have limited or no effect:

- default
- get
- indent
- load
- region

Example

This example shows the use of this lookup to create an SSM parameter that will retain value generated during the first deployment. Even through subsequent deployments generate a new value that is passed to the hook, the hook does not overwrite the value of an existing parameter.

```
pre_deploy: &hooks
  - path: runway.cfngin.hooks.ssm.parameter.SecureString
    args:
      name: /${namespace}/password
      overwrite: false
      value: ${random.string 12::punctuation=true}

post_destroy: *hooks
```

rxref

Query Syntax `<relative-stack-name>.<output-name>[::<arg>=<arg-val>, ...]`

The *rxref* lookup type is very similar to the *cfn* lookup type. Where the *cfn* type assumes you provided a fully qualified stack name, *rxref*, like the *output* expands and retrieves the output from the given Stack name within the current *namespace*, even if not defined in the CFNgin config you provided it.

Because there is no requirement to keep all stacks defined within the same CFNgin YAML config, you might need the ability to read outputs from other Stacks deployed by CFNgin into your same account under the same *namespace*. *rxref* gives you that ability. This is useful if you want to break up very large configs into smaller groupings.

Also, unlike the *output* type, *rxref* doesn't impact Stack requirements.

Changed in version 2.7.0: The `<relative-stack-name>::<output-name>` syntax is deprecated to comply with Runway's lookup syntax.

Arguments

This Lookup supports all *Common Lookup Arguments*.

Example

```yaml
namespace: namespace

stacks:
  - ...
    variables:
      ConfVariable0: ${rxref my-stack.SomeOutput}
      # both of these lookups are functionally equivalent
      ConfVariable1: ${cfn namespace-my-stack.SomeOutput}
```

Although possible, it is not recommended to use `rxref` for stacks defined within the same CFNgin YAML config. Doing so would require the use of *required_by* or *requires*.

ssm

Query Syntax `<parameter>[::<arg>=<arg-val>, ...]`

Retrieve a value from SSM Parameter Store.

If the Lookup is unable to find an SSM Parameter matching the provided query, the default value is returned or `ParameterNotFound` is raised if a default value is not provided.

Parameters of type `SecureString` are automatically decrypted.

Parameters of type `StringList` are returned as a list.

New in version 1.5.0.

Arguments

This Lookup supports all *Common Lookup Arguments*.

Example

```
stacks:
  - ...
    variables:
      Example: ${ssm /example/secret}
```

xref

Deprecated since version 1.11.0: Replaced by *cfn*

Query Syntax `<fully-qualified-stack-name>::<output-name>`

The *xref* lookup type is very similar to the *output* type, the difference being that *xref* resolves output values from stacks that aren't contained within the current CFNgin *namespace*, but are existing Stacks containing outputs within the same region on the AWS account you are deploying into. *xref* allows you to lookup these outputs from the Stacks already in your account by specifying the stacks fully qualified name in the CloudFormation console.

Where the *output* type will take a Stack name and use the current context to expand the fully qualified stack name based on the *namespace*, *xref* skips this expansion because it assumes you've provided it with the fully qualified stack name already. This allows you to reference output values from any CloudFormation Stack in the same region.

Also, unlike the *output* type, *xref* doesn't impact stack requirements.

Example

```
ConfVariable: ${xref fully-qualified-stack::SomeOutput}
```

Writing A Custom Lookup

A custom lookup may be registered within the config. It custom lookup must be in an executable, importable python package or standalone file. The lookup must be importable using your current `sys.path`. This takes into account the *sys_path* defined in the config file as well as any `paths` of *package_sources*.

The lookup must be a subclass of *LookupHandler* with a `@classmethod` of `handle` with a similar signature to what is provided in the example below. The subclass must override the *TYPE_NAME* class variable with a name that will be used to register the lookup. There must be only one lookup per file.

The lookup must return a string if being used for a CloudFormation parameter.

If using boto3 in a lookup, use *context.get_session()* instead of creating a new session to ensure the correct credentials are used.

Important: When using a `pydantic.root_validator()` or `pydantic.validator()` in a lookup `allow_reuse=True` must be passed to the decorator. This is because of how lookups are loaded/re-loaded when they are registered. Failure to do so will result in an error if the lookup is registered more than once.

Example

```python
"""Example lookup."""
from __future__ import annotations

from typing import TYPE_CHECKING, Any, Final, Literal, Optional, Union

from runway.cfngin.utils import read_value_from_path
from runway.lookups.handlers.base import LookupHandler

if TYPE_CHECKING:
    from runway.cfngin.providers.aws.default import Provider
    from runway.context import CfnginContext, RunwayContext


class MylookupLookup(LookupHandler):
    """My lookup."""

    TYPE_NAME: Final[Literal["mylookup"]] = "mylookup"
    """Name that the Lookup is registered as."""

    @classmethod
    def handle(
        cls,
        value: str,
        context: Union[CfnginContext, RunwayContext],
        *_args: Any,
        provider: Optional[Provider] = None,
        **_kwargs: Any
    ) -> str:
        """Do something.

        Args:
            value: Value to resolve.
            context: The current context object.
            provider: CFNgin AWS provider.

        """
        query, args = cls.parse(read_value_from_path(value))

        # example of using get_session for a boto3 session
        s3_client = context.get_session().client("s3")

        return "something"
```

Persistent Graph

Each time Runway's CFNgin is run, it creates a dependency *graph* of `stacks`. This is used to determine the order in which to execute them. This *graph* can be persisted between runs to track the removal of `stacks` from the config file.

When a `stack` is present in the persistent graph but not in the *graph* constructed from the config file, CFNgin will delete the Stack from CloudFormation. This takes effect when running either the *deploy command* or *destroy command*.

To enable persistent graph, define the `persistent_graph_key` field as a unique value that will be used to construct the path to the persistent graph object in S3. This object is stored in the `cfngin_bucket` which is also used for CloudFormation templates. The fully qualified path to the object will look like the below.

```
s3://${cfngin_bucket}/${namespace}/persistent_graphs/${namespace}/${persistent_graph_key}
↪.json
```

Note: It is recommended to enable versioning on the `cfngin_bucket` when using persistent graph to have a backup version in the event something unintended happens. A warning will be logged if this is not enabled.

If CFNgin creates a `cfngin_bucket` for you when persistent graph is enabled, it will be created with versioning enabled.

Important: When choosing a value for `persistent_graph_key`, it is vital to ensure the value is unique for the `namespace` being used. If the key is a duplicate, Stacks that are not intended to be destroyed will be destroyed.

When executing an action that will be modifying the persistent graph (deploy or destroy), the S3 object is *"locked"*. The lock is a tag applied to the object at the start of one of these actions. The tag-key is **cfngin_lock_code** and the tag-value is UUID generated each time a config is processed.

To lock a persistent graph object, the tag must not be present on the object. For CFNgin to act on the *graph* (modify or unlock) the value of the tag must match the UUID of the current CFNgin session. If the object is locked or the code does not match, an error will be raised and no action will be taken. This prevents two parties from acting on the same persistent graph object concurrently which would create a race condition.

Note: A persistent graph object can be unlocked manually by removing the **cfngin_lock_code** tag from it. This should be done with caution as it will cause any active sessions to raise an error.

Example

Listing 16: configuration file

```yaml
namespace: example
cfngin_bucket: cfngin-bucket
persistent_graph_key: my_graph  # .json - will be appended if not provided
stacks:
  - name: first_stack:
    ...
  - name: new_stack:
    ...
```

Listing 17: s3://cfngin-bucket/persistent_graphs/example/my_graph.json

```
{
  "first_stack": [],
  "removed_stack": [
    "first_stack"
  ]
}
```

Given the above config file and persistent graph, when running `runway deploy`, the following will occur.

1. The `{"Key": "cfngin_lock_code", "Value": "123456"}` tag is applied to **s3://cfngin-bucket/persistent_graphs/example/my_graph.json** to lock it to the current session.
2. **removed_stack** is deleted from CloudFormation and deleted from the persistent graph object in S3.
3. **first_stack** is updated in CloudFormation and updated in the persistent graph object in S3 (incase dependencies change).
4. **new_stack** is created in CloudFormation and added to the persistent graph object in S3.
5. The `{"Key": "cfngin_lock_code", "Value": "123456"}` tag is removed from **s3://cfngin-bucket/persistent_graphs/example/my_graph.json** to unlock it for use in other sessions.

Remote Sources

The *package_sources* field can be used to define additional locations to include when processing a configuration file. The locations can either be a local file path or a network accessible location.

By defining these additional sources you are able to extend your *$PATH* to make more resources accessible or even merge multiple configuration files into the current configuration file.

class cfngin.**package_sources**

There are three different types of package sources - git repository, local, and AWS S3.

git: Optional[List[*cfngin.package_source.git*]] = []

A list of git repositories to include when processing the configuration file.

See *Git Repository* for detailed information.

```
package_sources:
  git:
    ...
```

local: Optional[List[*cfngin.package_source.local*]] = []

A list of additional local directories to include when processing the configuration file.

See *Local* for detailed information.

```
package_sources:
  local:
    ...
```

s3: Optional[List[*cfngin.package_source.s3*]] = []

A list of AWS S3 objects to include when processing the configuration file.

See *AWS S3* for detailed information.

```
package_sources:
  s3:
    ...
```

Contents

Git Repository

class cfngin.package_source.**git**

Package source located in a git repository.

Cloned repositories are cached locally between runs. The cache location is defined by *cfngin.config.cfngin_cache_dir*.

branch: Optional[str] = None

Name of a branch to checkout after cloning the git repository.

Only one of *branch*, *commit*, or *tag* can be defined.

Example

```
package_sources:
  git:
    - branch: master
```

commit: Optional[str] = None

After cloning the git repository, reset *HEAD* to the given commit hash.

Only one of *branch*, *commit*, or *tag* can be defined.

Example

```
package_sources:
  git:
    - commit: 5d83f7ff1ad6527233be2c27e9f68816599b6c57
```

configs: Optional[List[str]] = []

Configuration files from this source location can also be used by specifying a list of file paths.

These configuration files are merged into the current configuration file with the current file taking precedence. When using this usage pattern, it is advised to use dictionary definitions for everything that supports it to allow for granular overriding.

Example

```yaml
package_sources:
  git:
    - configs:
      - example-01.yml
      - example-02.yml
```

paths: Optional[List[str]] = []

A list of subdirectories within the source location that should be added to *$PATH*.

Example

```yaml
package_sources:
  git:
    - paths:
      - some/directory/
      - another/
```

tag: Optional[str] = None

After cloning the git repository, reset *HEAD* to the given tag.

Only one of *branch*, *commit*, or *tag* can be defined.

Example

```yaml
package_sources:
  git:
    - tag: v1.0.0
```

uri: str

The protocol and URI address of the git repository.

Example

```yaml
package_sources:
  git:
    - uri: git@github.com:onicagroup/runway.git  # ssh
    - uri: https://github.com/onicagroup/runway.git  # https
```

Local

class cfngin.package_source.**local**

Package source located on a local disk.

configs: Optional[List[str]] = []

Configuration files from this source location can also be used by specifying a list of file paths.

These configuration files are merged into the current configuration file with the current file taking precedence. When using this usage pattern, it is advised to use dictionary definitions for everything that supports it to allow for granular overriding.

Example

```yaml
package_sources:
  local:
    - configs:
      - example-01.yml
      - example-02.yml
```

paths: `Optional[List[str]] = []`

A list of subdirectories within the source location that should be added to *$PATH*.

Example

```yaml
package_sources:
  local:
    - paths:
      - some/directory/
      - another/
```

source: `str`

Path relative to the current configuration file that is the root of the local package source. Can also be provided as an absolute path but this is not recommended as it will be bound to your system.

Example

```yaml
package_sources:
  local:
    - source: ./../example_code
```

AWS S3

class cfngin.package_source.**s3**

Package source located in AWS S3.

S3 objects are cached locally between runs. The cache location is defined by *cfngin.config.cfngin_cache_dir*.

bucket: `str`

Name of the AWS S3 bucket.

Example

```yaml
package_sources:
  s3:
    - bucket: example-bucket
```

configs: Optional[List[str]] = []

Configuration files from this source location can also be used by specifying a list of file paths.

These configuration files are merged into the current configuration file with the current file taking precedence. When using this usage pattern, it is advised to use dictionary definitions for everything that supports it to allow for granular overriding.

Example

```yaml
package_sources:
  s3:
    - configs:
      - example-01.yml
      - example-02.yml
```

key: str

Key for an S3 object within the *bucket*. The object should be an archived file that can be unzipped.

Example

```yaml
package_sources:
    s3:
      - key: path/to/example.zip
```

paths: Optional[List[str]] = []

A list of subdirectories within the source location that should be added to *$PATH*.

Example

```yaml
package_sources:
  s3:
    - paths:
      - some/directory/
      - another/
```

requester_pays: Optional[bool] = False

Confirms that the requester knows that they will be charged for the request

Example

```
package_sources:
  s3:
    - requester_pays: true
```

use_latest: Optional[bool] = True

Update the local copy if the last modified date in AWS S3 changes.

Example

```
package_sources:
  s3:
    - use_latest: true
```

Templates

CloudFormation templates can be provided via *Blueprints* or JSON/YAML. JSON/YAML templates are specified for `stacks` via the `template_path` config.

Contents

Jinja2 Templating

Templates with a `.j2` extension will be parsed using Jinja2. The CFNgin `context` and `mappings` objects and stack `variables` objects are available for use in the template:

```
Description: TestTemplate
Resources:
  Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: {{ context.environment.foo }}-{{ variables.myparamname }}
```

10.2.4 Migrating from Stacker

Contents

* *Build-in Hooks*
* *Custom Lookups*

Blueprints

Most components available in Stacker 1.7.0 are available in Runway's CFNgin at the same path within `runway.cfngin`.

Example

```
# what use to be this
from stacker.blueprints.base import Blueprint
from stacker.blueprints.variables.types import CFNString

# now becomes this
from runway.cfngin.blueprints.base import Blueprint
from runway.cfngin.blueprints.variables.types import CFNString
```

Config Files

There are some config top-level keys that have changed when used Runway's CFNgin. Below is a table of the Stacker key and what they have been changed to for Runway's CFNgin

Stacker	Runway's CFNgin
`stacker_bucket`	`cfngin_bucket`
`stacker_bucket_region`	`cfngin_bucket_region`
`stacker_cache_dir`	`cfngin_cache_dir`

Build-in Hooks

All hooks available in Stacker 1.7.0 are available in Runway's CFNgin at the same path within `runway.cfngin`.

Note: Some hooks have different *`args`* and/or altered functionality. It is advised to review the documentation for the hook before using it.

Example Definition

```
pre_deploy:
  - path: stacker.hooks.commands.run_command
    args:
      command: echo "Hello $USER!"
  - path: runway.cfngin.hooks.commands.run_command
    args:
      command: echo "Hello $USER!"
```

See also:

runway.cfngin CFNgin documentation

Custom Lookups

See the *Custom Lookups* section of the docs for detailed instructions on how lookups should be written.

10.3 Kubernetes

Kubernetes manifests can be deployed via Runway offering an ideal way to handle core infrastructure-layer (e.g. shared ConfigMaps & Service Accounts) configuration of clusters by using Kustomize overlays.

- *Configuration*
- *Directory Structure*
- *Examples*
- *Advanced Features*

10.3.1 Configuration

Configuration options and parameters for *Kubernetes* modules.

Contents

Options

kubectl_version: Optional[str] = None

Specify a version of Kubectl for Runway and download and use. See *Version Management* for more details.

Example

```
options:
  kubectl_version: 1.14.5
```

overlay_path: Optional[str] = None

Specify the directory containing the kustomize overlay to use.

Example

```yaml
options:
  overlay_path: overlays/${env DEPLOY_ENVIRONMENT}-blue
```

New in version 1.12.0.

Parameters

Kubernetes does not support the use of `deployment.parameters`/`module.parameters` at this time.

10.3.2 Directory Structure

Example directory structures for a *Kubernetes* module.

```
.
├── .gitignore
├── aws-auth-cm.k8s
│   ├── base
│   │   └── kustomization.yaml
│   └── overlays
│       └── template
│           └── kustomization.yaml
├── runway.yml
└── service-hello-world.k8s
    ├── README.md
    ├── base
    │   ├── configMap.yaml
    │   ├── deployment.yaml
    │   ├── kustomization.yaml
    │   └── service.yaml
    └── overlays
        ├── prod
        │   ├── deployment.yaml
        │   └── kustomization.yaml
        └── template
            ├── kustomization.yaml
            └── map.yaml
```

10.3.3 Examples

Example uses of the *Kubernetes* module

Contents

CloudFormation EKS

To view an example using an EKS cluster deployed with CloudFormation, run `runway gen-sample k8s-cfn-repo` in any directory. This will create a `k8s-cfn-infrastructure/` directory in your current working directory containing a `runway.yml` file, some modules that can be deployed, and a `README.md` that provides instructions on how to work with the example code.

Terraform EKS

To view an example using an EKS cluster deployed with Terraform, run `runway gen-sample k8s-tf-repo` in any directory. This will create a `k8s-tf-infrastructure/` directory in your current working directory containing a `runway.yml` file, some modules that can be deployed, and a `README.md` that provides instructions on how to work with the example code.

10.3.4 Advanced Features

Advanced features and detailed information for using Kubernetes with Runway.

Contents

Setting KUBECONFIG Location

If using a non-default kubeconfig location, you can provide it using *deployment.env_vars*/*module.env_vars* for setting environment variables. This can be set as a relative path or an absolute one.

```yaml
deployments:
  - modules:
      - path: myk8smodule
        env_vars:
          KUBECONFIG:
            - .kube
            - ${env DEPLOY_ENVIRONMENT}
            - config
```

This would set *KUBECONFIG* to `<path_to_runway.yml>/.kube/$DEPLOY_ENVIRONMENT/config` where `$DEPLOY_ENVIRONMENT` is the current Runway *deploy environment*.

Version Management

By specifying the version via a `.kubectl-version` file in your overlay directory or *deployment.module_options*/*module.options*, Runway will automatically download & use that version for the module. This is recommended to keep a predictable experience when deploying your module.

Without a version specified, Runway will fallback to whatever `kubectl` it finds first in your PATH.

Listing 18: .kubectl-version

```
1.14.5
```

Lookups can be used to provide different versions for each *deploy environment*.

Listing 19: runway.yml

```
deployments:
  - modules:
      - path: sampleapp.k8s
        options:
          kubectl_version: ${var kubectl_version.${env DEPLOY_ENVIRONMENT}}
  - module:
      - sampleapp-01.k8s
      - sampleapp-02.k8s
    module_options:
      kubectl_version: 1.14.5
```

10.4 Serverless Framework

The Serverless module type is deployed using the Serverless Framework. Runway uses system installed npm to install Serverless per-module. This means that Serverless must be included as a dev dependency in the **package.json** of the module.

- *Configuration*
- *Directory Structure*
- *Advanced Features*

10.4.1 Configuration

Standard Serverless Framework rules apply but, we have some added prerequisites, recommendations, and caveats.

Contents

* *File Naming*

Prerequisites

- npm installed on the system
- Serverless must be a dev dependency of the module (e.g. `npm install --save-dev serverless`)

We strongly recommend you commit the package-lock.json that is generated after running `npm install`.

Options

Options specific to Serverless Framework modules.

`args: Optional[List[str]] = []`

List of CLI arguments/options to pass to the Serverless Framework CLI. See *Specifying Serverless CLI Arguments/Options* for more details.

Example

```yaml
options:
  args:
    - '--config'
    - sls.yml
```

New in version 1.4.0.

`extend_serverless_yml: Optional[Dict[str, Any]] = {}`

If provided, the value of this option will be recursively merged into the modules *serverless.yml* file. See *Extending a Serverless Configuration File* for more details.

Example

```yaml
options:
  extend_serverless_yml:
    custom:
      env:
        memorySize: 512
```

New in version 1.8.0.

`promotezip: Optional[Dict[str, str]] = {}`

If provided, promote Serverless Framework generated zip files between environments from a *build* AWS account. See *Promoting Builds Through Environments* for more details.

Example

```
options:
  promotezip:
    bucketname: my-build-account-bucket-name
```

skip_npm_ci: bool = False

Skip running `npm ci` in the module directory prior to processing the module. See *Disabling NPM CI* for more details.

Example

```
options:
  skip_npm_ci: true
```

serverless.yml

Refer to the Serverless Framework Documentation.

Stages

Runway's concept of a *deploy environment* has a 1-to-1 mapping to Serverless's **stage**. For example, if the deploy environment is **dev**, Serverless will be run with `--stage dev`.

Each stage requires either its own variables file (even if empty for a particular stage) following a specific *File Naming* scheme and/or a configured `environment` for the module or deployment (see *Runway Config File* for details).

File Naming

- `env/STAGE-REGION.yml`
- `config-STAGE-REGION.yml`
- `env/STAGE.yml`
- `config-STAGE.yml`
- `env/STAGE-REGION.json`
- `config-STAGE-REGION.json`
- `env/STAGE.json`
- `config-STAGE.json`

10.4.2 Directory Structure

Example directory structures for a Serverless module.

Contents

Python Example

```
.
├── __init__.py
├── _gitignore
├── config-dev-us-east-1.json
├── hello_world
│   └── __init__.py
├── package.json
├── poetry.lock
├── pyproject.toml
└── serverless.yml
```

TypeScript Example

```
.
├── .gitignore
├── env
│   └── dev-us-east-1
├── jest.config.js
├── package.json
├── package-lock.json
├── serverless.yml
├── src
│   ├── helloWorld.test.ts
│   └── helloWorld.ts
├── tsconfig.json
├── tslint.json
└── webpack.config.js
```

10.4.3 Advanced Features

Advanced features and detailed information for using Serverless Framework with Runway.

Contents

Disabling NPM CI

At the start of each module execution, Runway will execute `npm ci` to ensure Serverless Framework is installed in the project (so Runway can execute it via `npx sls`). This can be disabled (e.g. for use when the `node_modules` directory is pre-compiled) via the `skip_npm_ci` module option.

Example

```yaml
deployments:
  - modules:
      - path: myslsproject.sls
        options:
          skip_npm_ci: true
```

Extending a Serverless Configuration File

Runway has the ability to extend the contents of a *serverless.yml* file using the value of the `extend_serverless_yml` option. The value of this option is recursively merged into a resolved clone of the module's Serverless configuration. To create this resolved clone, Runway uses "serverless print" (including args) to resolve the module's Serverless configuration file and output the contents to a temporary file. The temporary file is deleted after each execution of Runway.

This functionality can be especially useful when used alongside *remote module paths* such as a module from a *git repository* to change values on the fly without needing to modify the source for small differences in each environment.

Example

```yaml
deployments:
  - modules:
      - path: git::git://github.com/onicagroup/example.git//sampleapp?tag=v1.0.0
        options:
          extend_serverless_yml:
            custom:
              env:
                memorySize: 512
    regions:
      - us-east-1
```

Merge Logic

The two data sources are merged by iterating over their content and combining the lowest level nodes possible.

Example

Listing 20: serverless.yml

```yaml
functions:
  example:
    handler: handler.example
    runtime: python3.9
    memorySize: 512
```

Listing 21: runway.yml

```yaml
deployments:
  - modules:
      - path: sampleapp.sls
        options:
          extend_serverless_yml:
            functions:
              example:
                memorySize: 1024
            resources:
              Resources:
                ExampleResource:
                  Type: AWS::CloudFormation::WaitConditionHandle
    regions:
      - us-east-1
```

Listing 22: Resulting serverless.yml

```yaml
functions:
  example:
    handler: handler.example
    runtime: python3.9
```

(continues on next page)

(continued from previous page)

```
    memorySize: 1024
  resources:
    Resources:
      ExampleResource:
        Type: AWS::CloudFormation::WaitConditionHandle
```

Promoting Builds Through Environments

Serverless build `.zips` can be used between environments by setting the `promotezip` module option and providing a bucket name in which to cache the builds.

The first time the Serverless module is deployed using this option, it will build/deploy as normal and cache the artifact on S3. On subsequent deploys, Runway will use the cached artifact (finding it by comparing the module source code).

This enables a common build account to deploy new builds in a dev/test environment, and then promote that same zip through other environments. Any of these environments can be in the same or different AWS accounts.

The CloudFormation Stack deploying the zip will be re-generated on each deployment so environment-specific values/lookups will work as normal.

Example

```
deployments:
  - modules:
      - path: myslsproject.sls
        options:
        promotezip:
          bucketname: my-build-account-bucket-name
```

Specifying Serverless CLI Arguments/Options

Runway can pass custom arguments/options to the Serverless CLI by using the `args` option. These will always be placed after the default arguments/options.

The value of `args` must be a list of arguments/options to pass to the CLI. Each element of the argument/option should be it's own list item (e.g. `--config sls.yml` would be `['--config', 'sls.yml']`).

Important: Do not provide `--region <region>` or `--stage <stage>` here, these will be provided by Runway. Runway will also provide `--no-color` if stdout is not a TTY.

Example

Listing 23: runway.yml

```yaml
deployments:
  - modules:
      - path: sampleapp.sls
        options:
          args:
            - '--config'
            - sls.yml
    regions:
      - us-east-2
    environments:
      example: true
```

Listing 24: Command equivalent

```
$ serverless deploy -r us-east-1 --stage example --config sls.yml
```

10.5 Static Site

This module type performs idempotent deployments of static websites. It combines CloudFormation stacks (for S3 buckets & CloudFront Distribution) with additional logic to build & sync the sites.

A start-to-finish example walkthrough is available in the *Conduit quickstart*.

Note: The CloudFront Distribution that is created by default can take a significant amount of time to spin up on initial deploy (5 to 60 minutes is not abnormal). Incorporating CloudFront with a static site is a common best practice, however, if you are working on a development project it may benefit you to add the *staticsite_cf_disable* parameter.

- *Configuration*
- *Directory Structure*
- *Examples*
- *Advanced Features*

10.5.1 Configuration

Configuration options and parameters for *static site* modules. Example uses of the options and parameters can be found in the *Examples* section.

Contents

Options

build_output: Optional[str] = None

Overrides default directory of top-level path setting.

Example

```yaml
options:
  build_output: dist
```

build_steps: Optional[List[str]] = []

The steps to run during the build portion of deployment.

Example

```yaml
options:
  build_steps:
    - npm ci
    - npm run build
```

extra_files: Optional[List[Dict[str, Union[str, Dict[str, Any]]]]] = []

Specifies extra files that should be uploaded to S3 after the build.

Use `extra_files` if you want to have a single build artifact that can be used in many environments. These files should be excluded from source hashing and the build system. The end result is that you have a build artifact that can be deployed in any environment and behave exactly the same.

See *Extra Files* for more details.

Example

```yaml
options:
  extra_files:
    - name: config.json # yaml or other text files are supported
      content: # this object will be json or yaml serialized
        endpoint: ${var api_endpoint.${env DEPLOY_ENVIRONMENT}}
    - name: config.any
      content_type: text/yaml # Explicit content type
      content:
        endpoint: ${var api_endpoint.${env DEPLOY_ENVIRONMENT}}
    - name: logo.png
      content_type: image/png
      file: logo-${env DEPLOY_ENVIRONMENT}.png # a reference to an existing file
```

The example above produces a file named `config.json` with the contents below and a `logo.png` file.

```
{
  "endpoint": "<api_endpoint value>"
}
```

New in version 1.9.0.

pre_build_steps: Optional[List[Dict[str, str]]] = []

Commands to be run before generating the hash of files.

Example

```yaml
options:
  pre_build_steps:
    - command: npm ci
      cwd: ../myothermodule # directory relative to top-level path setting
    - command: npm run export
      cwd: ../myothermodule
```

source_hashing: Optional[Dict[str, str]] = {}

Overrides for source hash collection and tracking

Example

```yaml
options:
  source_hashing:
    enabled: true # if false, build & upload will occur on every deploy
    parameter: /${namespace}/myparam # defaults to <namespace>-<name/path>-hash
    directories: # overrides default hash directory of top-level path setting
      - path: ./
      - path: ../common
        # Additional (gitignore-format) exclusions to
        # hashing (.gitignore files are loaded automatically)
        exclusions:
          - foo/*
```

Parameters

cloudformation_service_role: Optional[str] = None

IAM Role ARN that CloudFormation will use when creating, deleting and updating the CloudFormation stack resources.

See the AWS CloudFormation service role for more information.

Example

```yaml
parameters:
  cloudformation_service_role: arn:aws:iam::123456789012:role/name
```

namespace: str

The unique namespace for the deployment.

Example

```
parameters:
  namespace: my-awesome-website-${env DEPLOY_ENVIRONMENT}
```

staticsite_acmcert_arn: Optional[str] = None

The certificate arn used for any alias domains supplied. This is a requirement when supplying any custom domain.

Example

```
parameters:
  staticsite_acmcert_arn: arn:aws:acm:<region>:<account-id>:certificate/<cert>
```

staticsite_aliases: Optional[str] = None

Any custom domains that should be added to the CloudFront Distribution. This should be represented as a comma delimited list of domains.

Requires *staticsite_acmcert_arn*.

Example

```
parameters:
  staticsite_aliases: example.com,foo.example.com
```

staticsite_auth_at_edge: Optional[bool] = False

Auth@Edge make the static site *private* by placing it behind an authorization wall. See *Auth@Edge* for more details.

Example

```
parameters:
  staticsite_auth_at_edge: true
```

New in version 1.5.0.

staticsite_cf_disable: Optional[bool] = False

Whether deployment of the CloudFront Distribution should be disabled.

Useful for a development site as it makes it accessible via an S3 url with a much shorter launch time. This cannot be set to `true` when using *Auth@Edge*.

Example

```
parameters:
  staticsite_cf_disable: false
```

New in version 1.5.0.

staticsite_compress: Optional[bool] = True

Whether the CloudFront default cache behavior will automatically compress certain files.

Example

```yaml
parameters:
  staticsite_compress: false
```

staticsite_cookie_settings: Optional[Dict[str, str]] = {"idToken": "Path=/; Secure; SameSite=Lax", "accessToken": "Path=/; Secure; SameSite=Lax", "refreshToken": "Path=/; Secure; SameSite=Lax", "nonce": "Path=/; Secure; HttpOnly; Max-Age=1800; SameSite=Lax"}

The default cookie settings for retrieved tokens and generated nonce's.

Requires *staticsite_auth_at_edge*.

Example

```yaml
parameters:
  staticsite_cookie_settings:
    idToken: "Path=/; Secure; SameSite=Lax"
    accessToken: "Path=/; Secure; SameSite=Lax"
    refreshToken: "Path=/; Secure; SameSite=Lax"
    nonce: "Path=/; Secure; HttpOnly; Max-Age=1800; SameSite=Lax"
```

New in version 1.5.0.

staticsite_create_user_pool: Optional[bool] = False

Whether to create a User Pool for the *Auth@Edge* configuration.

Requires *staticsite_auth_at_edge*.

Example

```yaml
parameters:
  staticsite_create_user_pool: true
```

New in version 1.5.0.

staticsite_custom_error_responses: Optional[List[Dict[str, Union[int, str]]]] = []

Define custom error responses.

Example

```yaml
parameters:
  staticsite_custom_error_responses:
    - ErrorCode: 404
      ResponseCode: 200
      ResponsePagePath: /index.html
```

staticsite_enable_cf_logging: Optional[bool] = True

Whether logging should be enabled for the CloudFront distribution.

Example

```yaml
parameters:
  staticsite_enable_cf_logging: true
```

staticsite_http_headers: Optional[Dict[str, str]] = {"Content-Security-Policy": "default-src https: 'unsafe-eval' 'unsafe-inline'; font-src 'self' 'unsafe-inline' 'unsafe-eval' data: https:; object-src 'none'; connect-src 'self' https://*.amazonaws.com https://*.amazoncognito.com", "Strict-Transport-Security": "max-age=31536000; includeSubdomains; preload", "Referrer-Policy": "same-origin", "X-XSS-Protection": "1; mode=block", "X-Frame-Options": "DENY", "X-Content-Type-Options": "nosniff"}

Headers that should be sent with each origin response.

Requires *staticsite_auth_at_edge*.

Note: Please note that the Content-Security-Policy is intentionally lax to allow for Single Page Application framework's to work as expected. Review your Content Security Policy for your project and update these as need be to match.

Example

```yaml
parameters:
  staticsite_http_headers:
    Content-Security-Policy: "default-src https: 'unsafe-eval' 'unsafe-inline';␣
↪font-src 'self' 'unsafe-inline' 'unsafe-eval' data: https:; object-src 'none';␣
↪connect-src 'self' https://*.amazonaws.com https://*.amazoncognito.com"
    Strict-Transport-Security: "max-age=31536000; includeSubdomains; preload"
    Referrer-Policy: "same-origin"
    X-XSS-Protection: "1; mode=block"
    X-Frame-Options: "DENY"
    X-Content-Type-Options: "nosniff"
```

New in version 1.5.0.

staticsite_lambda_function_associations: Optional[List[Dict[str, str]]] = []

This section allows the user to deploy custom *Lambda@Edge* associations with their pre-build function versions. This takes precedence over *staticsite_rewrite_directory_index* and cannot currently be used with *staticsite_auth_at_edge*.

Example

```yaml
parameters:
  staticsite_lambda_function_associations:
    - type: origin-request
      arn: arn:aws:lambda:<region>:<account-id>:function:<function>:<version>
```

staticsite_non_spa: Optional[bool] = False

Whether this site is a single page application (*SPA*).

A custom error response directing `ErrorCode: 404` to the primary `/index.html` as a `ResponseCode: 200` is added, allowing the *SPA* to take over error handling. If you are not running an *SPA*, setting this to `true` will prevent this custom error from being added. If provided, *staticsite_custom_error_responses* takes precedence over this setting.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_non_spa: true
```

New in version 1.5.0.

staticsite_oauth_scopes: Optional[List[str]] = ["phone", "email", "profile", "openid", "aws.cognito.signin.user.admin"]

Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes. This information is then presented to the user in the consent screen and the access token issued to the application will be limited to the scopes granted.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_oauth_scopes:
    - phone
    - email
    - profile
    - openid
    - aws.cognito.signin.user.admin
```

New in version 1.5.0.

staticsite_redirect_path_auth_refresh: Optional[str] = "/refreshauth"

The path that a user is redirected to when their authorization tokens have expired (1 hour).

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_redirect_path_auth_refresh: /refreshauth
```

New in version 1.5.0.

staticsite_redirect_path_sign_in: Optional[str] = "/parseauth"

The path that a user is redirected to after sign-in. This corresponds with the `parseauth` *Lambda@Edge* function which will parse the authentication details and verify the reception.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_redirect_path_sign_in: /parseauth
```

New in version 1.5.0.

staticsite_redirect_path_sign_out: Optional[str] = "/"

The path that a user is redirected to after sign-out. This typically should be the root of the site as the user will be asked to re-login.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_redirect_path_sign_out: /
```

New in version 1.5.0.

staticsite_rewrite_directory_index: Optional[str] = None

Deploy a *Lambda@Edge* function designed to rewrite directory indexes, e.g. supports accessing urls such as `example.org/foo/`

Example

```
parameters:
  staticsite_rewrite_directory_index: index.html
```

staticsite_role_boundary_arn: Optional[str] = None

Defines an IAM Managed Policy that will be set as the permissions boundary for any IAM Roles created to support the site. (e.g. when using *staticsite_auth_at_edge* or *staticsite_rewrite_directory_index*)

Example

```
parameters:
  staticsite_role_boundary_arn: arn:aws:iam::<account-id>:policy/<policy>
```

New in version 1.8.0.

staticsite_sign_out_url: Optional[str] = "/signout"

The path a user should access to sign themselves out of the application.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_sign_out_url: /signout
```

New in version 1.5.0.

staticsite_supported_identity_providers: Optional[str] = "COGNITO"

A comma delimited list of the User Pool client identity providers.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_supported_identity_providers: facebook,onelogin
```

New in version 1.5.0.

staticsite_user_pool_arn: Optional[str] = None

The ARN of a pre-existing Cognito User Pool to use with *Auth@Edge*.

Requires *staticsite_auth_at_edge*.

Example

```
parameters
  staticsite_user_pool_arn: arn:aws:cognito-idp:<region>:<account-id>:userpool/
↪<pool>
```

New in version 1.5.0.

staticsite_additional_redirect_domains: Optional[str] = None

Additional domains (beyond the *staticsite_aliases* domains or the CloudFront URL if no aliases are provided) that will be authorized by the *Auth@Edge* UserPool AppClient. This parameter typically won't be needed in production environments, but can be useful in development environments to allow bypassing Runway Auth@Edge.

This should be represented as a comma delimited list of domains with protocols. Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_additional_redirect_domains: http://localhost:3000
```

New in version 1.14.0.

staticsite_web_acl: Optional[str] = None

The ARN of a web access control list (web ACL) to associate with the CloudFront Distribution.

Example

```
parameters:
  staticsite_web_acl: arn:aws:waf::<account-id>:certificate/<cert>
```

staticsite_required_group: Optional[str] = None

Name of Cognito User Pool group of which users must be a member to be granted access to the site. Omit to allow all UserPool users to have access.

Requires *staticsite_auth_at_edge*.

Example

```
parameters:
  staticsite_required_group: AuthorizedUsers
```

New in version 1.5.0.

10.5.2 Directory Structure

Example directory structures for a ref:*static site <mod-staticsite>* module.

Contents

Angular SPA

```
.
├── runway.yml
└── sample-app
    ├── README.md
    ├── .gitignore
    ├── angular.json
    ├── browserslist
    ├── e2e
    │   ├── protractor.conf.js
    │   ├── src
    │   │   ├── app.e2e-spec.ts
    │   │   └── app.po.ts
    │   └── tsconfig.json
    ├── karma.conf.js
    ├── package-lock.json
    ├── package.json
    ├── src
    │   ├── app
```

(continues on next page)

(continued from previous page)

```
    │   │   ├── app-routing.module.ts
    │   │   ├── app.component.css
    │   │   ├── app.component.html
    │   │   ├── app.component.spec.ts
    │   │   ├── app.component.ts
    │   │   └── app.module.ts
    │   ├── assets
    │   ├── environments
    │   │   ├── environment.prod.ts
    │   │   └── environment.ts
    │   ├── favicon.ico
    │   ├── index.html
    │   ├── main.ts
    │   ├── polyfills.ts
    │   ├── styles.css
    │   └── test.ts
    ├── tsconfig.app.json
    ├── tsconfig.json
    ├── tsconfig.spec.json
    └── tslint.json
```

React SPA

```
.
├── runway.yml
└── sample-app
    ├── README.md
    ├── package.json
    ├── public
    │   ├── favicon.ico
    │   ├── index.html
    │   ├── logo192.png
    │   ├── logo512.png
    │   ├── manifest.json
    │   └── robots.txt
    └── src
        ├── App.css
        ├── App.js
        ├── App.test.js
        ├── index.css
        ├── index.js
        ├── logo.svg
        ├── serviceWorker.js
        └── setupTests.js
```

10.5.3 Examples

Example uses of the *static site* module

Contents

Angular SPA

To view an example deployment of an Angular single page application, run `runway gen-sample static-angular` in any directory. This will create a new `static-angular/` directory in your current working directory containing a `runway.yml` and a `sample-app` module that can be deployed.

Extra Files in Angular

By default, angular uses `environment/environment.ts` as its way to pass environment specific configurations into your application. The downside to this is that you need to build your application for each environment and lose ability to build once and deploy everywhere.

The static site `extra_files` option solves this problem by moving environment configuration out of angular and into Runway. A small change to the way the application references environment config will need to be made.

1. Wrap configuration access into a service that can be injected into your components.
2. In the new config service, make an http request to load the config. Since `extra_files` uploads the files to the static site bucket, this request should be relative to your application.
3. Cache the config and use normal observable patterns to provide access.

Listing 25: app.config.ts

```
export interface Config {
  endpoint: string;
}

@Injectable({
  providedIn: 'root'
})
export class AppConfig {
  constructor(private http: HttpClient) {}

  getConfig(): Observable<Config> {
```

(continues on next page)

(continued from previous page)

```
    return this.http.get<Config>('assets/config.json');
  }
}
```

Listing 26: app.component.ts

```
export class AppComponent implements OnInit {
  title = 'Angular App';

  constructor(private config: AppConfig) {}

  ngOnInit() {
    this.config.getConfig().subscribe((config) => {
      this.title = config.endpoint;
    });
  }
}
```

Listing 27: runway.yml

```
variables:
  website:
    api_endpoint:
      dev: https://api.dev.example.com
      test: https://api.test.example.com

deployments:
  - name: WebApp
    modules:
      - path: .
        type: static
        environments:
          dev: true
          test: true
        options:
          build_steps:
            - npm ci
            - npm run build
          build_output: dist/web
          extra_files:
            - name: assets/config.json
              content:
                endpoint: ${var website.api_endpoint.${env DEPLOY_ENVIRONMENT}}
        parameters:
          namespace: my-app-namespace
          staticsite_cf_disable: true
    regions:
      - us-east-1
```

Angular Development Workflow

While developing an Angular application, a local live environment is typically used and Runway is not. This means that `assets/config.json` does not exist and your application would likely fail. Take the following steps to get your development environment running.

1. Create a stub `src/assets/config.json` that defines all the configuration attributes. The values can be empty strings.
2. Create a 'dev' config file: `src/assets/config-dev.json`. Populate the configuration values with appropriate values for your local dev environment.
3. Edit `angular.json`
   - Add a `fileReplacements` option to `projects.<app>.architect.build.options`.

```
{
  "fileReplacements": [{
    "replace": "src/assets/config.json",
    "with": "src/assets/config-dev.json"
  }]
}
```

4. Run `npx ng serve`

Note: It would be a better practice to define a new 'local' configuration target instead of adding `fileReplacements` to the default configuration target.

Listing 28: "build" Configuration

```
{
  "configurations": {
    "local": {
      "fileReplacements": [{
        "replace": "src/assets/config.json",
        "with": "src/assets/config-local.json"
      }]
    }
  }
}
```

Listing 29: "serve" Configuration

```
{
  "configurations": {
    "local": {
      "browserTarget": "<app>:build:local"
    }
  }
}
```

```
$ npx ng serve --configuration=local
```

React SPA

To view an example deployment of a React single page application, run `runway gen-sample static-react` in any directory. This will create a new `static-react/` directory in your current working directory containing a `runway.yml` and a `sample-app` module that can be deployed.

Extra Files in React

React by itself is not concerned with different environments or how a developer initializes the application with different backends. This is more of a concern with other layers of your application stack, e.g. Redux. However, the concept is similar to the Angular examples.

Listing 30: Plain React

```
// Use your favorite http client
import axios from 'axios';

// Make a request to load the config
axios.get('config.json').then(resp => {
  return resp.data.endpoint;
})
.then(endpoint => {
  // Render the react component
  ReactDOM.render(<App message={endpoint} />, document.getElementById('root'));
});
```

Initialize the redux store with an initial config

Listing 31: React Redux

```
axios.get('config.json').then(resp => {
  return resp.data;
})
.then(config => {
  // Create a redux store
  return store(config);
})
.then(store => {
  ReactDOM.render(
    <Provider store={store}>
      <App/>
    </Provider>,
    document.getElementById('root')
  );
});
```

Listing 32: runway.yml

```
variables:
  website:
    api_endpoint:
      dev: https://api.dev.example.com
      test: https://api.test.example.com
```

(continues on next page)

(continued from previous page)

```yaml
deployments:
  - name: WebApp
    modules:
      - path: .
        type: static
        environments:
          dev: true
          test: true
        options:
          build_output: build
          build_steps:
            - npm ci
            - npm run build
          extra_files:
            - name: config.json
              content:
                endpoint: ${var website.api_endpoint.${env DEPLOY_ENVIRONMENT}}
        parameters:
          namespace: my-app-namespace
          staticsite_cf_disable: true
    regions:
      - us-west-2
```

React Development Workflow

React doesn't have an equivalent feature as Angular's fileReplacements so this solution isn't as flexible.

1. Create the file `public/config.json`.

   Add content that matches the structure defined in `extra_files` and populate the values needed for local development.

   #### Example

   ```
   {
     "endpoint": "https://api.dev.example.com"
   }
   ```

2. *(Optional)* Add `public/config.json` to `.gitignore`

   Note: If you don't want to add `public/config.json` to `.gitignore`, you should configure Runways source hashing to exclude it.

   ```yaml
   source_hashing:
     enabled: true
     directories:
       - path: ./
         exclusions:
           - public/config.json
   ```

3. Run `npm run start`

10.5.4 Advanced Features

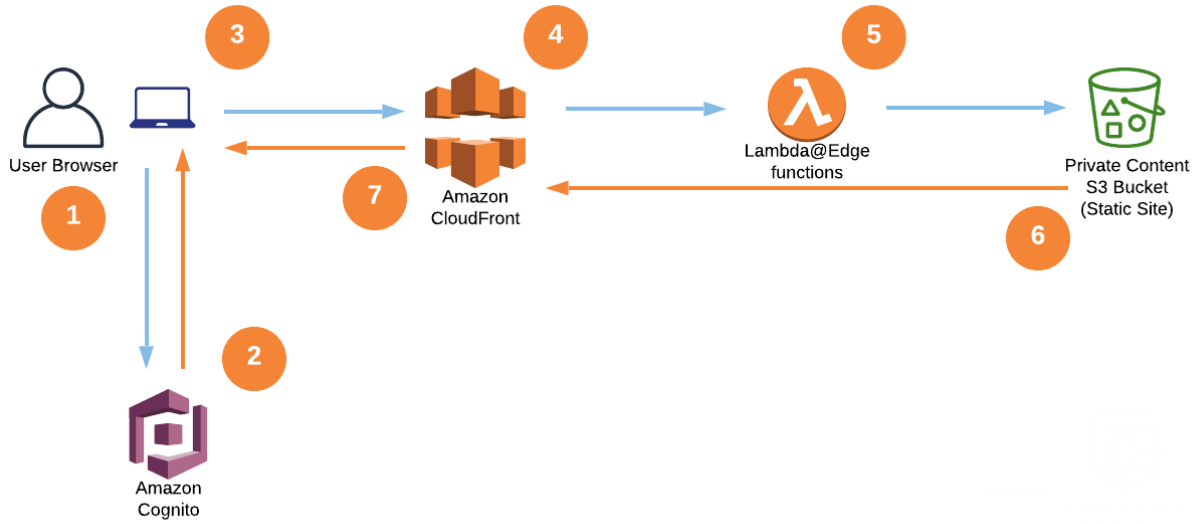
Contents

Auth@Edge

Important: *Auth@Edge* static sites can only be deployed to us-east-1. This is due to the limitations of *Lambda@Edge*.

Auth@Edge provides the ability to make a static site private by using Cognito for authentication. The solution is inspired by similar implementations such as aws-samples/cloudfront-authorization-at-edge.

The following diagram depicts a high-level overview of this solution.



Here is how the solution works:

1. The viewer's web browser is redirected to Amazon Cognito custom UI page to sign up and authenticate.
2. After authentication, Cognito generates and cryptographically signs a JWT then responds with a redirect containing the JWT embedded in the URL.
3. The viewer's web browser extracts JWT from the URL and makes a request to private content (private/* path), adding Authorization request header with JWT.

4. Amazon CloudFront routes the request to the nearest AWS edge location. The CloudFront distribution's private behavior is configured to launch a *Lambda@Edge* function on ViewerRequest event.
5. *Lambda@Edge* decodes the JWT and checks if the user belongs to the correct Cognito User Pool. It also verifies the cryptographic signature using the public RSA key for Cognito User Pool. Crypto verification ensures that JWT was created by the trusted party.
6. After passing all of the verification steps, *Lambda@Edge* strips out the Authorization header and allows the request to pass through to designated origin for CloudFront. In this case, the origin is the private content Amazon S3 bucket.
7. After receiving response from the origin S3 bucket, CloudFront sends the response back to the browser. The browser displays the data from the returned response.

An example of an *Auth@Edge* static site configuration is as follows:

```
variables:
  dev:
    namespace: sample-app-dev
    staticsite_user_pool_arn: arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-
↪east-1_example

deployments:
  - modules:
    - path: sampleapp
      type: static
      parameters:
        namespace: ${var ${env DEPLOY_ENVIRONMENT}.namespace}
        staticsite_auth_at_edge: true
        staticsite_user_pool_arn: ${var ${env DEPLOY_ENVIRONMENT}.staticsite_user_pool_
↪arn}
    regions:
      - us-east-1
```

The *Auth@Edge* functionality uses an existing Cognito User Pool (optionally configured with federated identity providers) or can create one for you with the *staticsite_create_user_pool* option. A user pool app client will be automatically created within the pool for use with the application.

Extra Files

The extra files option allows you to use a single build across many deployments. Some popular front end frameworks guide you into including environment specific parameters as part of the build(e.g. Angular and Redux-React). This forces you to abandon 12 factor principles and slows down deployments to other environments.

The static site `extra_files` option solves this problem by moving environment configuration out of your code and into Runway. A small change to the way the application references environment config will need to be made.

1. While bootstrapping or early in the application lifecycle, make an HTTP call to load one of the `extra_files`.
2. Make the content of the `extra_file` available to your app using an appropriate abstraction.

See *Static Site Examples* to see how to do this in Angular and React.

Configuration (`extra_files` list item)

`name: str`

The destination name of the file to create.

`file: Optional[str] = None`

A reference to an existing file. The content of this file will be uploaded to the static site S3 bucket using the name as the object key. This or `content` must be specified.

`content_type: Optional[str] = None`

An explicit content type of the file. If not given, the content type will be auto detected based on the name. Only `.json`, `.yml`, and `.yaml` extensions are recognized automatically.

- `application/json` to serialize `content` into JSON.
- `text/yaml` to serialize `content` into YAML.

`content: Optional[Union[str, List[Any], Dict[str, Any]]] = None`

Inline content that will be used as the file content. This or `file` must be specified.

Note: If none of the files or content changed between builds and source hashing is enabled, the upload will be skipped.

10.6 Terraform

Runway provides a simple way to run the Terraform versions you want with variable values specific to each environment. Terraform does not need to be installed prior to using this module type. Runway maintains a cache of Terraform versions on a system, downloading and installing different versions as needed.

- *Configuration*
- *Directory Structure*
- *Advanced Features*

10.6.1 Configuration

Contents

Options

Options specific to Terraform Modules.

args: Optional[Union[Dict[str, List[str]], List[str]]] = None

List of CLI arguments/options to pass to Terraform. See *Specifying Terraform CLI Arguments/Options* for more details.

Example

```yaml
options:
  args:
    - '-parallelism=25'
```

New in version 1.8.1.

terraform_backend_config: Optional[Dict[str, str]] = {}

Mapping to configure Terraform backend. See *Backend* for more details.

Example

```yaml
options:
  terraform_backend_config:
    bucket: mybucket
    dynamodb_table: mytable
    region: us-east-1
```

Changed in version 1.11.0: Added support for any *key: value*.

terraform_version: Optional[str] = None

String containing the Terraform version or a mapping of deploy environment to a Terraform version. See *Version Management* for more details.

Example

```yaml
options:
  terraform_version: 0.11.13
```

terraform_write_auto_tfvars: Optional[bool] = False

Optionally write parameters to a tfvars file instead of updating variables. This can be useful in cases where Runway may not be parsing/passing parameters as expected.

When `True`, Runway creates a temporary `runway-parameters.auto.tfvars.json` file in the module directory. This file contains all of the modules parameters in JSON format. This file is then automatically loaded by Terraform as needed. If using a remote backend, use of this file to pass variables is required as environment variables are not available from the CLI and `-var-file` currently cannot be used. Once the module has finished processing, the file is deleted.

Example

```yaml
options:
  terraform_write_auto_tfvars: true
```

New in version 1.11.0.

Variables

Variables can be defined per-environment using one or both of the following options.

tfvars

Standard Terraform tfvars files can be used, exactly as one normally would with `terraform apply -var-file`. Runway will automatically detect them when named like `ENV-REGION.tfvars` or `ENV.tfvars`.

Example

Listing 33: common-us-east-1.tfvars

```
region = "us-east-1"
image_id = "ami-abc123"
```

runway.yml

Variable values can also be specified as *deployment.parameters*/*module.parameters* values in runway.yml. It is recommended to use *Lookups* in the `parameters` section to assist in selecting the appropriate values for the deploy environment and/or region being deployed to but, this is not a requirement if the value will remain the same.

```yaml
deployments:
  - modules:
      - path: sampleapp-01.tf
        parameters:
          region: ${env AWS_REGION}
          image_id: ${var image_id.${env AWS_REGION}}
          my_list:
            - item1
            - item2
          my_map:
            key1: value1
            key2: value1
  - modules:
      - path: sampleapp-02.tf
    parameters:
      region: ${env AWS_REGION}
      image_id: ${var image_id.${env AWS_REGION}}
      my_list:
        - item1
        - item2
```

(continues on next page)

(continued from previous page)

```
    my_map:
      key1: value1
      key2: value1
```

10.6.2 Directory Structure

Example directory structures for a Terraform module.

Example

```
.
├── .terraform-version
├── backend-us-east-1.tfvars
├── dev-us-east-1.tfvars
└── main.tf
```

10.6.3 Advanced Features

Advanced features and detailed information for using Terraform with Runway.

Contents

Backend Configuration

If your Terraform will only ever be used with a single backend, it can be defined inline.

main.tf

```
terraform {
  backend "s3" {
    region = "us-east-1"
    key = "some_unique_identifier_for_my_module"
    bucket = "some_s3_bucket_name"
    dynamodb_table = "some_ddb_table_name"
  }
}
```

However, it's generally preferable to separate the backend configuration out from the rest of the Terraform code. This form of configuration is known as partial configuration and allows for dynamic or secret values to be passed in at runtime.

Below are examples of how to implement partial configuration with Runway. All examples provided showcase the use of the s3 backend type as it is the easiest to use when going from zero to deployed (try *runway gen-sample cfngin* for quickstart Terraform backend infrastructure). However, Runway supports the use of any backend type (refer to Terraform's documentation for proper partial configuration instructions).

See also:

https://www.terraform.io/docs/backends/config.html#partial-configuration Terraform partial configuration

https://www.terraform.io/docs/backends/types/index.html Terraform backend types

Backend Config File

Backend config options can be specified in a separate file or multiple files per environment and/or region using one of the following naming schemes.

- *backend-ENV-REGION.(hcl|tfvars)*
- *backend-ENV.(hcl|tfvars)*
- *backend-REGION.(hcl|tfvars)*
- *backend.(hcl|tfvars)*

Example

```
region = "us-east-1"
bucket = "some_s3_bucket_name"
dynamodb_table = "some_ddb_table_name"
```

In the above example, where all but the key are defined, the **main.tf** backend definition is reduced to the following.

main.tf

```
terraform {
  backend "s3" {
    key = "some_unique_identifier_for_my_module"
  }
}
```

Changed in version 1.11.0: Added support for hcl files.

runway.yml

Backend config options can also be specified as a module option in the Runway Config File. *Lookups* can be used to provide dynamic values to this option.

Important: There has been a *bug* since Terraform 0.12 that prevents passing blocks to `-backend-config` (issue). This means that for backends that use blocks in their config (e.g. remote), the blocks must be provided via file. Attributes are unaffected.

Listing 34: backend.hcl

```
workspaces {
  prefix = "example-"
}
```

Listing 35: Module Level

```
deployments:
  - modules:
      - path: sampleapp-01.tf
        options:
          terraform_backend_config:
            bucket: mybucket
            dynamodb_table: mytable
            region: us-east-1
      - path: sampleapp-02.tf
        options:
          terraform_backend_config:
            bucket: ${cfn common-tf-state.TerraformStateBucketName}
            dynamodb_table: ${cfn common-tf-state.TerraformStateTableName}
            region: ${env AWS_REGION}
```

Listing 36: Deployment Level

```yaml
deployments:
  - modules:
      - path: sampleapp-01.tf
      - path: sampleapp-02.tf
    module_options:  # shared between all modules in deployment
      terraform_backend_config:
        bucket: ${ssm ParamNameHere::region=us-east-1}
        dynamodb_table: ${ssm ParamNameHere::region=us-east-1}
        region: ${env AWS_REGION}
```

Specifying Terraform CLI Arguments/Options

Runway can pass custom arguments/options to the Terraform CLI by using the `args` option.

The value of `args` can be provided in one of two ways. The simplest way is to provide a *list* of arguments/options which will be appended to `terraform apply` when executed by Runway. Each element of the argument/option should be it's own list item (e.g. `-parallelism=25 -no-color` would be `['-parallelism=25, '-no-color']`).

For more control, a map can be provided to pass arguments/options to other commands. Arguments can be passed to `terraform apply`, `terraform init`, and/or `terraform plan` by using the *action* as the key in the map (see the **Runway Example** section below). The value of each key in the map must be a list as described in the previous section.

Important: The following arguments/options are provided by Runway and should not be provided manually: *auto-approve*, *backend-config*, *force*, *no-color*, *reconfigure*, *update*, and *var-file*. Providing any of these manually could result in unintended side-effects.

Listing 37: Runway Example

```yaml
deployments:
  - modules:
      - path: sampleapp-01.tf
        options:
          args:
            - '-no-color'
            - '-parallelism=25'
      - path: sampleapp-02.tf
        options:
          args:
            apply:
              - '-no-color'
              - '-parallelism=25'
            init:
              - '-no-color'
            plan:
              - '-no-color'
              - '-parallelism=25'
    regions:
      - us-east-2
```

(continues on next page)

(continued from previous page)

```
    environments:
      example: true
```

Listing 38: Command Equivalent

```
# runway deploy - sampleapp-01.tf
terraform init -reconfigure
terraform apply -no-color -parallelism=25 -auto-approve=false

# runway plan - sampleapp-01.tf
terraform plan

# runway deploy - sampleapp-02.tf
terraform init -reconfigure -no-color
terraform apply -no-color -parallelism=25 -auto-approve=false

# runway plan - sampleapp-02.tf
terraform plan -no-color -parallelism=25
```

Version Management

By specifying which version of Terraform to use via a `.terraform-version` file in your module directory or in *deployment.module_options*/*module.options*, Runway will automatically download & use that version for the module. This, alongside tightly pinning Terraform provider versions, is highly recommended to keep a predictable experience when deploying your module.

Listing 39: .terraform-version

```
0.11.6
```

Listing 40: runway.yml

```yaml
deployments:
  - modules:
      - path: sampleapp-01.tf
        options:
          terraform_version: 0.11.13
```

Without a version specified, Runway will fallback to whatever `terraform` it finds first in your PATH.

10.6.4 API Docs

runway package

Set package version.

Subpackages

runway.aws_sso_botocore package

Support for AWS SSO.

IMPORTANT: This will be removed upon botocore/boto3 officially supporting authentication with AWS SSO profiles.

The code in this directory is taken from https://github.com/boto/botocore/tree/bf6d31f42f90a547707df4e83943702beacda45a and modified to meet the requirements of this project.

Submodules

runway.aws_sso_botocore.credentials module

Botocore with support for AWS SSO credential assets.

`runway.aws_sso_botocore.credentials.`**`create_credential_resolver`**(*session*, *cache=None*, *region_name=None*)

Create a default credential resolver.

This creates a pre-configured credential resolver that includes the default lookup chain for credentials.

`class` `runway.aws_sso_botocore.credentials.`**`ProfileProviderBuilder`**

Bases: `botocore.credentials.ProfileProviderBuilder`

Extends the botocore profile provider builder to support AWS SSO.

`__init__`(*session*, *cache=None*, *region_name=None*, *sso_token_cache=None*)

Instantiate class.

`providers`(*profile_name*, *disable_env_vars=False*)

Return list of providers.

__new__(**kwargs)

class runway.aws_sso_botocore.credentials.SSOCredentialFetcher

Bases: botocore.credentials.CachedCredentialFetcher

AWS SSO credential fetcher.

__init__(start_url, sso_region, role_name, account_id, client_creator, token_loader=None, cache=None, expiry_window_seconds=None)

Instantiate class.

__new__(**kwargs)

class runway.aws_sso_botocore.credentials.SSOProvider

Bases: botocore.credentials.CredentialProvider

AWS SSO credential provider.

__init__(load_config, client_creator, profile_name, cache=None, token_cache=None)

Instantiate class.

load()

Load AWS SSO credentials.

__new__(**kwargs)

runway.aws_sso_botocore.exceptions module

Botocore with support for AWS SSO exceptions.

exception runway.aws_sso_botocore.exceptions.SSOError

Bases: botocore.exceptions.BotoCoreError

Base class for AWS SSO authentication errors.

__init__(**kwargs)

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.aws_sso_botocore.exceptions.PendingAuthorizationExpiredError

Bases: *runway.aws_sso_botocore.exceptions.SSOError*

Pending AWS SSO authorization expired.

__init__(**kwargs)

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.aws_sso_botocore.exceptions.SSOTokenLoadError

Bases: *runway.aws_sso_botocore.exceptions.SSOError*

AWS SSO token load error.

__init__(***kwargs*)

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.aws_sso_botocore.exceptions.**UnauthorizedSSOTokenError**

Bases: *runway.aws_sso_botocore.exceptions.SSOError*

Unauthorized AWS SSO token.

__init__(***kwargs*)

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

runway.aws_sso_botocore.session module

Botocore with support for AWS SSO session assets.

class runway.aws_sso_botocore.session.**Session**

Bases: botocore.session.Session

Extends the botocore session to support AWS SSO.

__init__(*session_vars=None*, *event_hooks=None*, *include_builtin_handlers=True*, *profile=None*)

Create a new Session object.

Parameters

- **session_vars** (*dict*) – A dictionary that is used to override some or all of the environment variables associated with this session. The key/value pairs defined in this dictionary will override the corresponding variables defined in SESSION_VARIABLES.
- **event_hooks** (*BaseEventHooks*) – The event hooks object to use. If one is not provided, an event hooks object will be automatically created for you.
- **include_builtin_handlers** (*bool*) – Indicates whether or not to automatically register builtin handlers.
- **profile** (*str*) – The name of the profile to use for this session. Note that the profile can only be set when the session is created.

__new__(***kwargs*)

create_client(*service_name*, *region_name=None*, *api_version=None*, *use_ssl=True*, *verify=None*, *endpoint_url=None*, *aws_access_key_id=None*, *aws_secret_access_key=None*, *aws_session_token=None*, *config=None*)

Create a botocore client.

Parameters

- **service_name** (*string*) – The name of the service for which a client will be created. You can use the Session.get_available_services() method to get a list of all available service names.

- **region_name** (*string*) – The name of the region associated with the client. A client is associated with a single region.
- **api_version** (*string*) – The API version to use. By default, botocore will use the latest API version when creating a client. You only need to specify this parameter if you want to use a previous API version of the client.
- **use_ssl** (*boolean*) – Whether or not to use SSL. By default, SSL is used. Note that not all services support non-ssl connections.
- **verify** (*boolean/string*) – Whether or not to verify SSL certificates. By default SSL certificates are verified. You can provide the following values:
  - False - do not validate SSL certificates. SSL will still be used (unless use_ssl is False), but SSL certificates will not be verified.
  - path/to/cert/bundle.pem - A filename of the CA cert bundle to uses. You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.
- **endpoint_url** (*string*) – The complete URL to use for the constructed client. Normally, botocore will automatically construct the appropriate URL to use when communicating with a service. You can specify a complete URL (including the "http/https" scheme) to override this behavior. If this value is provided, then `use_ssl` is ignored.
- **aws_access_key_id** (*string*) – The access key to use when creating the client. This is entirely optional, and if not provided, the credentials configured for the session will automatically be used. You only need to provide this argument if you want to override the credentials used for this specific client.
- **aws_secret_access_key** (*string*) – The secret key to use when creating the client. Same semantics as aws_access_key_id above.
- **aws_session_token** (*string*) – The session token to use when creating the client. Same semantics as aws_access_key_id above.
- **config** (*botocore.client.Config*) – Advanced client configuration options. If a value is specified in the client config, its value will take precedence over environment variables and configuration values, but not over a value passed explicitly to the method. If a default config object is set on the session, the config object used when creating the client will be the result of calling `merge()` on the default config with the config provided to this call.

Return type botocore.client.BaseClient

Returns A botocore client instance

`property full_config`

Return the parsed config file.

The `get_config` method returns the config associated with the specified profile. This property returns the contents of the **entire** config file.

Return type dict

`get_auth_token()`

Return the `botocore.tokens.AuthToken` object associated with this session. If the authorization token has not yet been loaded, this will attempt to load it. If it has already been loaded, this will return the cached authorization token.

`get_available_partitions()`

Lists the available partitions found on disk

Return type list

Returns Returns a list of partition names (e.g., [“aws”, “aws-cn”])

`get_available_regions`(*service_name*, *partition_name='aws'*, *allow_non_regional=False*)

Lists the region and endpoint names of a particular partition.

Parameters

- **service_name** (*string*) – Name of a service to list endpoint for (e.g., s3). This parameter accepts a service name (e.g., “elb”) or endpoint prefix (e.g., “elasticloadbalancing”).
- **partition_name** (*string*) – Name of the partition to limit endpoints to. (e.g., aws for the public AWS endpoints, aws-cn for AWS China endpoints, aws-us-gov for AWS GovCloud (US) Endpoints, etc.
- **allow_non_regional** (*bool*) – Set to True to include endpoints that are not regional endpoints (e.g., s3-external-1, fips-us-gov-west-1, etc).

Returns Returns a list of endpoint names (e.g., [“us-east-1”]).

`get_available_services`()

Return a list of names of available services.

`get_credentials`()

Return the `botocore.credential.Credential` object associated with this session. If the credentials have not yet been loaded, this will attempt to load them. If they have already been loaded, this will return the cached credentials.

`get_data`(*data_path*)

Retrieve the data associated with *data_path*.

Parameters **data_path** (*str*) – The path to the data you wish to retrieve.

`get_default_client_config`()

Retrieves the default config for creating clients

Return type botocore.client.Config

Returns The default client config object when creating clients. If the value is `None` then there is no default config object attached to the session.

`get_partition_for_region`(*region_name*)

Lists the partition name of a particular region.

Parameters **region_name** (*string*) – Name of the region to list partition for (e.g., us-east-1).

Return type string

Returns Returns the respective partition name (e.g., aws).

`get_scoped_config`()

Returns the config values from the config file scoped to the current profile.

The configuration data is loaded **only** from the config file. It does not resolve variables based on different locations (e.g. first from the session instance, then from environment variables, then from the config file). If you want this lookup behavior, use the `get_config_variable` method instead.

Note that this configuration is specific to a single profile (the `profile` session variable).

If the `profile` session variable is set and the profile does not exist in the config file, a `ProfileNotFound` exception will be raised.

Raises ConfigNotFound, ConfigParseError, ProfileNotFound

Return type dict

get_service_data(*service_name*, *api_version=None*)

Retrieve the fully merged data associated with a service.

get_service_model(*service_name*, *api_version=None*)

Get the service model object.

Parameters

- **service_name** (*string*) – The service name
- **api_version** (*string*) – The API version of the service. If none is provided, then the latest API version will be used.

Return type L{botocore.model.ServiceModel}

Returns The botocore service model for the service.

register(*event_name*, *handler*, *unique_id=None*, *unique_id_uses_count=False*)

Register a handler with an event.

Parameters

- **event_name** (*str*) – The name of the event.
- **handler** (*callable*) – The callback to invoke when the event is emitted. This object must be callable, and must accept `**kwargs`. If either of these preconditions are not met, a `ValueError` will be raised.
- **unique_id** (*str*) – An optional identifier to associate with the registration. A unique_id can only be used once for the entire session registration (unless it is unregistered). This can be used to prevent an event handler from being registered twice.
- **unique_id_uses_count** – boolean
- **unique_id_uses_count** – Specifies if the event should maintain a count when a `unique_id` is registered and unregisted. The event can only be completely unregistered once every register call using the unique id has been matched by an `unregister` call. If `unique_id` is specified, subsequent `register` calls must use the same value for `unique_id_uses_count` as the `register` call that first registered the event.

Raises **ValueError** – If the call to `register` uses `unique_id` but the value for `unique_id_uses_count` differs from the `unique_id_uses_count` value declared by the very first `register` call for that `unique_id`.

set_config_variable(*logical_name*, *value*)

Set a configuration variable to a specific value.

By using this method, you can override the normal lookup process used in `get_config_variable` by explicitly setting a value. Subsequent calls to `get_config_variable` will use the `value`. This gives you per-session specific configuration values.

::

```
>>> # Assume logical name 'foo' maps to env var 'FOO'
>>> os.environ['FOO'] = 'myvalue'
>>> s.get_config_variable('foo')
'myvalue'
>>> s.set_config_variable('foo', 'othervalue')
>>> s.get_config_variable('foo')
'othervalue'
```

Parameters

- **logical_name** (*str*) – The logical name of the session variable you want to set. These are the keys in SESSION_VARIABLES.
- **value** – The value to associate with the config variable.

set_credentials(*access_key*, *secret_key*, *token=None*)

Manually create credentials for this session. If you would prefer to use botocore without a config file, environment variables, or IAM roles, you can pass explicit credentials into this method to establish credentials for this session.

Parameters

- **access_key** (*str*) – The access key part of the credentials.
- **secret_key** (*str*) – The secret key part of the credentials.
- **token** (*str*) – An option session token used by STS session credentials.

set_debug_logger(*logger_name='botocore'*)

Convenience function to quickly configure full debug output to go to the console.

set_default_client_config(*client_config*)

Sets the default config for creating clients

Parameters **client_config** (*botocore.client.Config*) – The default client config object when creating clients. If the value is None then there is no default config object attached to the session.

set_file_logger(*log_level*, *path*, *logger_name='botocore'*)

Convenience function to quickly configure any level of logging to a file.

Parameters

- **log_level** (*int*) – A log level as specified in the *logging* module
- **path** (*string*) – Path to the log file. The file will be created if it doesn't already exist.

set_stream_logger(*logger_name*, *log_level*, *stream=None*, *format_string=None*)

Convenience method to configure a stream logger.

Parameters

- **logger_name** (*str*) – The name of the logger to configure
- **log_level** (*str*) – The log level to set for the logger. This is any param supported by the .setLevel() method of a Log object.
- **stream** (*file*) – A file like object to log to. If none is provided then sys.stderr will be used.
- **format_string** (*str*) – The format string to use for the log formatter. If none is provided this will default to self.LOG_FORMAT.

unregister(*event_name*, *handler=None*, *unique_id=None*, *unique_id_uses_count=False*)

Unregister a handler with an event.

Parameters

- **event_name** (*str*) – The name of the event.
- **handler** (*callable*) – The callback to unregister.

- **unique_id** (*str*) – A unique identifier identifying the callback to unregister. You can provide either the handler or the unique_id, you do not have to provide both.
- **unique_id_uses_count** – boolean
- **unique_id_uses_count** – Specifies if the event should maintain a count when a `unique_id` is registered and unregisted. The event can only be completely unregistered once every `register` call using the `unique_id` has been matched by an `unregister` call. If the `unique_id` is specified, subsequent `unregister` calls must use the same value for `unique_id_uses_count` as the `register` call that first registered the event.

Raises `ValueError` – If the call to `unregister` uses `unique_id` but the value for `unique_id_uses_count` differs from the `unique_id_uses_count` value declared by the very first `register` call for that `unique_id`.

user_agent()

Return a string suitable for use as a User-Agent header. The string will be of the form:

<agent_name>/<agent_version> Python/<py_ver> <plat_name>/<plat_ver> <exec_env>

Where:

- agent_name is the value of the *user_agent_name* attribute of the session object (*Botocore* by default).
- agent_version is the value of the *user_agent_version* attribute of the session object (the botocore version by default). by default.
- py_ver is the version of the Python interpreter beng used.
- plat_name is the name of the platform (e.g. Darwin)
- plat_ver is the version of the platform
- exec_env is exec-env/$AWS_EXECUTION_ENV

If `user_agent_extra` is not empty, then this value will be appended to the end of the user agent string.

runway.aws_sso_botocore.util module

Botocore with support for AWS SSO utilities.

class runway.aws_sso_botocore.util.**SSOTokenLoader**

Bases: `object`

AWS SSO token loader.

__init__(*cache=None*)

Instantiate class.

__call__(*start_url*)

Call instance of class directly.

__new__(***kwargs*)

runway.blueprints package

Empty init for python import traversal.

Subpackages

runway.blueprints.k8s package

Empty init for python import traversal.

Submodules

runway.blueprints.k8s.k8s_iam module

Module with k8s IAM resources.

class runway.blueprints.k8s.k8s_iam.**Iam**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating k8s IAM resources.

create_template() → None

Create template (main function called by CFNgin).

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.blueprints.k8s.k8s_master module

Module with k8s cluster resources.

class runway.blueprints.k8s.k8s_master.**Cluster**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating k8s cluster resources.

create_template() → None

Create template (main function called by CFNgin).

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(**kwargs)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[runway.variables.Variable]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.blueprints.k8s.k8s_workers module

Module with k8s nodegroup.

runway.blueprints.k8s.k8s_workers.**get_valid_instance_types**() → Any

Return list of instance types from either a JSON or gzipped JSON file.

class runway.blueprints.k8s.k8s_workers.**NodeGroup**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating k8s nodegroup.

create_template() → None

Create template (main function called by CFNgin).

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of <parameter name>: <parameter attributes>.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.blueprints.staticsite package

Empty init for python import traversal.

Submodules

runway.blueprints.staticsite.auth_at_edge module

Blueprint for the Authorization@Edge implementation of a Static Site.

Described in detail in this blogpost: https://aws.amazon.com/blogs/networking-and-content-delivery/authorizationedge-how-to-use-lambdaedge-and-json-web-tokens-to-enhance-web-application-security/

class runway.blueprints.staticsite.auth_at_edge.**AuthAtEdge**

Bases: *runway.blueprints.staticsite.staticsite.StaticSite*

Auth@Edge Blueprint.

__init__(*name: str*, *context:* runway.context.CfnginContext, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *description: Optional[str] = None*) → None

Initialize the Blueprint.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **mappings** – CloudFormation Mappings to be used in the template.
- **description** – Used to describe the resulting CloudFormation template.

create_template() → None

Create the Blueprinted template for Auth@Edge.

get_auth_at_edge_lambda_and_ver(*title: str*, *description: str*, *handle: str*, *role: troposphere.iam.Role*) → Dict[str, Any]

Create a lambda function and its version.

Parameters

- **title** – The name of the function in PascalCase.
- **description** – Description to be displayed in the lambda panel.
- **handle** – The underscore separated representation of the name of the lambda. This handle is used to determine the handler for the lambda as well as identify the correct Code hook_data information.
- **role** – The Lambda Execution Role.

get_auth_at_edge_lambda(*title: str*, *description: str*, *handler: str*, *role: troposphere.iam.Role*) → troposphere.awslambda.Function

Create an Auth@Edge lambda resource.

Parameters

- **title** – The name of the function in PascalCase.
- **description** – Description to be displayed in the lambda panel.
- **handler** – The underscore separated representation of the name of the lambda. This handle is used to determine the handler for the lambda as well as identify the correct Code hook_data information.
- **role** – The Lambda Execution Role.

add_version(*title: str*, *lambda_function: troposphere.awslambda.Function*) → troposphere.awslambda.Version

Create a version association with a Lambda@Edge function.

In order to ensure different versions of the function are appropriately uploaded a hash based on the code of the lambda is appended to the name. As the code changes so will this hash value.

Parameters

- **title** – The name of the function in PascalCase.
- **lambda_function** – The Lambda function.

get_distribution_options(*bucket: troposphere.s3.Bucket*, *oai: troposphere.cloudfront.CloudFrontOriginAccessIdentity*, *lambda_funcs: List[troposphere.cloudfront.LambdaFunctionAssociation]*, *check_auth_lambda_version: troposphere.awslambda.Version*, *http_headers_lambda_version: troposphere.awslambda.Version*, *parse_auth_lambda_version: troposphere.awslambda.Version*, *refresh_auth_lambda_version: troposphere.awslambda.Version*, *sign_out_lambda_version: troposphere.awslambda.Version*) → Dict[str, Any]

Retrieve the options for our CloudFront distribution.

Keyword Arguments

- **bucket** – The bucket resource.
- **oai** – The origin access identity resource.
- **lambda_funcs** – List of Lambda Function associations.
- **check_auth_lambda_version** – Lambda Function Version to use.
- **http_headers_lambda_version** – Lambda Function Version to use.
- **parse_auth_lambda_version** – Lambda Function Version to use.
- **refresh_auth_lambda_version** – Lambda Function Version to use.
- **sign_out_lambda_version** – Lambda Function Version to use.

Returns The CloudFront Distribution Options.

__new__(***kwargs*)

property acm_certificate_specified: bool

ACM Certification specified conditional.

`add_acm_cert()` → Union[cloudfront.ViewerCertificate, Ref]

Add ACM cert.

`add_aliases()` → Union[List[str], Ref]

Add aliases.

`add_bucket()` → troposphere.s3.Bucket

Add the bucket resource along with an output of it's name / website url.

Returns The bucket resource.

`add_bucket_policy`(*bucket: troposphere.s3.Bucket*) → troposphere.s3.BucketPolicy

Add a policy to the bucket if CloudFront is disabled. Ensure PublicRead.

Parameters **bucket** – The bucket resource to place the policy.

Returns The Bucket Policy Resource.

`add_cloudfront_bucket_policy`(*bucket: troposphere.s3.Bucket*, *oai: troposphere.cloudfront.CloudFrontOriginAccessIdentity*) → troposphere.s3.BucketPolicy

Given a bucket and oai resource add cloudfront access to the bucket.

Keyword Arguments

- `bucket` – A bucket resource.
- `oai` – An Origin Access Identity resource.

Returns The CloudFront Bucket access resource.

`add_cloudfront_directory_index_rewrite`(*role: troposphere.iam.Role*) → troposphere.awslambda.Function

Add an index CloudFront directory index rewrite lambda function to the template.

Keyword Arguments `role` – The index rewrite role resource.

Returns The CloudFront directory index rewrite lambda function resource.

`add_cloudfront_directory_index_rewrite_version`(*directory_index_rewrite: troposphere.awslambda.Function*) → troposphere.awslambda.Version

Add a specific version to the directory index rewrite lambda.

Parameters **directory_index_rewrite** – The directory index rewrite lambda resource.

Returns The CloudFront directory index rewrite version.

`add_cloudfront_distribution`(*bucket_policy: troposphere.s3.BucketPolicy*, *cloudfront_distribution_options: Dict[str, Any]*) → troposphere.cloudfront.Distribution

Add the CloudFront distribution to the template / output the id and domain name.

Parameters

- **bucket_policy** – Bucket policy to allow CloudFront access.
- **cloudfront_distribution_options** – The distribution options.

Returns The CloudFront Distribution resource

add_lambda_execution_role(*name: str = 'LambdaExecutionRole'*, *function_name: str = ''*) → troposphere.iam.Role

Create the Lambda@Edge execution role.

Parameters

- **name** – Name for the Lambda Execution Role.
- **function_name** – Name of the Lambda Function the Role will be attached to.

add_logging_bucket() → Union[cloudfront.Logging, Ref]

Add Logging Bucket.

add_origin_access_identity() → troposphere.cloudfront.CloudFrontOriginAccessIdentity

Add the origin access identity resource to the template.

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

add_web_acl() → Union[str, Ref]

Add Web ACL.

property aliases_specified: bool

Aliases are specified conditional.

property cf_enabled: bool

CloudFront enabled conditional.

property cf_logging_enabled: bool

CloudFront Logging specified conditional.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

property directory_index_specified: bool

Directory Index specified conditional.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

`get_cloudfront_distribution_options`(*bucket: troposphere.s3.Bucket*, *oai: troposphere.cloudfront.CloudFrontOriginAccessIdentity*, *lambda_function_associations: List[troposphere.cloudfront.LambdaFunctionAssociation]*) → Dict[str, Any]

Retrieve the options for our CloudFront distribution.

Parameters

- **bucket** – The bucket resource
- **oai** – The origin access identity resource.
- **lambda_function_associations** – List of Lambda Function associations.

Returns The CloudFront Distribution Options.

`static get_directory_index_lambda_association`(*lambda_associations: List[troposphere.cloudfront.LambdaFunctionAssociation]*, *directory_index_rewrite_version: troposphere.awslambda.Version*) → List[troposphere.cloudfront.LambdaFunctionAssociation]

Retrieve the directory index lambda associations with the added rewriter.

Parameters

- **lambda_associations** – The lambda associations.
- **directory_index_rewrite_version** – The directory index rewrite version.

`get_lambda_associations`() → List[troposphere.cloudfront.LambdaFunctionAssociation]

Retrieve any lambda associations from the instance variables.

`get_output_definitions`() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

`get_parameter_definitions`() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

`get_parameter_values`() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of <parameter name>: <parameter attributes>.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

property role_boundary_specified: bool

IAM Role Boundary specified conditional.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *`defined_variables`*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

property version: str

Template version.

property waf_name_specified: bool

WAF name specified conditional.

runway.blueprints.staticsite.dependencies module

Module with static website supporting infrastructure.

class runway.blueprints.staticsite.dependencies.**Dependencies**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating static website buckets.

create_template() → None

Create template (main function called by CFNgin).

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.blueprints.staticsite.staticsite module

Module with static website bucket and CloudFront distribution.

class runway.blueprints.staticsite.staticsite.**StaticSite**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating S3 bucket and CloudFront distribution.

property aliases_specified: bool

Aliases are specified conditional.

property cf_enabled: bool

CloudFront enabled conditional.

property acm_certificate_specified: bool

ACM Certification specified conditional.

property cf_logging_enabled: bool

CloudFront Logging specified conditional.

property directory_index_specified: bool

Directory Index specified conditional.

property role_boundary_specified: bool

IAM Role Boundary specified conditional.

property waf_name_specified: bool

WAF name specified conditional.

create_template() → None

Create template (main function called by CFNgin).

get_lambda_associations() → List[troposphere.cloudfront.LambdaFunctionAssociation]

Retrieve any lambda associations from the instance variables.

static get_directory_index_lambda_association(*lambda_associations: List[troposphere.cloudfront.LambdaFunctionAssociation], directory_index_rewrite_version: troposphere.awslambda.Version*) → List[troposphere.cloudfront.LambdaFunctionAssociation]

Retrieve the directory index lambda associations with the added rewriter.

Parameters

- **lambda_associations** – The lambda associations.
- **directory_index_rewrite_version** – The directory index rewrite version.

get_cloudfront_distribution_options(*bucket: troposphere.s3.Bucket, oai: troposphere.cloudfront.CloudFrontOriginAccessIdentity, lambda_function_associations: List[troposphere.cloudfront.LambdaFunctionAssociation]*) → Dict[str, Any]

Retrieve the options for our CloudFront distribution.

Parameters

- **bucket** – The bucket resource
- **oai** – The origin access identity resource.
- **lambda_function_associations** – List of Lambda Function associations.

Returns The CloudFront Distribution Options.

add_aliases() → Union[List[str], Ref]

Add aliases.

add_web_acl() → Union[str, Ref]

Add Web ACL.

add_logging_bucket() → Union[cloudfront.Logging, Ref]

Add Logging Bucket.

add_acm_cert() → Union[cloudfront.ViewerCertificate, Ref]

Add ACM cert.

add_origin_access_identity() → troposphere.cloudfront.CloudFrontOriginAccessIdentity

Add the origin access identity resource to the template.

add_bucket_policy(*bucket: troposphere.s3.Bucket*) → troposphere.s3.BucketPolicy

Add a policy to the bucket if CloudFront is disabled. Ensure PublicRead.

Parameters **bucket** – The bucket resource to place the policy.

Returns The Bucket Policy Resource.

add_bucket() → troposphere.s3.Bucket

Add the bucket resource along with an output of it's name / website url.

Returns The bucket resource.

add_cloudfront_bucket_policy(*bucket: troposphere.s3.Bucket, oai: troposphere.cloudfront.CloudFrontOriginAccessIdentity*) → troposphere.s3.BucketPolicy

Given a bucket and oai resource add cloudfront access to the bucket.

Keyword Arguments

- **bucket** – A bucket resource.
- **oai** – An Origin Access Identity resource.

Returns The CloudFront Bucket access resource.

add_lambda_execution_role(*name: str = 'LambdaExecutionRole'*, *function_name: str = ''*) → troposphere.iam.Role

Create the Lambda@Edge execution role.

Parameters

- **name** – Name for the Lambda Execution Role.
- **function_name** – Name of the Lambda Function the Role will be attached to.

add_cloudfront_directory_index_rewrite(*role: troposphere.iam.Role*) → troposphere.awslambda.Function

Add an index CloudFront directory index rewrite lambda function to the template.

Keyword Arguments **role** – The index rewrite role resource.

Returns The CloudFront directory index rewrite lambda function resource.

add_cloudfront_directory_index_rewrite_version(*directory_index_rewrite: troposphere.awslambda.Function*) → troposphere.awslambda.Version

Add a specific version to the directory index rewrite lambda.

Parameters **directory_index_rewrite** – The directory index rewrite lambda resource.

Returns The CloudFront directory index rewrite version.

add_cloudfront_distribution(*bucket_policy: troposphere.s3.BucketPolicy*, *cloudfront_distribution_options: Dict[str, Any]*) → troposphere.cloudfront.Distribution

Add the CloudFront distribution to the template / output the id and domain name.

Parameters

- **bucket_policy** – Bucket policy to allow CloudFront access.
- **cloudfront_distribution_options** – The distribution options.

Returns The CloudFront Distribution resource

__init__(*name: str*, *context: runway.context.CfnginContext*, ***, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.

- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

`__new__`(***kwargs*)

`add_output`(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

`property cfn_parameters: Dict[str, Union[List[Any], str]]`

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

`property defined_variables: Dict[str, BlueprintVariableTypeDef]`

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

`get_cfn_parameters`() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

`get_output_definitions`() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

`get_parameter_definitions`() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

`get_parameter_values`() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of <parameter name>: <parameter attributes>.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

property version: str

Template version.

Submodules

runway.blueprints.tf_state module

Module with Terraform state resources.

class runway.blueprints.tf_state.**TfState**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin blueprint for creating Terraform state resources.

create_template() → None

Create template (main function called by CFNgin).

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[runway.variables.Variable]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.cfngin package

Empty init for python import traversal.

Subpackages

runway.cfngin.actions package

Import modules.

Submodules

runway.cfngin.actions.base module

CFNgin base action.

runway.cfngin.actions.base.**build_walker**(*concurrency: int*) → Callable[[...], Any]

Return a function for waling a graph.

Passed to *runway.cfngin.plan.Plan* for walking the graph.

If concurrency is 1 (no parallelism) this will return a simple topological walker that doesn't use any multithreading.

If concurrency is 0, this will return a walker that will walk the graph as fast as the graph topology allows.

If concurrency is greater than 1, it will return a walker that will only execute a maximum of concurrency steps at any given time.

Parameters **concurrency** – Number of threads to use while walking.

Returns Function to walk a *runway.cfngin.dag.DAG*.

runway.cfngin.actions.base.**stack_template_url**(*bucket_name: str*, *blueprint:* Blueprint, *endpoint: str*)

Produce an s3 url for a given blueprint.

Parameters

- **bucket_name** – The name of the S3 bucket where the resulting templates are stored.
- **blueprint** – The blueprint object to create the URL to.
- **endpoint** – The s3 endpoint used for the bucket.

class runway.cfngin.actions.base.**BaseAction**

Bases: object

Actions perform the actual work of each Command.

Each action is responsible for building the *runway.cfngin.plan.Plan* that will be executed.

DESCRIPTION

Description used when creating a plan for an action.

Type ClassVar[str]

NAME

Name of the action.

Type ClassVar[Optional[str]]

bucket_name

S3 bucket used by the action.

Type Optional[str]

bucket_region

AWS region where S3 bucket is located.

Type Optional[str]

cancel

Cancel handler.

Type threading.Event

context

The context for the current run.

Type *CfnginContext*

provider_builder

An object that will build a provider that will be interacted with in order to perform the necessary actions.

Type Optional[*ProviderBuilder*]

s3_conn

Boto3 S3 client.

Type S3Client

__init__(*context: CfnginContext*, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

property provider: ***Provider***

Return a generic provider using the default region.

Used for running things like hooks.

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

__new__(***kwargs*)

pre_run(**, dump: Union[bool, str] = False, outline: bool = False, **_BaseAction__kwargs: Any*) → None

Perform steps before running the action.

post_run(**, dump: Union[bool, str] = False, outline: bool = False, **_BaseAction__kwargs: Any*) → None

Perform steps after running the action.

run(**, concurrency: int = 0, dump: Union[bool, str] = False, force: bool = False, outline: bool = False, tail: bool = False, upload_disabled: bool = False, **_kwargs: Any*) → None

Abstract method for running the action.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

runway.cfngin.actions.deploy module

CFNgin deploy action.

runway.cfngin.actions.deploy.**build_stack_tags**(*stack:* Stack) → List[*TagTypeDef*]

Build a common set of tags to attach to a stack.

runway.cfngin.actions.deploy.**should_update**(*stack:* Stack) → bool

Test whether a stack should be submitted for updates to CloudFormation.

Parameters **stack** – The stack object to check.

runway.cfngin.actions.deploy.**should_submit**(*stack:* Stack) → bool

Test whether a stack should be submitted to CF for update/create.

Parameters **stack** – The stack object to check.

runway.cfngin.actions.deploy.**should_ensure_cfn_bucket**(*outline: bool, dump: bool*) → bool

Test whether access to the cloudformation template bucket is required.

Parameters

- **outline** – The outline action.
- **dump** – The dump action.

Returns If access to CF bucket is needed, return True.

class runway.cfngin.actions.deploy.**UsePreviousParameterValue**

Bases: object

Class used to indicate a Parameter should use it's existing value.

__init__()

__new__(***kwargs*)

runway.cfngin.actions.deploy.**handle_hooks**(*stage: Literal['post_deploy', 'pre_deploy']*, *hooks: List[CfnginHookDefinitionModel]*, *provider: Provider*, *context: CfnginContext*, ***, *dump: Union[bool, str] = False*, *outline: bool = False*) → None

Handle pre/post hooks.

Parameters

- **stage** – The name of the hook stage - pre_deploy/post_deploy.
- **hooks** – A list of dictionaries containing the hooks to execute.
- **provider** – The provider the current stack is using.
- **context** – The current CFNgin context.
- **dump** – Whether running with dump set or not.
- **outline** – Whether running with outline set or not.

class runway.cfngin.actions.deploy.**Action**

Bases: *runway.cfngin.actions.base.BaseAction*

Responsible for building & deploying CloudFormation stacks.

Generates the deploy plan based on stack dependencies (these dependencies are determined automatically based on output lookups from other stacks).

The plan can then either be printed out as an outline or executed. If executed, each stack will get launched in order which entails:

- Pushing the generated CloudFormation template to S3 if it has changed
- Submitting either a create or update of the given stack to the *runway.cfngin.providers.base.BaseProvider*.

upload_explicitly_disabled

Explicitly disable uploading rendered templates to S3.

Type bool

property upload_disabled: **bool**

Whether the CloudFormation template should be uploaded to S3.

static build_parameters(*stack: Stack*, *provider_stack: Optional[StackTypeDef] = None*) → List[ParameterTypeDef]

Build the CloudFormation Parameters for our stack.

Parameters

- **stack** – A CFNgin stack.
- **provider_stack** – An optional CFNgin provider object.

Returns The parameters for the given stack

pre_run(***, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken prior to running the action.

run(*, *concurrency: int = 0*, *dump: Union[bool, str] = False*, *force: bool = False*, *outline: bool = False*, *tail: bool = False*, *upload_disabled: bool = False*, ***_kwargs: Any*) → None

Kicks off the create/update of the stacks in the stack_definitions.

This is the main entry point for the action.

Parameters

- **concurrency** – The maximum number of concurrent deployments.
- **dump** – Dump the plan rather than execute it.
- **force** – Not used by this action.
- **outline** – Outline the plan rather than execute it.
- **tail** – Tail the stack's events.
- **upload_disabled** – Whether to explicitly disable uploading the CloudFormation template to S3.

__init__(*context: CfnginContext*, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

__new__(***kwargs*)

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken after running the action.

property provider: ***Provider***

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint: Blueprint*, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint: Blueprint*) → str

S3 URL for CloudFormation template object.

runway.cfngin.actions.destroy module

CFNgin destroy action.

class runway.cfngin.actions.destroy.**Action**

Bases: *runway.cfngin.actions.base.BaseAction*

Responsible for destroying CloudFormation stacks.

Generates a destruction plan based on stack dependencies. Stack dependencies are reversed from the deploy action. For example, if a Stack B requires Stack A during deploy, during destroy Stack A requires Stack B be destroyed first.

The plan defaults to printing an outline of what will be destroyed. If forced to execute, each stack will get destroyed in order.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Any steps that need to be taken prior to running the action.

run(*, *concurrency: int = 0*, *dump: Union[bool, str] = False*, *force: bool = False*, *outline: bool = False*, *tail: bool = False*, *upload_disabled: bool = False*, ***_kwargs: Any*) → None

Kicks off the destruction of the stacks in the stack_definitions.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Any steps that need to be taken after running the action.

__init__(*context:* CfnginContext, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

__new__(***kwargs*)

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

property provider: ***Provider***

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

runway.cfngin.actions.diff module

CFNgin diff action.

class runway.cfngin.actions.diff.**DictValue**

Bases: Generic[runway.cfngin.actions.diff._OV, runway.cfngin.actions.diff._NV]

Used to create a diff of two dictionaries.

__init__(*key: str*, *old_value: runway.cfngin.actions.diff._OV*, *new_value: runway.cfngin.actions.diff._NV*) → None

Instantiate class.

__eq__(*other: object*) → bool

Compare if self is equal to another object.

changes() → List[str]

Return changes to represent the diff between old and new value.

Returns Representation of the change (if any) between old and new value.

status() → str

Status of changes between the old value and new value.

__new__(***kwargs*)

runway.cfngin.actions.diff.**diff_dictionaries**(*old_dict: Dict[str, runway.cfngin.actions.diff._OV]*, *new_dict: Dict[str, runway.cfngin.actions.diff._NV]*) → Tuple[int, List[*runway.cfngin.actions.diff.DictValue*[runway.cfngin.actions.diff._OV, runway.cfngin.actions.diff._NV]]]

Calculate the diff two single dimension dictionaries.

Parameters

- **old_dict** – Old dictionary.
- **new_dict** – New dictionary.

Returns Number of changed records and the *DictValue* object containing the changes.

runway.cfngin.actions.diff.**format_params_diff**(*parameter_diff: List[*runway.cfngin.actions.diff.DictValue*[Any, Any]]*) → str

Handle the formatting of differences in parameters.

Parameters **parameter_diff** – A list of *DictValue* detailing the differences between two dicts returned by *diff_dictionaries()*.

Returns A formatted string that represents a parameter diff

runway.cfngin.actions.diff.**diff_parameters**(*old_params: Dict[str, runway.cfngin.actions.diff._OV]*, *new_params: Dict[str, runway.cfngin.actions.diff._NV]*) → List[*runway.cfngin.actions.diff.DictValue*[runway.cfngin.actions.diff._OV, runway.cfngin.actions.diff._NV]]

Compare the old vs. new parameters and returns a "diff".

If there are no changes, we return an empty list.

Parameters

- **old_params** – old parameters
- **new_params** – new parameters

Returns A list of differences.

class runway.cfngin.actions.diff.**Action**

Bases: *runway.cfngin.actions.deploy.Action*

Responsible for diffing CloudFormation stacks in AWS and locally.

Generates the deploy plan based on stack dependencies (these dependencies are determined automatically based on references to output values from other stacks).

The plan is then used to create a changeset for a stack using a generated template based on the current config.

run(*, *concurrency: int = 0*, *dump: Union[bool, str] = False*, *force: bool = False*, *outline: bool = False*, *tail: bool = False*, *upload_disabled: bool = False*, ***_kwargs: Any*) → None

Kicks off the diffing of the stacks in the stack_definitions.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Any steps that need to be taken prior to running the action.

Handle CFNgin bucket access denied & not existing.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Do nothing.

__init__(*context: CfnginContext*, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

__new__(***kwargs*)

static build_parameters(*stack: Stack*, *provider_stack: Optional[StackTypeDef] = None*) → List[ParameterTypeDef]

Build the CloudFormation Parameters for our stack.

Parameters

- **stack** – A CFNgin stack.
- **provider_stack** – An optional CFNgin provider object.

Returns The parameters for the given stack

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

property provider: *Provider*

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

property upload_disabled: bool

Whether the CloudFormation template should be uploaded to S3.

runway.cfngin.actions.graph module

CFNgin graph action.

runway.cfngin.actions.graph.**each_step**(*graph:* Graph) → Iterable[Tuple[*Step*, List[*Step*]]]

Yield each step and it's direct dependencies.

Parameters **graph** – Graph to iterate over.

runway.cfngin.actions.graph.**dot_format**(*out: TextIO*, *graph:* Graph, *name: str = 'digraph'*) → None

Output a graph using the graphviz "dot" format.

Parameters

- **out** – Where output will be written.
- **graph** – Graph to be output.
- **name** – Name of the graph.

runway.cfngin.actions.graph.**json_format**(*out: TextIO*, *graph:* Graph) → None

Output the graph in a machine readable JSON format.

Parameters

- **out** – Where output will be written.
- **graph** – Graph to be output.

class runway.cfngin.actions.graph.**Action**

Bases: *runway.cfngin.actions.base.BaseAction*

Responsible for outputting a graph for the current CFNgin config.

run(**, concurrency: int = 0, dump: Union[bool, str] = False, force: bool = False, outline: bool = False, tail: bool = False, upload_disabled: bool = False, **kwargs: Any*) → None

Generate the underlying graph and prints it.

__init__(*context:* CfnginContext, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

__new__(***kwargs*)

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_BaseAction__kwargs: Any*) → None

Perform steps after running the action.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_BaseAction__kwargs: Any*) → None

Perform steps before running the action.

property provider: *Provider*

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

runway.cfngin.actions.info module

CFNgin info action.

class runway.cfngin.actions.info.**Action**

Bases: *runway.cfngin.actions.base.BaseAction*

Get information on CloudFormation stacks.

Displays the outputs for the set of CloudFormation stacks.

run(**_args: Any*, ***_kwargs: Any*) → None

Get information on CloudFormation stacks.

__init__(*context:* CfnginContext, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

__new__(***kwargs*)

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_BaseAction__kwargs: Any*) → None

Perform steps after running the action.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_BaseAction__kwargs: Any*) → None

Perform steps before running the action.

property provider: *Provider*

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

runway.cfngin.actions.init module

CFNgin init action.

class runway.cfngin.actions.init.**Action**

Bases: *runway.cfngin.actions.base.BaseAction*

Initialize environment.

__init__(*context:* CfnginContext, *provider_builder: Optional[ProviderBuilder] = None*, *cancel: Optional[threading.Event] = None*)

Instantiate class.

This class creates a copy of the context object prior to initialization as some of it can perform destructive actions on the context object.

Parameters

- **context** – The context for the current run.
- **provider_builder** – An object that will build a provider that will be interacted with in order to perform the necessary actions.
- **cancel** – Cancel handler.

property cfngin_bucket: Optional[*runway.core.providers.aws.s3._bucket.Bucket*]

CFNgin bucket.

Raises *CfnginBucketRequired* – cfngin_bucket not defined.

property default_cfngin_bucket_stack:
runway.config.models.cfngin.CfnginStackDefinitionModel

CFNgin bucket stack.

run(*, *concurrency: int = 0*, *dump: Union[bool, str] = False*, *force: bool = False*, *outline: bool = False*, *tail: bool = False*, *upload_disabled: bool = True*, ***_kwargs: Any*) → None

Run the action.

Parameters

- **concurrency** – The maximum number of concurrent deployments.
- **dump** – Not used by this action
- **force** – Not used by this action.
- **outline** – Not used by this action.
- **tail** – Tail the stack's events.
- **upload_disabled** – Not used by this action.

Raises *CfnginBucketAccessDenied* – Could not head cfngin_bucket.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Do nothing.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_Action__kwargs: Any*) → None

Do nothing.

__new__(***kwargs*)

build_provider() → *Provider*

Build a CFNgin provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

property provider: *Provider*

Return a generic provider using the default region.

Used for running things like hooks.

s3_stack_push(*blueprint:* Blueprint, *force:* *bool* = *False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

runway.cfngin.blueprints package

Import modules.

Subpackages

runway.cfngin.blueprints.variables package

Import modules.

Submodules

runway.cfngin.blueprints.variables.types module

CFNgin blueprint variable types.

class runway.cfngin.blueprints.variables.types.**TroposphereType**

Bases: Generic[runway.cfngin.blueprints.variables.types.TroposphereT]

Represents a Troposphere type.

`Troposphere` will convert the value provided to the variable to the specified Troposphere type.

Both resource and parameter classes (which are just used to configure other resources) are acceptable as configuration values.

Complete resource definitions must be dictionaries, with the keys identifying the resource titles, and the values being used as the constructor parameters.

Parameter classes can be defined as dictionary or a list of dictionaries. In either case, the keys and values will be used directly as constructor parameters.

__init__(*defined_type:* *Type[runway.cfngin.blueprints.variables.types.TroposphereT]*, *, *many:* *bool* = *False*, *optional:* *bool* = *False*, *validate:* *bool* = *True*) → None

Instantiate class.

Parameters

- **defined_type** – Troposphere type.
- **many** – Whether or not multiple resources can be constructed. If the defined type is a resource, multiple resources can be passed as a dictionary of dictionaries. If it is a parameter class, multiple resources are passed as a list.
- **optional** – Whether an undefined/null configured value is acceptable. In that case a value of `None` will be passed to the template, even if `many` is enabled.

- **validate** – Whether to validate the generated object on creation. Should be left enabled unless the object will be augmented with mandatory parameters in the template code, such that it must be validated at a later point.

property resource_name: str

Name of the type or resource.

create(*value: Dict[str, Any]*) → runway.cfngin.blueprints.variables.types.TroposphereT

create(*value: List[Dict[str, Any]]*) → List[runway.cfngin.blueprints.variables.types.TroposphereT]

create(*value: None*) → None

Create the troposphere type from the value.

Parameters **value** – A dictionary or list of dictionaries (see class documentation for details) to use as parameters to create the Troposphere type instance. Each dictionary will be passed to the `from_dict` method of the type.

Returns Returns the value converted to the troposphere type.

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**CFNType**

Bases: `object`

Represents a CloudFormation Parameter Type.

CFNType can be used as the `type` for a Blueprint variable. Unlike other variables, a variable with `type: CFNType`, will be submitted to CloudFormation as a Parameter.

parameter_type

Name of the CloudFormation Parameter type to specify when submitting as a CloudFormation Parameter.

Type ClassVar[str]

See also:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**CFNString**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

A literal string.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**CFNNumber**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An integer or float.

AWS CloudFormation validates the parameter value as a number; however, when you use the parameter elsewhere in your template (for example, by using the Ref intrinsic function), the parameter value becomes a string.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**CFNNumberList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of integers or floats that are separated by commas.

AWS CloudFormation validates the parameter value as numbers; however, when you use the parameter elsewhere in your template (for example, by using the Ref intrinsic function), the parameter value becomes a list of strings.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**CFNCommaDelimitedList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of literal strings that are separated by commas.

The total number of strings should be one more than the total number of commas. Also, each member string is space trimmed.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2AvailabilityZoneName**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Availability Zone, such as us-west-2a.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2ImageId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Amazon EC2 image ID, such as ami-0ff8a91507f77f867.

Note that the AWS CloudFormation console doesn't show a drop-down list of values for this parameter type.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2InstanceId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Amazon EC2 instance ID, such as i-1e731a32.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2KeyPairKeyName**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Amazon EC2 key pair name.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SecurityGroupGroupName**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An EC2-Classic or default VPC security group name, such as my-sg-abc.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SecurityGroupId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

A security group ID, such as sg-a123fd85.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SubnetId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

A subnet ID, such as subnet-123a351e.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2VolumeId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Amazon EBS volume ID, such as vol-3cdd3f56.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2VPCId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

A VPC ID, such as vpc-a123baa3.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**Route53HostedZoneId**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An Amazon Route 53 hosted zone ID, such as Z23YXV4OVPL04A.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2AvailabilityZoneNameList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of Availability Zones for a region, such as us-west-2a, us-west-2b.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2ImageIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of Amazon EC2 image IDs, such as ami-0ff8a91507f77f867, ami-0a584ac55a7631c0c.

Note that the AWS CloudFormation console doesn't show a drop-down list of values for this parameter type.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2InstanceIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of Amazon EC2 instance IDs, such as i-1e731a32, i-1e731a34.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SecurityGroupGroupNameList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of EC2-Classic or default VPC security group names.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SecurityGroupIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of security group IDs, such as sg-a123fd85, sg-b456fd85.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2SubnetIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of subnet IDs, such as subnet-123a351e, subnet-456b351e.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2VolumeIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of Amazon EBS volume IDs, such as vol-3cdd3f56, vol-4cdd3f56.

__init__()

__new__(***kwargs*)

class runway.cfngin.blueprints.variables.types.**EC2VPCIdList**

Bases: *runway.cfngin.blueprints.variables.types.CFNType*

An array of VPC IDs, such as vpc-a123baa3, vpc-b456baa3.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.Route53HostedZoneIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

An array of Amazon Route 53 hosted zone IDs, such as Z23YXV4OVPL04A, Z23YXV4OVPL04B.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterName

Bases: runway.cfngin.blueprints.variables.types.CFNType

The name of a Systems Manager parameter key.

Use this parameter when you want to pass the parameter key. For example, you can use this type to validate that the parameter exists.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueString

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is a string.

This corresponds to the String parameter type in Parameter Store.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueStringList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is a list of strings.

This corresponds to the StringList parameter type in Parameter Store.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueCommaDelimitedList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is a list of strings.

This corresponds to the StringList parameter type in Parameter Store.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2AvailabilityZoneName

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2ImageId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2InstanceId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2KeyPairKeyName

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SecurityGroupGroupName

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SecurityGroupId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SubnetId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2VolumeId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2VPCId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueRoute53HostedZoneId

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2AvailabilityZoneNameList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2ImageIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2InstanceIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SecurityGroupGroupNameList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SecurityGroupIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2SubnetIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2VolumeIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueEC2VPCIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

class runway.cfngin.blueprints.variables.types.SSMParameterValueRoute53HostedZoneIdList

Bases: runway.cfngin.blueprints.variables.types.CFNType

A Systems Manager parameter whose value is an AWS-specific parameter type.

__init__()

__new__(**kwargs)

Submodules

runway.cfngin.blueprints.base module

CFNgin blueprint base classes.

class runway.cfngin.blueprints.base.CFNParameter

Bases: object

Wrapper around a value to indicate a CloudFormation Parameter.

__init__(*name: str*, *value: Union[bool, float, int, List[Any], str, Any]*) → None

Instantiate class.

Parameters

- **name** – The name of the CloudFormation Parameter.
- **value** – The value we're going to submit as a CloudFormation Parameter.

__repr__() → str

Object represented as a string.

property ref: troposphere.Ref

Ref the value of a parameter.

to_parameter_value() → Union[List[Any], str]

Return the value to be submitted to CloudFormation.

__new__(***kwargs*)

runway.cfngin.blueprints.base.**build_parameter**(*name: str*, *properties:* BlueprintVariableTypeDef) → Parameter

Build a troposphere Parameter with the given properties.

Parameters

- **name** – The name of the parameter.
- **properties** – Contains the properties that will be applied to the parameter. See: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html

Returns The created parameter object.

runway.cfngin.blueprints.base.**validate_variable_type**(*var_name: str*, *var_type: Union[Type[*runway.cfngin.blueprints.variables.types.CFNType*],* runway.cfngin.blueprints.variables.types.TroposphereType*[Any], type]*, *value: Any*) → Any

Ensure the value is the correct variable type.

Parameters

- **var_name** – The name of the defined variable on a blueprint.
- **var_type** – The type that the value should be.
- **value** – The object representing the value provided for the variable

Returns The appropriate value object. If the original value was of CFNType, the returned value will be wrapped in CFNParameter.

Raises **ValueError** – If the *value* isn't of *var_type* and can't be cast as that type, this is raised.

runway.cfngin.blueprints.base.**validate_allowed_values**(*allowed_values: Optional[List[Any]]*, *value: Any*) → bool

Support a variable defining which values it allows.

Parameters

- **allowed_values** – A list of allowed values from the variable definition.
- **value** – The object representing the value provided for the variable.

Returns Boolean for whether or not the value is valid.

runway.cfngin.blueprints.base.**resolve_variable**(*var_name: str*, *var_def:* BlueprintVariableTypeDef, *provided_variable: Optional[*Variable*]*, *blueprint_name: str*) → Any

Resolve a provided variable value against the variable definition.

Parameters

- **var_name** – The name of the defined variable on a blueprint.
- **var_def** – A dictionary representing the defined variables attributes.
- **provided_variable** – The variable value provided to the blueprint.
- **blueprint_name** – The name of the blueprint that the variable is being applied to.

Returns The resolved variable value, could be any python object.

Raises

- ***MissingVariable*** – Raised when a variable with no default is not provided a value.
- ***UnresolvedBlueprintVariable*** – Raised when the provided variable is not already resolved.
- **ValueError** – Raised when the value is not the right type and cannot be cast as the correct type. Raised by *runway.cfngin.blueprints.base.validate_variable_type()*
- ***ValidatorError*** – Raised when a validator raises an exception. Wraps the original exception.

runway.cfngin.blueprints.base.**parse_user_data**(*variables: Dict[str, Any]*, *raw_user_data: str*, *blueprint_name: str*) → str

Parse the given user data and renders it as a template.

It supports referencing template variables to create userdata that's supplemented with information from the stack, as commonly required when creating EC2 userdata files.

Example

Given a raw_user_data string: `'open file ${file}'` And a variables dictionary with: `{'file': 'test.txt'}` parse_user_data would output: `open file test.txt`

Parameters

- **variables** – Variables available to the template.
- **raw_user_data** – The user_data to be parsed.
- **blueprint_name** – The name of the blueprint.

Returns The parsed user data, with all the variables values and refs replaced with their resolved values.

Raises

- ***InvalidUserdataPlaceholder*** – Raised when a placeholder name in raw_user_data is not valid. E.g `${100}` would raise this.
- ***MissingVariable*** – Raised when a variable is in the raw_user_data that is not given in the blueprint

class runway.cfngin.blueprints.base.**Blueprint**

Bases: *runway.mixins.DelCachedPropMixin*

Base implementation for rendering a troposphere template.

VARIABLES

Class variable that defines the values that can be passed from CFNgin to the template. These definition include metadata used to validate the provided value and to propagate the variable to the resulting CloudFormation template.

Type ClassVar[Dict[str, *BlueprintVariableTypeDef*]]

context

CFNgin context object.

Type *CfnginContext*

description

The description of the CloudFormation template that will be generated by this Blueprint.

Type Optional[str]

mappings

CloudFormation Mappings to be added to the template during the rendering process.

Type Optional[Dict[str, Dict[str, Any]]]

name

Name of the Stack that will be created by the Blueprint.

Type str

template

Troposphere template.

Type *Template*

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added *template*.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

create_template() → None

Abstract method called to create a template from the blueprint.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of *VARIABLES* to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within *VARIABLES* or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.

- **value** – The value to put in the output.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

__new__(***kwargs*)

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[runway.variables.Variable]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

runway.cfngin.blueprints.cfngin_bucket module

CFNgin Bucket Blueprint.

class runway.cfngin.blueprints.cfngin_bucket.**CfnginBucket**

Bases: *runway.cfngin.blueprints.base.Blueprint*

CFNgin Bucket Blueprint.

property bucket: troposphere.s3.Bucket

CFNgin Bucket.

property bucket_encryption: Union[AWSHelperFn, s3.BucketEncryption]

CFNgin bucket encryption.

This cached property can be overridden in a subclass to customize the BucketEncryption property of the bucket without needing to override the bucket cached property.

property bucket_name: AWSHelperFn

CFNgin Bucket name.

property bucket_tags: troposphere.Tags

CFNgin bucket tags.

This cached property can be overridden in a subclass to customize the Tags property of the bucket without needing to override the bucket cached property.

create_template() → None

Create template.

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

property version: str

Template version.

runway.cfngin.blueprints.raw module

CFNgin blueprint representing raw template module.

runway.cfngin.blueprints.raw.**get_template_path**(*file_path: pathlib.Path*) → Optional[pathlib.Path]

Find raw template in working directory or in sys.path.

template_path from config may refer to templates co-located with the CFNgin config, or files in remote package_sources. Here, we emulate python module loading to find the path to the template.

Parameters **filename** – Template path.

Returns Path to file, or None if no file found

runway.cfngin.blueprints.raw.**resolve_variable**(*provided_variable: Optional[*Variable*]*, *blueprint_name: str*) → Any

Resolve a provided variable value against the variable definition.

This acts as a subset of resolve_variable logic in the base module, leaving out everything that doesn't apply to CFN parameters.

Parameters

- **provided_variable** – The variable value provided to the blueprint.
- **blueprint_name** – The name of the blueprint that the variable is being applied to.

Raises *UnresolvedBlueprintVariable* – Raised when the provided variable is not already resolved.

class runway.cfngin.blueprints.raw.**RawTemplateBlueprint**

Bases: *runway.cfngin.blueprints.base.Blueprint*

Blueprint class for blueprints auto-generated from raw templates.

context

CFNgin context object.

description

The description of the CloudFormation template that will be generated by this Blueprint.

mappings

CloudFormation Mappings to be added to the template during the rendering process.

name

Name of the Stack that will be created by the Blueprint.

raw_template_path

Path to the raw CloudFormation template file.

Type Path

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Any]] = None*, *raw_template_path: pathlib.Path*, ***_: Any*) → None

Instantiate class.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, Any]

Get the parameter definitions to submit to CloudFormation.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of `<parameter name>: <parameter value>`.

property rendered: str

Return (generating first if needed) rendered template.

property requires_change_set: bool

Return True if the underlying template has transforms.

property version: str

Return (generating first if needed) version hash.

to_dict() → Dict[str, Any]

Return the template as a python dictionary.

Returns the loaded template as a python dictionary

Return type dict

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Return the template in JSON.

Parameters **variables** – Unused in this subclass (variables won't affect the template).

render_template() → Tuple[str, str]

Load template and generate its md5 hash.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

create_template() → None

Abstract method called to create a template from the blueprint.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the template parameters with values from the env file, the config, and any lookups resolved. The resolution is run twice, in case the blueprint is jinja2 templated and requires provided variables to render.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises *UnresolvedBlueprintVariables* – If variables are unresolved.

runway.cfngin.blueprints.testutil module

Provides a subclass of unittest.TestCase for testing blueprints.

runway.cfngin.blueprints.testutil.**diff**(*first: str*, *second: str*) → str

Human readable differ.

class runway.cfngin.blueprints.testutil.**BlueprintTestCase**

Bases: unittest.case.TestCase

Extends the functionality of unittest.TestCase for testing blueprints.

assertRenderedBlueprint(*blueprint:* Blueprint) → None

Test that the rendered blueprint json matches the expected result.

Result files are to be stored in the repo as test/fixtures/blueprints/${blueprint.name}.json.

__init__(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

__new__(***kwargs*)

classmethod addClassCleanup(*function*, */*, **args*, ***kwargs*)

Same as addCleanup, except the cleanup items are called even if setUpClass fails (unlike tearDownClass).

addCleanup(*function*, */*, **args*, ***kwargs*)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Parameters

- **typeobj** – The data type to call this function on when both values are of the same type in assertEqual().
- **function** – The callable taking two arguments and an optional msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertCountEqual(*first*, *second*, *msg=None*)

Asserts that two iterables have the same elements, the same number of times, without regard to order.

self.assertEqual(Counter(list(first)), Counter(list(second)))

Example:

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertDictContainsSubset(*subset*, *dictionary*, *msg=None*)

Checks whether dictionary is a superset of subset.

assertEqual(*first*, *second*, *msg=None*)

Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)

Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)

Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)

Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)

Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)

Included for symmetry with assertIsNone.

assertLess(*a*, *b*, *msg=None*)

Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)

A list-specific equality assertion.

Parameters

- **list1** – The first list to compare.
- **list2** – The second list to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertLogs(*logger=None*, *level=None*)

Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. If omitted, *level* defaults to INFO and *logger* defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: *output* and *records*. At the end of the context manager, the *output* attribute will be a list of the matching formatted log messages and the *records* attribute will be a list of the corresponding LogRecord objects.

Example:

```
with self.assertLogs('foo', level='INFO') as cm:
    logging.getLogger('foo').info('first message')
    logging.getLogger('foo.bar').error('second message')
self.assertEqual(cm.output, ['INFO:foo:first message',
                             'ERROR:foo.bar:second message'])
```

assertMultiLineEqual(*first*, *second*, *msg=None*)

Assert that two multi-line strings are equal.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)

Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)

Included for symmetry with assertIsInstance.

assertNotRegex(*text*, *unexpected_regex*, *msg=None*)

Fail the test if the text matches the regular expression.

assertRaises(*expected_exception*, **args*, ***kwargs*)

Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertRaises(SomeException):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

```
with self.assertRaises(SomeException) as cm:
    do_something()
the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

assertRaisesRegex(*expected_exception*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a raised exception matches a regex.

Parameters

- **expected_exception** – Exception class expected to be raised.
- **expected_regex** – Regex (re.Pattern object or string) expected to be found in error message.
- **args** – Function to be called and extra positional args.
- **kwargs** – Extra kwargs.
- **msg** – Optional message used in case of failure. Can only be used when assertRaisesRegex is used as a context manager.

assertRegex(*text*, *expected_regex*, *msg=None*)

Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Parameters

- **seq1** – The first sequence to compare.
- **seq2** – The second sequence to compare.
- **seq_type** – The expected datatype of the sequences, or None if no datatype should be enforced.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual(*set1*, *set2*, *msg=None*)

A set-specific equality assertion.

Parameters

- **set1** – The first set to compare.
- **set2** – The second set to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)

Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)

A tuple-specific equality assertion.

Parameters

- **tuple1** – The first tuple to compare.

- **tuple2** – The second tuple to compare.
- **msg** – Optional message to use on failure instead of a list of differences.

assertWarns(*expected_warning*, **args*, ***kwargs*)

Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertWarns(SomeWarning):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion:

```
with self.assertWarns(SomeWarning) as cm:
    do_something()
the_warning = cm.warning
self.assertEqual(the_warning.some_attribute, 147)
```

assertWarnsRegex(*expected_warning*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a triggered warning matches a regexp. Basic functioning is similar to assertWarns() with the addition that only warnings whose messages also match the regular expression are considered successful matches.

Parameters

- **expected_warning** – Warning class expected to be triggered.
- **expected_regex** – Regex (re.Pattern object or string) expected to be found in error message.
- **args** – Function to be called and extra positional args.
- **kwargs** – Extra kwargs.
- **msg** – Optional message used in case of failure. Can only be used when assertWarnsRegex is used as a context manager.

debug()

Run the test without collecting errors in a TestResult

classmethod doClassCleanups()

Execute all class cleanup functions. Normally called for you after tearDownClass.

doCleanups()

Execute all cleanup functions. Normally called for you after tearDown.

fail(*msg=None*)

Fail immediately, with the given message.

failureException

alias of `AssertionError`

setUp()

Hook method for setting up the test fixture before exercising it.

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

shortDescription()

Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)

Skip this test.

subTest(*msg=<object object>*, ***params*)

Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

tearDown()

Hook method for deconstructing the test fixture after testing it.

classmethod tearDownClass()

Hook method for deconstructing the class fixture after running all tests in the class.

class runway.cfngin.blueprints.testutil.**YamlDirTestGenerator**

Bases: object

Generate blueprint tests from yaml config files.

This class creates blueprint tests from yaml files with a syntax similar to CFNgin configuration syntax. For example:

```
namespace: test
stacks:
    - name: test_sample
    class_path: blueprints.test.Sample
    variables:
        var1: value1
```

Will create a test for the specified blueprint, passing that variable as part of the test.

The test will generate a `.json` file for this blueprint, and compare it with the stored result.

By default, the generator looks for files named `test_*.yaml` in its same directory. In order to use it, subclass it in a directory containing such tests, and name the class with a pattern that will include it in nosetests' tests (for example, TestGenerator).

The subclass may override some `@property` definitions:

base_class By default, the generated tests are subclasses of *runway.cfngin.blueprints.testutil.BlueprintTestCase*. In order to change this, set this property to the desired base class.

yaml_dirs: By default, the directory where the generator is subclassed is searched for test files. Override this array for specifying more directories. These must be relative to the directory in which the subclass lives in. Globs may be used. Default: `['.']`. Example override: `['.', 'tests/*/']`

yaml_filename: By default, the generator looks for files named `test_*.yaml`. Use this to change this pattern. Globs may be used.

__init__() → None

Instantiate class.

property base_class: Type[*runway.cfngin.blueprints.testutil.BlueprintTestCase*]

Return the baseclass.

property yaml_dirs: List[str]

Yaml directories.

property yaml_filename: str

Yaml filename.

__new__(***kwargs*)

test_generator() → Iterator[*runway.cfngin.blueprints.testutil.BlueprintTestCase*]

Test generator.

runway.cfngin.blueprints.type_defs module

CFNgin Blueprint type definitions.

class runway.cfngin.blueprints.type_defs.**BlueprintVariableTypeDef**

Bases: runway.cfngin.blueprints.type_defs._RequiredBlueprintVariableTypeDef, runway.cfngin.blueprints.type_defs._OptionalBlueprintVariableTypeDef

Type definition for *runway.cfngin.blueprints.base.Blueprint.VARIABLES* items.

allowed_pattern

Only valid for variables whose type subclasses *CFNType*. A regular expression that represents the patterns you want to allow for the Cloudformation Parameter.

Type str

allowed_values

Only valid for variables whose type subclasses *CFNType*. The values that should be allowed for the CloudFormation Parameter.

Type List[Any]

constraint_description

Only valid for variables whose type subclasses *CFNType*. A string that explains the constraint when the constraint is violated for the CloudFormation Parameter.

Type str

default

The default value that should be used for the variable if none is provided in the config.

Type Any

description

A string that describes the purpose of the variable.

Type str

max_length

Only valid for variables whose type subclasses *CFNType*. The maximum length of the value for the CloudFormation Parameter.

Type int

max_value

Only valid for variables whose type subclasses *CFNType*. The max value for the CloudFormation Parameter.

Type int

min_length

Only valid for variables whose type subclasses *CFNType*. The minimum length of the value for the CloudFormation Parameter.

Type int

min_value

Only valid for variables whose type subclasses *CFNType*. The minimum value for the CloudFormation Parameter.

Type int

no_echo

Only valid for variables whose type subclasses *CFNType*. Whether to mask the parameter value whenever anyone makes a call that describes the stack. If you set the value to true, the parameter value is masked with asterisks (*).

Type bool

type

The type for the variable value. This can either be a native python type or one of the CFNgin variable types.

Type Any

validator

An optional function that can do custom validation of the variable. A validator function should take a single argument, the value being validated, and should return the value if validation is successful. If there is an issue validating the value, an exception (`ValueError`, `TypeError`, etc) should be raised by the function.

Type Callable[[Any], Any]

runway.cfngin.dag package

CFNgin directed acyclic graph (DAG) implementation.

exception runway.cfngin.dag.**DAGValidationError**

Bases: Exception

Raised when DAG validation fails.

__init__(**args*, ***kwargs*)

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class runway.cfngin.dag.**DAG**

Bases: object

Directed acyclic graph implementation.

__init__() → None

Instantiate a new DAG with no nodes or edges.

add_node(*node_name: str*) → None

Add a node if it does not exist yet, or error out.

Parameters **node_name** – The unique name of the node to add.

Raises **KeyError** – Raised if a node with the same name already exist in the graph

add_node_if_not_exists(*node_name: str*) → None

Add a node if it does not exist yet, ignoring duplicates.

Parameters **node_name** – The name of the node to add.

delete_node(*node_name: str*) → None

Delete this node and all edges referencing it.

Parameters **node_name** – The name of the node to delete.

Raises **KeyError** – Raised if the node does not exist in the graph.

delete_node_if_exists(*node_name: str*) → None

Delete this node and all edges referencing it.

Ignores any node that is not in the graph, rather than throwing an exception.

Parameters **node_name** – The name of the node to delete.

add_edge(*ind_node: str*, *dep_node: str*) → None

Add an edge (dependency) between the specified nodes.

Parameters

- **ind_node** – The independent node to add an edge to.
- **dep_node** – The dependent node that has a dependency on the ind_node.

Raises

- **KeyError** – Either the ind_node, or dep_node do not exist.
- ***DAGValidationError*** – Raised if the resulting graph is invalid.

delete_edge(*ind_node: str*, *dep_node: str*) → None

Delete an edge from the graph.

Parameters

- **ind_node** – The independent node to delete an edge from.
- **dep_node** – The dependent node that has a dependency on the ind_node.

Raises **KeyError** – Raised when the edge doesn't already exist.

transpose() → *runway.cfngin.dag.DAG*

Build a new graph with the edges reversed.

walk(*walk_func: Callable[[str], Any]*) → None

Walk each node of the graph in reverse topological order.

This can be used to perform a set of operations, where the next operation depends on the previous operation. It's important to note that walking happens serially, and is not parallelized.

Parameters **walk_func** – The function to be called on each node of the graph.

transitive_reduction() → None

Perform a transitive reduction on the DAG.

The transitive reduction of a graph is a graph with as few edges as possible with the same reachability as the original graph.

See https://en.wikipedia.org/wiki/Transitive_reduction

rename_edges(*old_node_name: str*, *new_node_name: str*) → None

Change references to a node in existing edges.

Parameters

- **old_node_name** – The old name for the node.
- **new_node_name** – The new name for the node.

predecessors(*node: str*) → List[str]

Return a list of all immediate predecessors of the given node.

Parameters **node** (*str*) – The node whose predecessors you want to find.

Returns A list of nodes that are immediate predecessors to node.

Return type List[str]

downstream(*node: str*) → List[str]

Return a list of all nodes this node has edges towards.

Parameters **node** – The node whose downstream nodes you want to find.

Returns A list of nodes that are immediately downstream from the node.

all_downstreams(*node: str*) → List[str]

Return a list of all nodes downstream in topological order.

Parameters **node** – The node whose downstream nodes you want to find.

Returns A list of nodes that are downstream from the node.

filter(*nodes: List[str]*) → *runway.cfngin.dag.DAG*

Return a new DAG with only the given nodes and their dependencies.

Parameters **nodes** – The nodes you are interested in.

all_leaves() → List[str]

Return a list of all leaves (nodes with no downstreams).

from_dict(*graph_dict: Dict[str, Union[Iterable[str], Any]]*) → None

Reset the graph and build it from the passed dictionary.

The dictionary takes the form of {node_name: [directed edges]}

Parameters **graph_dict** – The dictionary used to create the graph.

Raises **TypeError** – Raised if the value of items in the dict are not lists.

reset_graph() → None

Restore the graph to an empty state.

ind_nodes() → List[str]

Return a list of all nodes in the graph with no dependencies.

validate() → Tuple[bool, str]

Return (Boolean, message) of whether DAG is valid.

topological_sort() → List[str]

Return a topological ordering of the DAG.

Raises **ValueError** – Raised if the graph is not acyclic.

size() → int

Count of nodes in the graph.

__len__() → int

How the length of a DAG is calculated.

__new__(***kwargs*)

runway.cfngin.dag.**walk**(*dag:* runway.cfngin.dag.DAG, *walk_func: Callable[[str], Any]*) → None

Walk a DAG.

class runway.cfngin.dag.**UnlimitedSemaphore**

Bases: object

threading.Semaphore, but acquire always succeeds.

acquire(**args: Any*) → Any

Do nothing.

release() → Any

Do nothing.

__init__()

__new__(***kwargs*)

class runway.cfngin.dag.**ThreadedWalker**

Bases: object

Walk a DAG as quickly as the graph topology allows, using threads.

__init__(*semaphore: Union[threading.Semaphore,* UnlimitedSemaphore*]*)

Instantiate class.

Parameters **semaphore** – A semaphore object which can be used to control how many steps are executed in parallel.

__new__(***kwargs*)

walk(*dag:* runway.cfngin.dag.DAG, *walk_func: Callable[[str], Any]*) → None

Walk each node of the graph, in parallel if it can.

The walk_func is only called when the nodes dependencies have been satisfied.

runway.cfngin.hooks package

Import modules.

Subpackages

runway.cfngin.hooks.awslambda package

Hooks for AWS Lambda.

class runway.cfngin.hooks.awslambda.**PythonFunction**

Bases: *runway.cfngin.hooks.awslambda.base_classes.AwsLambdaHook*[*runway.cfngin.hooks.awslambda.python_requirements._project.PythonProject*]

Hook for creating an AWS Lambda Function using Python runtime.

BUILD_LAYER: ClassVar[bool] = False

Flag to denote that this hook creates a Lambda Function deployment package.

__init__(*context:* runway.context.CfnginContext, ***kwargs: Any*) → None

Instantiate class.

args: *PythonHookArgs*

Parsed hook arguments.

property deployment_package: *DeploymentPackage*[*PythonProject*]

AWS Lambda deployment package.

property project:
runway.cfngin.hooks.awslambda.python_requirements._project.PythonProject

Project being deployed as an AWS Lambda Function.

cleanup() → None

Cleanup after execution.

cleanup_on_error() → None

Cleanup after an error has occurred.

__new__(***kwargs*)

build_response(*stage: Literal['deploy', 'destroy', 'plan']*) → Optional[*BaseModel*]

Build response object that will be returned by this hook.

Parameters **stage** – The current stage being executed by the hook.

plan() → *AwsLambdaHookDeployResponseTypedDict*

Run during the **plan** stage.

post_deploy() → Any

Run during the **post_deploy** stage.

post_destroy() → Any

Run during the **post_destroy** stage.

pre_deploy() → Any

Run during the **pre_deploy** stage.

pre_destroy() → Any

Run during the **pre_destroy** stage.

ctx: *CfnginContext*

CFNgin context object.

class runway.cfngin.hooks.awslambda.**PythonLayer**

Bases: *runway.cfngin.hooks.awslambda._python_hooks.PythonFunction*

Hook for creating an AWS Lambda Layer using Python runtime.

__init__(*context:* runway.context.CfnginContext, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

build_response(*stage: Literal['deploy', 'destroy', 'plan']*) → Optional[*BaseModel*]

Build response object that will be returned by this hook.

Parameters **stage** – The current stage being executed by the hook.

cleanup() → None

Cleanup after execution.

cleanup_on_error() → None

Cleanup after an error has occurred.

property deployment_package: *DeploymentPackage*[*PythonProject*]

AWS Lambda deployment package.

plan() → *AwsLambdaHookDeployResponseTypedDict*

Run during the **plan** stage.

post_deploy() → Any

Run during the **post_deploy** stage.

post_destroy() → Any

Run during the **post_destroy** stage.

pre_deploy() → Any

Run during the **pre_deploy** stage.

pre_destroy() → Any

Run during the **pre_destroy** stage.

property project:
runway.cfngin.hooks.awslambda.python_requirements._project.PythonProject

Project being deployed as an AWS Lambda Function.

args: *PythonHookArgs*

Parsed hook arguments.

BUILD_LAYER: ClassVar[bool] = True

Flag to denote that this hook creates a Lambda Layer deployment package.

Subpackages

runway.cfngin.hooks.awslambda.models package

Data models.

Submodules

runway.cfngin.hooks.awslambda.models.args module

Argument data models.

class runway.cfngin.hooks.awslambda.models.args.**DockerOptions**

Bases: *runway.utils.BaseModel*

Docker options.

disabled: bool

Explicitly disable the use of docker (default `False`).

If not disabled and Docker is unreachable, the hook will result in an error.

Example

```
args:
  docker:
    disabled: true
```

extra_files: List[str]

List of absolute file paths within the Docker container to copy into the deployment package.

Some Python packages require extra OS libraries (`*.so`) files at runtime. These files need to be included in the deployment package for the Lambda Function to run. List the files here and the hook will handle copying them into the deployment package.

The file name may end in a wildcard (*) to accommodate `.so` files that end in an variable number (see example below).

If the file does not exist, it will result in an error.

Example

```
args:
  docker:
    extra_files:
      - /usr/lib64/mysql/libmysqlclient.so.*
      - /usr/lib64/libxmlsec1-openssl.so
```

file: Optional[pathlib.Path]

Dockerfile to use to build an image for use in this process.

This, `image` , or `runtime` must be provided. If not provided, `image` will be used.

Example

```
args:
  docker:
    file: Dockerfile
```

image: Optional[str]

Docker image to use. If the image does not exist locally, it will be pulled.

This, `file` (takes precedence), or `runtime` must be provided. If only `runtime` is provided, it will be used to determine the appropriate image to use.

Example

```
args:
  docker:
    image: public.ecr.aws/sam/build-python3.9:latest
```

name: Optional[str]

When providing a Dockerfile, this will be the name applied to the resulting image. It is the equivalent to `name` in the `name:tag` syntax of the `docker build [--tag, -t]` command option.

If not provided, a default image name is used.

This field is ignore unless `file` is provided.

Example

```
args:
  docker:
    file: Dockerfile
    name: ${namespace}.runway.awslambda
```

pull: bool

Always download updates to the specified image before use. When building an image, the `FROM` image will be updated during the build process (default `True`).

Example

```
args:
  docker:
    pull: false
```

class Config

Bases: `object`

Model configuration.

__init__()

__new__(***kwargs*)

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include

- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.awslambda.models.args.**AwsLambdaHookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Base class for AWS Lambda hook arguments.

bucket_name: str

Name of the S3 Bucket where deployment package is/will be stored. The Bucket must be in the same region the Lambda Function is being deployed in.

cache_dir: Optional[pathlib.Path]

Explicitly define the directory location. Must be an absolute path or it will be relative to the CFNgin module directory.

compatible_architectures: Optional[List[str]]

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html)

Only used by Lambda Layers.

Example

```yaml
args:
  compatible_architectures:
    - x86_64
    - arm64
```

compatible_runtimes: Optional[List[str]]

A list of compatible function runtimes. When provided, the `runtime` being used to build the deployment package must be included in the list or an error will be raised.

Only used by Lambda Layers.

Example

```yaml
args:
  compatible_runtimes:
    - python3.9
    - python3.10
```

docker: *runway.cfngin.hooks.awslambda.models.args.DockerOptions*

Docker options.

extend_gitignore: List[str]

gitignore rules that should be added to the rules already defined in a `.gitignore` file in the source code directory. This can be used with or without an existing file. Files that match a gitignore rule will not be included in the deployment package.

`.git/` & `.gitignore` will always be added.

Important: This only applies to files in the `source_code` directory.

Example

```yaml
args:
  extend_gitignore:
    - cfngin.yml
    - poetry.lock
    - poetry.toml
    - pyproject.toml
```

license: Optional[str]

The layer's software license. Can be any of the following:

- A SPDX license identifier (e.g. `Apache-2.0`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/Apache-2.0`).
- The full text of the license.

Only used by Lambda Layers.

Example

```
args:
  license: Apache-2.0
```

object_prefix: Optional[str]

Prefix to add to the S3 Object key.

The object will always be prefixed with `awslambda/functions`. If provided, the value will be added to the end of the static prefix (e.g. `awslambda/<functions|layers>/<object_prefix>/<file name>`).

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(**kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(**localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runtime: Optional[str]

Runtime of the Lambda Function (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

This, `docker.file`, or `docker.image` must be provided. If `docker.disabled`, this field is required.

When provided, the runtime available on the build system (Docker container or localhost) will be checked against this value. If they do not match, an error will be raised.

If the defined or detected runtime ever changes so that it no longer matches the deployment package in S3, the deployment package in S3 will be deleted and a new one will be built and uploaded.

slim: bool

Automatically remove information and caches from dependencies (default `True`). This is done by applying gitignore rules to the dependencies. These rules vary by language/runtime.

source_code: *pathlib.Path*

Path to the Lambda Function source code.

Example

```
args:
  source_code: ./my/package
```

use_cache: *bool*

Whether to use a cache directory with pip that will persist builds (default `True`).

class runway.cfngin.hooks.awslambda.models.args.**PythonHookArgs**

Bases: *runway.cfngin.hooks.awslambda.models.args.AwsLambdaHookArgs*

Hook arguments for a Python AWS Lambda deployment package.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

bucket_name: `str`

Name of the S3 Bucket where deployment package is/will be stored. The Bucket must be in the same region the Lambda Function is being deployed in.

cache_dir: `Optional[pathlib.Path]`

Explicitly define the directory location. Must be an absolute path or it will be relative to the CFNgin module directory.

compatible_architectures: Optional[List[str]]

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html)

Only used by Lambda Layers.

Example

```
args:
  compatible_architectures:
    - x86_64
    - arm64
```

compatible_runtimes: Optional[List[str]]

A list of compatible function runtimes. When provided, the `runtime` being used to build the deployment package must be included in the list or an error will be raised.

Only used by Lambda Layers.

Example

```
args:
  compatible_runtimes:
    - python3.9
    - python3.10
```

docker: *runway.cfngin.hooks.awslambda.models.args.DockerOptions*

Docker options.

extend_gitignore: List[str]

gitignore rules that should be added to the rules already defined in a `.gitignore` file in the source code directory. This can be used with or without an existing file. Files that match a gitignore rule will not be included in the deployment package.

`.git/` & `.gitignore` will always be added.

Important: This only applies to files in the `source_code` directory.

Example

```
args:
  extend_gitignore:
    - cfngin.yml
    - poetry.lock
    - poetry.toml
    - pyproject.toml
```

license: Optional[str]

The layer's software license. Can be any of the following:

- A SPDX license identifier (e.g. `Apache-2.0`).

- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/Apache-2.0`).
- The full text of the license.

Only used by Lambda Layers.

Example

```
args:
  license: Apache-2.0
```

object_prefix: Optional[str]

Prefix to add to the S3 Object key.

The object will always be prefixed with `awslambda/functions`. If provided, the value will be added to the end of the static prefix (e.g. `awslambda/<functions|layers>/<object_prefix>/<file name>`).

runtime: Optional[str]

Runtime of the Lambda Function (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

This, `docker.file`, or `docker.image` must be provided. If `docker.disabled`, this field is required.

When provided, the runtime available on the build system (Docker container or localhost) will be checked against this value. If they do not match, an error will be raised.

If the defined or detected runtime ever changes so that it no longer matches the deployment package in S3, the deployment package in S3 will be deleted and a new one will be built and uploaded.

source_code: pathlib.Path

Path to the Lambda Function source code.

Example

```
args:
  source_code: ./my/package
```

use_cache: bool

Whether to use a cache directory with pip that will persist builds (default `True`).

extend_pip_args: Optional[List[str]]

Additional arguments that should be passed to `pip install`.

Important: When providing this field, be careful not to duplicate any of the arguments passed by this hook (e.g. `--requirements`, `--target`, `--no-input`). Providing duplicate arguments will result in an error.

Example

```
args:
  extend_pip_args:
    - '--proxy'
    - '[user:passwd@]proxy.server:port'
```

slim: bool

Automatically remove information and caches from dependencies (default True). This is done by applying the following gitignore rules to the dependencies:

- **/*.dist-info*
- **/*.py[c|d|i|o]
- **/*.so
- **/__pycache__*

strip: bool

Whether or not to strip binary files from the dependencies (default True). This only takes effect if slim: true.

If false, the gitignore rule **/*.so is not used.

use_pipenv: bool

Whether pipenv should be used if determined appropriate.

use_poetry: bool

Whether poetry should be used if determined appropriate.

class Config

Bases: object

Model configuration.

__init__()

__new__(***kwargs*)

runway.cfngin.hooks.awslambda.models.responses module

Response data models.

class runway.cfngin.hooks.awslambda.models.responses.**AwsLambdaHookDeployResponse**

Bases: *runway.utils.BaseModel*

Data model for AwsLambdaHook deploy response.

When returned by the hook as hook_data, this model is dumped to a standard Dict using the field's aliases as the key in place of the attribute names. This is done so that the key is a direct match to a CloudFormation Property where the value should be used.

bucket_name: str

Name of the S3 Bucket where the deployment package is located. (alias S3Bucket)

code_sha256: str

SHA256 of the deployment package. This can be used by CloudFormation as the value of AWS::Lambda::Version.CodeSha256. (alias CodeSha256)

compatible_architectures: Optional[List[str]]

A list of compatible instruction set architectures. (https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html) (alias `CompatibleArchitectures`)

compatible_runtimes: Optional[List[str]]

A list of compatible function runtimes. Used for filtering with `ListLayers` and `ListLayerVersions`. (alias `CompatibleRuntimes`)

license: Optional[str]

The layer's software license (alias `License`). Can be any of the following:

- A SPDX license identifier (e.g. `MIT`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/MIT`).
- The full text of the license.

object_key: str

Key (file path) of the deployment package S3 Object. (alias `S3Key`)

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

object_version_id: Optional[str]

The version ID of the deployment package S3 Object. This will only have a value if the S3 Bucket has versioning enabled. (alias `S3ObjectVersion`)

runtime: str

Runtime of the Lambda Function. (alias `Runtime`)

class Config

Bases: object

Model configuration.

__init__()

__new__(**kwargs)

runway.cfngin.hooks.awslambda.python_requirements package

Handle python requirements.

class runway.cfngin.hooks.awslambda.python_requirements.**PythonDeploymentPackage**

Bases: *runway.cfngin.hooks.awslambda.deployment_package.DeploymentPackage*[*PythonProject*]

AWS Lambda Python Deployment Package.

property gitignore_filter: Optional[igittigitt.igittigitt.IgnoreParser]

Filter to use when zipping dependencies.

This should be overridden by subclasses if a filter should be used.

static insert_layer_dir(*file_path: pathlib.Path*, *relative_to: pathlib.Path*) → pathlib.Path

Insert python directory into local file path for layer archive.

Parameters

- **file_path** – Path to local file.
- **relative_to** – Path to a directory that the file_path will be relative to in the deployment package.

__init__(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → None

Instantiate class.

The provided .init() class method should be used in place of direct instantiation.

Parameters

- **project** – Project that is being built into a deployment package.
- **usage_type** – How the deployment package can be used by AWS Lambda.

__new__(**kwargs)

property archive_file: Path

Path to archive file.

Because the archive file path contains runtime, it's use can cause a race condition or recursion error if used in some locations. If we removed runtime from the path we would not have a way to track changes to runtime which is more important than needing to be mindful of where this is used.

property bucket: *runway.core.providers.aws.s3._bucket.Bucket*

AWS S3 bucket where deployment package will be uploaded.

build() → Path

Build the deployment package.

build_tag_set(*, *url_encoded: bool = True*) → Union[Dict[str, str], str]

Build tag set to be applied to the S3 object.

Parameters

- **layer** – Tag the deployment package as a Lambda Layer or not.
- **url_encoded** – Whether to return a dict or URL encoded query string.

property code_sha256: str

SHA256 of the archive file.

Returns Value to pass to CloudFormation `AWS::Lambda::Version.CodeSha256`.

Raises **FileNotFoundError** – Property accessed before archive file has been built.

property compatible_architectures: Optional[List[str]]

List of compatible instruction set architectures.

property compatible_runtimes: Optional[List[str]]

List of compatible runtimes.

delete() → None

Delete deployment package.

property exists: bool

Whether the deployment package exists.

classmethod init(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → *runway.cfngin.hooks.awslambda.deployment_package.DeploymentPackage*[runway.cfngin.hooks.awslambda.d

Initialize deployment package.

This should be used in place of creating an instance of this class directly as it will automatically account for the S3 object already existing.

Parameters

- **project** – Project that is being built into a deployment package.
- **usage_type** – How the deployment package can be used by AWS Lambda.

Returns Instance of generic S3 object class if S3 object exists else an instance of this class.

iterate_dependency_directory() → Iterator[Path]

Iterate over the contents of the dependency directory.

If `gitignore_filter` is set, it will be used to exclude files.

property license: Optional[str]

Software license for the project.

property md5_checksum: str

MD5 of the archive file.

Returns Value to pass as ContentMD5 when uploading to AWS S3.

Raises **FileNotFoundError** – Property accessed before archive file has been built.

property object_key: str

Key to use when upload object to AWS S3.

property object_version_id: Optional[str]

S3 object version ID.

Returns The ID of the current object version. This will only have a value if versioning is enabled on the bucket.

property runtime: str

Runtime of the deployment package.

upload(*, *build: bool = True*) → None

Upload deployment package.

Parameters **build** – If true, the deployment package will be built before before trying to upload it. If false, it must have already been built.

usage_type: Literal['function', 'layer']

How the deployment package can be used by AWS Lambda.

class runway.cfngin.hooks.awslambda.python_requirements.PythonDockerDependencyInstaller

Bases: *runway.cfngin.hooks.awslambda.docker.DockerDependencyInstaller*

Docker dependency installer for Python.

__init__(*project: PythonProject*, *, *client: Optional[DockerClient] = None*, *context: Optional[Union[CfnginContext, RunwayContext]] = None*) → None

Instantiate class.

Parameters

- **project** – awslambda project.
- **client** – Pre-configured docker.client.DockerClient.
- **context** – CFNgin or Runway context object.

property bind_mounts: List[docker.types.services.Mount]

Bind mounts that will be used by the container.

property environment_variables: Dict[str, str]

Environment variables to pass to the docker container.

This is a subset of the environment variables stored in the context object as some will cause issues if they are passed.

property install_commands: List[str]

Commands to run to install dependencies.

property python_version: Optional[runway.utils._version.Version]

Version of Python installed in the docker container.

property runtime: Optional[str]

AWS Lambda runtime determined from the docker container's Python version.

__new__(**kwargs*)

build_image(*docker_file: Path*, *, *name: Optional[str] = None*, *tag: Optional[str] = None*) → Image

Build Docker image from Dockerfile.

This method is exposed as a low-level interface. *image* should be used in place for this for most cases.

Parameters

- **docker_file** – Path to the Dockerfile to build. This path should be absolute, must exist, and must be a file.
- **name** – Name of the Docker image. The name should not contain a tag. If not provided, a default value is use.
- **tag** – Tag to apply to the image after it is built. If not provided, a default value of `latest` is used.

Returns Object representing the image that was built.

`classmethod from_project`(*project:* Project*[*AwsLambdaHookArgs*]*) → Optional[_T]

Instantiate class from a project.

High-level method that wraps instantiation in error handling.

Parameters **project** – Project being processed.

Returns Object to handle dependency installation with Docker if Docker is available and not disabled.

Raises **DockerConnectionRefused** – Docker is not install or is unreachable.

`property image: Union[docker.models.images.Image, str]`

Docker image that will be used.

Raises **ValueError** – Insufficient data to determine the desired Docker image.

`install`() → None

Install dependencies using Docker.

Commands are run as they are defined in the following cached properties:

- *pre_install_commands*
- *install_commands*
- *post_install_commands*

`log_docker_msg_bytes`(*stream: Iterator[bytes]*, *, *level: int = 20*) → List[str]

Log Docker output message from blocking generator that return bytes.

Parameters

- **stream** – Blocking generator that returns log messages as bytes.
- **level** – Log level to use when logging messages.

Returns List of log messages.

`log_docker_msg_dict`(*stream: Iterator[Dict[str, Any]]*, *, *level: int = 20*) → List[str]

Log Docker output message from blocking generator that return dict.

Parameters

- **stream** – Blocking generator that returns log messages as a dict.
- **level** – Log level to use when logging messages.

Returns list of log messages.

`property post_install_commands: List[str]`

Commands to run after dependencies have been installed.

property pre_install_commands: List[str]

Commands to run before dependencies have been installed.

pull_image(*name: str*, *, *force: bool = True*) → docker.models.images.Image

Pull a Docker image from a repository if it does not exist locally.

This method is exposed as a low-level interface. *image* should be used in place for this for most cases.

Parameters

- **name** – Name of the Docker image including tag.
- **force** – Always pull the image even if it exists locally. This will ensure that the latest version is always used.

Returns Object representing the image found locally or pulled from a repository.

run_command(*command: str*, *, *level: int = 20*) → List[str]

Execute equivalent of `docker container run`.

Parameters

- **command** – Command to be run.
- **level** – Log level to use when logging messages.

Raises *DockerExecFailedError* – Docker container returned a non-zero exit code.

Returns List of log messages.

client: DockerClient

Docker client.

ctx: Union[*CfnginContext*, *RunwayContext*]

Context object.

options: *DockerOptions*

Hook arguments specific to Docker.

class runway.cfngin.hooks.awslambda.python_requirements.PythonProject

Bases: *runway.cfngin.hooks.awslambda.base_classes.Project*[*runway.cfngin.hooks.awslambda.models.args.PythonHookArgs*]

Python project.

DEFAULT_CACHE_DIR_NAME: ClassVar[str] = 'pip_cache'

Name of the default cache directory.

property docker: Optional[*runway.cfngin.hooks.awslambda.python_requirements._docker.PythonDockerDependencyInstaller*]

Docker interface that can be used to build the project.

property metadata_files: Tuple[Path, ...]

Project metadata files.

Files are only included in return value if they exist.

property runtime: str

Runtime of the build system.

Value should be a valid Lambda Function runtime (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

property pip: *runway.dependency_managers._pip.Pip*

Pip dependency manager.

property pipenv: Optional[*runway.dependency_managers._pipenv.Pipenv*]

Pipenv dependency manager.

Returns If the project uses pipenv and pipenv is not explicitly disabled, an object for interfacing with pipenv will be returned.

Raises *PipenvNotFoundError* – pipenv is not installed or not found in PATH.

property poetry: Optional[*runway.dependency_managers._poetry.Poetry*]

Poetry dependency manager.

Returns If the project uses poetry and poetry is not explicitly disabled, an object for interfacing with poetry will be returned.

Raises PoetryNotFound – poetry is not installed or not found in PATH.

property project_type: typing_extensions.Literal[pip, pipenv, poetry]

Type of python project.

property requirements_txt: Optional[Path]

Dependency file for the project.

property supported_metadata_files: Set[str]

Names of all supported metadata files.

Returns Set of file names - not paths.

property tmp_requirements_txt: Path

Temporary requirements.txt file.

This path is only used when exporting from another format.

cleanup() → None

Cleanup temporary files after the build process has run.

__init__(*args: runway.cfngin.hooks.awslambda.base_classes._AwsLambdaHookArgsTypeVar_co*, *context:* runway.context.CfnginContext) → None

Instantiate class.

Parameters

- **args** – Parsed hook arguments.
- **context** – Context object.

__new__(***kwargs*)

property build_directory: pathlib.Path

Directory being used to build deployment package.

property cache_dir: Optional[pathlib.Path]

Directory where a dependency manager's cache data will be stored.

Returns Explicit cache directory if provided or default cache directory if it is not provided. If configured to not use cache, will always be None.

cleanup_on_error() → None

Cleanup project files when an error occurs.

This will be run before `self.cleanup()` if an error has occurred.

Hooks should call this method in an `except` block and reraise the error afterward.

property compatible_architectures: Optional[List[str]]

List of compatible instruction set architectures.

property compatible_runtimes: Optional[List[str]]

List of compatible runtimes.

Value should be valid Lambda Function runtimes (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

Raises ValueError – Defined or detected runtime is not in the list of compatible runtimes.

property dependency_directory: pathlib.Path

Directory to use as the target of `pip install --target`.

install_dependencies() → None

Install project dependencies.

property license: Optional[str]

Software license for the project.

Can be any of the following:

- A SPDX license identifier (e.g. `MIT`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/MIT`).
- The full text of the license.

property project_root: pathlib.Path

Root directory of the project.

The top-level directory containing the source code and all configuration/metadata files (e.g. pyproject.toml, package.json).

The project root can be different from the source code directory but, if they are different, the project root should contain the source code directory. If it does not, the source code directory will be always be used.

The primary use case for this property is to allow configuration files to exist outside of the source code directory. The `project_type` can and should rely on the value of this property when determining the type.

property source_code: *runway.cfngin.hooks.awslambda.source_code.SourceCode*

Project source code.

Lazy load source code object. Extends gitignore as needed.

args: _AwsLambdaHookArgsTypeVar_co

Parsed hook arguments.

ctx: *CfnginContext*

CFNgin context object.

Submodules

runway.cfngin.hooks.awslambda.base_classes module

Base classes.

class runway.cfngin.hooks.awslambda.base_classes.**Project**

Bases: Generic[runway.cfngin.hooks.awslambda.base_classes._AwsLambdaHookArgsTypeVar_co]

Project containing source code for an AWS Lambda Function.

DEFAULT_CACHE_DIR_NAME: ClassVar[str] = 'cache'

Name of the default cache directory.

__init__(*args: runway.cfngin.hooks.awslambda.base_classes._AwsLambdaHookArgsTypeVar_co*, *context:* runway.context.CfnginContext) → None

Instantiate class.

Parameters

- **args** – Parsed hook arguments.
- **context** – Context object.

args: _AwsLambdaHookArgsTypeVar_co

Parsed hook arguments.

ctx: *CfnginContext*

CFNgin context object.

property build_directory: pathlib.Path

Directory being used to build deployment package.

property cache_dir: Optional[pathlib.Path]

Directory where a dependency manager's cache data will be stored.

Returns Explicit cache directory if provided or default cache directory if it is not provided. If configured to not use cache, will always be None.

property compatible_architectures: Optional[List[str]]

List of compatible instruction set architectures.

property compatible_runtimes: Optional[List[str]]

List of compatible runtimes.

Value should be valid Lambda Function runtimes (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

Raises ValueError – Defined or detected runtime is not in the list of compatible runtimes.

property dependency_directory: pathlib.Path

Directory to use as the target of `pip install --target`.

property license: Optional[str]

Software license for the project.

Can be any of the following:

- A SPDX license identifier (e.g. `MIT`).
- The URL of a license hosted on the internet (e.g. `https://opensource.org/licenses/MIT`).

- The full text of the license.

property metadata_files: Tuple[pathlib.Path, ...]

Project metadata files (e.g. `project.json`, `pyproject.toml`).

property runtime: str

runtime of the build system.

Value should be a valid Lambda Function runtime (https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html).

This property can be overwritten by subclasses when runtime can be determined through additional means.

property source_code: *runway.cfngin.hooks.awslambda.source_code.SourceCode*

Project source code.

Lazy load source code object. Extends gitignore as needed.

property project_root: pathlib.Path

Root directory of the project.

The top-level directory containing the source code and all configuration/metadata files (e.g. pyproject.toml, package.json).

The project root can be different from the source code directory but, if they are different, the project root should contain the source code directory. If it does not, the source code directory will be always be used.

The primary use case for this property is to allow configuration files to exist outside of the source code directory. The `project_type` can and should rely on the value of this property when determining the type.

property project_type: str

Type of project (e.g. poetry, yarn).

This should be considered more of a "subtype" as the subclass should distinguish project language. The value of this property should reflect the project/dependency management tool used within the project.

The value of this property should be calculated without initializing other properties (e.g. `source_code`) except for `project_root` so that it can be used in their initialization process.

property supported_metadata_files: Set[str]

Names of all supported metadata files.

Returns Set of file names - not paths.

cleanup() → None

Cleanup project files at the end of execution.

If any cleanup is needed (e.g. removal of temporary dependency directory) it should be implimented here. Hook's should call this method in a `finally` block to ensure it is run even if the rest of the hook encountered an error.

cleanup_on_error() → None

Cleanup project files when an error occurs.

This will be run before `self.cleanup()` if an error has occurred.

Hooks should call this method in an `except` block and reraise the error afterward.

install_dependencies() → None

Install project dependencies.

Arguments/options should be read from the `args` attribute of this object instead of being passed into the method call. The method itself only exists for timing and filling in custom handling that is required for each project type.

__new__(***kwargs*)

class runway.cfngin.hooks.awslambda.base_classes.**AwsLambdaHook**

Bases: *runway.cfngin.hooks.protocols.CfnginHookProtocol*, Generic[runway.cfngin.hooks.awslambda.base_classes._ProjectTypeVar]

Base class for AWS Lambda hooks.

BUILD_LAYER: ClassVar[bool] = False

Flag to denote if the hook creates a Lambda Function or Layer deployment package.

args: *AwsLambdaHookArgs*

Parsed hook arguments.

__init__(*context:* runway.context.CfnginContext, ***_kwargs: Any*) → None

Instantiate class.

This method should be overridden by subclasses. This is required to set the value of the args attribute.

Parameters **context** – CFNgin context object (passed in by CFNgin).

ctx: *CfnginContext*

CFNgin context object.

property deployment_package: *DeploymentPackage*[_ProjectTypeVar]

AWS Lambda deployment package.

property project: runway.cfngin.hooks.awslambda.base_classes._ProjectTypeVar

Project being deployed as an AWS Lambda Function.

__new__(***kwargs*)

build_response(*stage: typing_extensions.Literal[deploy]*) → *runway.cfngin.hooks.awslambda.models.responses.AwsLambdaHookDeployResponse*

build_response(*stage: typing_extensions.Literal[destroy]*) → Optional[*BaseModel*]

build_response(*stage: typing_extensions.Literal[plan]*) → *runway.cfngin.hooks.awslambda.models.responses.AwsLambdaHookDeployResponse*

Build response object that will be returned by this hook.

Parameters **stage** – The current stage being executed by the hook.

cleanup() → None

Cleanup temporary files at the end of execution.

If any cleanup is needed (e.g. removal of temporary dependency directory) it should be implimented here. A Hook's stage methods should call this method in a `finally` block to ensure it is run even if the rest of the hook encountered an error.

Example

```
def pre_deploy(self) -> Any:
    try:
        pass  # primary logic
    except BaseException:
        self.cleanup_on_error()
        raise
    finally:
        self.cleanup()
```

cleanup_on_error() → None

Cleanup temporary files when an error occurs.

This will be run before `self.cleanup()` if an error has occurred.

A Hook's stage method should call this method in an `except` block and reraise the error afterward.

Example

```
def pre_deploy(self) -> Any:
    try:
        pass  # primary logic
    except BaseException:
        self.cleanup_on_error()
        raise
    finally:
        self.cleanup()
```

plan() → *AwsLambdaHookDeployResponseTypedDict*

Run during the **plan** stage.

post_deploy() → Any

Run during the **post_deploy** stage.

post_destroy() → Any

Run during the **post_destroy** stage.

pre_deploy() → Any

Run during the **pre_deploy** stage.

pre_destroy() → Any

Run during the **pre_destroy** stage.

runway.cfngin.hooks.awslambda.constants module

Constant values.

runway.cfngin.hooks.awslambda.constants.**AWS_SAM_BUILD_IMAGE_PREFIX = 'public.ecr.aws/sam/build-'**

Prefix for build image registries.

runway.cfngin.hooks.awslambda.constants.**DEFAULT_IMAGE_NAME = 'runway.cfngin.hooks.awslambda'**

Default name to apply to an image when building from a Dockerfile.

runway.cfngin.hooks.awslambda.constants.**DEFAULT_IMAGE_TAG = 'latest'**

Default tag to apply to an image when building from a Dockerfile.

runway.cfngin.hooks.awslambda.deployment_package module

Deployment package.

class runway.cfngin.hooks.awslambda.deployment_package.**DeploymentPackage**

Bases: *runway.mixins.DelCachedPropMixin*, Generic[runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar]

AWS Lambda Deployment Package.

When interacting with subclass of this instance, it is recommended to only call the methods defined within this parent class. This ensures compatibility with the S3 object class that can be returned.

META_TAGS: ClassVar[Dict[str, str]] = {'code_sha256': 'runway.cfngin:awslambda.code_sha256', 'compatible_architectures': 'runway.cfngin:awslambda.compatible_architectures', 'compatible_runtimes': 'runway.cfngin:awslambda.compatible_runtimes', 'license': 'runway.cfngin:awslambda.license', 'md5_checksum': 'runway.cfngin:awslambda.md5_checksum', 'runtime': 'runway.cfngin:awslambda.runtime', 'source_code.hash': 'runway.cfngin:awslambda.source_code.hash'}

Mapping of metadata to the tag-key is is stored in on the S3 object.

SIZE_EOCD: Final[Literal[22]] = 22

Size of a zip file's End of Central Directory Record (empty zip).

ZIPFILE_PERMISSION_MASK: ClassVar[int] = 33488896

Mask to retrieve unix file permissions from the external attributes property of a zipfile.ZipInfo.

__init__(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → None

Instantiate class.

The provided .init() class method should be used in place of direct instantiation.

Parameters

- **project** – Project that is being built into a deployment package.
- **usage_type** – How the deployment package can be used by AWS Lambda.

project: _ProjectTypeVar

Project that is being built into a deployment package.

usage_type: Literal['function', 'layer']

How the deployment package can be used by AWS Lambda.

property archive_file: Path

Path to archive file.

Because the archive file path contains runtime, it's use can cause a race condition or recursion error if used in some locations. If we removed runtime from the path we would not have a way to track changes to runtime which is more important than needing to be mindful of where this is used.

property bucket: *runway.core.providers.aws.s3._bucket.Bucket*

AWS S3 bucket where deployment package will be uploaded.

property code_sha256: str

SHA256 of the archive file.

Returns Value to pass to CloudFormation `AWS::Lambda::Version.CodeSha256`.

Raises FileNotFoundError – Property accessed before archive file has been built.

property compatible_architectures: Optional[List[str]]

List of compatible instruction set architectures.

property compatible_runtimes: Optional[List[str]]

List of compatible runtimes.

property exists: bool

Whether the deployment package exists.

property gitignore_filter: Optional[igittigitt.IgnoreParser]

Filter to use when zipping dependencies.

This should be overridden by subclasses if a filter should be used.

property license: Optional[str]

Software license for the project.

property md5_checksum: str

MD5 of the archive file.

Returns Value to pass as ContentMD5 when uploading to AWS S3.

Raises FileNotFoundError – Property accessed before archive file has been built.

property object_key: str

Key to use when upload object to AWS S3.

property object_version_id: Optional[str]

S3 object version ID.

Returns The ID of the current object version. This will only have a value if versioning is enabled on the bucket.

property runtime: str

Runtime of the deployment package.

build() → Path

Build the deployment package.

build_tag_set(*, *url_encoded: typing_extensions.Literal[True] = True*) → str

build_tag_set(*, *url_encoded: typing_extensions.Literal[False] = True*) → Dict[str, str]

build_tag_set(*, *url_encoded: bool = True*) → Union[Dict[str, str], str]

Build tag set to be applied to the S3 object.

Parameters

- **layer** – Tag the deployment package as a Lambda Layer or not.

- **url_encoded** – Whether to return a dict or URL encoded query string.

delete() → None

Delete deployment package.

static insert_layer_dir(*file_path: Path*, *relative_to: Path*) → Path

Insert directory into local file path for layer archive.

If required, this should be overridden by a subclass for language specific requirements.

Parameters

- **file_path** – Path to local file.
- **relative_to** – Path to a directory that the file_path will be relative to in the deployment package.

iterate_dependency_directory() → Iterator[Path]

Iterate over the contents of the dependency directory.

If `gitignore_filter` is set, it will be used to exclude files.

upload(*, *build: bool = True*) → None

Upload deployment package.

Parameters **build** – If true, the deployment package will be built before before trying to upload it. If false, it must have already been built.

classmethod init(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → *runway.cfngin.hooks.awslambda.deployment_package.DeploymentPackage*[runway.cfngin.hooks.awslambda.d

Initialize deployment package.

This should be used in place of creating an instance of this class directly as it will automatically account for the S3 object already existing.

Parameters

- **project** – Project that is being built into a deployment package.
- **usage_type** – How the deployment package can be used by AWS Lambda.

Returns Instance of generic S3 object class if S3 object exists else an instance of this class.

__new__(***kwargs*)

class runway.cfngin.hooks.awslambda.deployment_package.**DeploymentPackageS3Object**

Bases: *runway.cfngin.hooks.awslambda.deployment_package.DeploymentPackage*[runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar]

AWS Lambda Deployment Package.

This should not need to be subclassed as the interactions required should be universal.

project

Project that is being built into a deployment package.

Type _ProjectTypeVar

property code_sha256: str

SHA256 of the archive file.

Returns Value to pass to CloudFormation `AWS::Lambda::Version.CodeSha256`.

Raises **RequiredTagNotFound** – A required tag was not found.

property compatible_architectures: Optional[List[str]]

List of compatible instruction set architectures.

property compatible_runtimes: Optional[List[str]]

List of compatible runtimes.

property exists: bool

Whether the S3 object exists.

__init__(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → None

Instantiate class.

The provided `.init()` class method should be used in place of direct instantiation.

Parameters

- **project** – Project that is being built into a deployment package.
- **usage_type** – How the deployment package can be used by AWS Lambda.

__new__(***kwargs*)

property archive_file: Path

Path to archive file.

Because the archive file path contains runtime, it's use can cause a race condition or recursion error if used in some locations. If we removed runtime from the path we would not have a way to track changes to runtime which is more important than needing to be mindful of where this is used.

property bucket: *runway.core.providers.aws.s3._bucket.Bucket*

AWS S3 bucket where deployment package will be uploaded.

build_tag_set(*, *url_encoded: bool = True*) → Union[Dict[str, str], str]

Build tag set to be applied to the S3 object.

Parameters

- **layer** – Tag the deployment package as a Lambda Layer or not.
- **url_encoded** – Whether to return a dict or URL encoded query string.

property gitignore_filter: Optional[igittigitt.IgnoreParser]

Filter to use when zipping dependencies.

This should be overridden by subclasses if a filter should be used.

property head: Optional[HeadObjectOutputTypeDef]

Response from HeadObject API call.

classmethod init(*project: runway.cfngin.hooks.awslambda.deployment_package._ProjectTypeVar*, *usage_type: typing_extensions.Literal[function, layer] = 'function'*) → *runway.cfngin.hooks.awslambda.deployment_package.DeploymentPackage*[runway.cfngin.hooks.awslambda.d

Initialize deployment package.

This should be used in place of creating an instance of this class directly as it will automatically account for the S3 object already existing.

Parameters

- **project** – Project that is being built into a deployment package.

- **usage_type** – How the deployment package can be used by AWS Lambda.

Returns Instance of generic S3 object class if S3 object exists else an instance of this class.

static insert_layer_dir(*file_path: Path*, *relative_to: Path*) → Path

Insert directory into local file path for layer archive.

If required, this should be overridden by a subclass for language specific requirements.

Parameters

- **file_path** – Path to local file.
- **relative_to** – Path to a directory that the file_path will be relative to in the deployment package.

iterate_dependency_directory() → Iterator[Path]

Iterate over the contents of the dependency directory.

If `gitignore_filter` is set, it will be used to exclude files.

property object_key: str

Key to use when upload object to AWS S3.

property license: Optional[str]

Software license for the project.

property md5_checksum: str

MD5 of the archive file.

Returns Value to pass as ContentMD5 when uploading to AWS S3.

Raises *RequiredTagNotFoundError* – A required tag was not found.

property object_tags: Dict[str, str]

S3 object tags.

property object_version_id: Optional[str]

S3 object version ID.

Returns The ID of the current object version. This will only have a value if versioning is enabled on the bucket.

property runtime: str

Runtime of the deployment package.

Raises *RequiredTagNotFoundError* – A required tag was not found.

build() → Path

Build the deployment package.

The object should already exist. This method only exists as a "placeholder" to match the parent class. If the object does not already exist, and error is raised.

Raises *S3ObjectDoesNotExistError* – The S3 object does not exist.

delete() → None

Delete deployment package.

update_tags() → None

Update tags of the S3 object.

upload(*, *build: bool = True*) → None

Upload deployment package.

The object should already exist. This method only exists as a "placeholder" to match the parent class. If the object does not already exist, and error is raised.

Parameters **build** – If true, the deployment package will be built before before trying to upload it. If false, it must have already been built.

Raises *S3ObjectDoesNotExistError* – The S3 object does not exist.

runway.cfngin.hooks.awslambda.docker module

Docker logic for the awslambda hook.

class runway.cfngin.hooks.awslambda.docker.**DockerDependencyInstaller**

Bases: object

Docker dependency installer.

CACHE_DIR: ClassVar[str] = '/var/task/cache_dir'

Mount path where dependency managers can cache data.

DEPENDENCY_DIR: ClassVar[str] = '/var/task/lambda'

Mount path were dependencies will be installed to within the Docker container. Other files can be moved to this directory to be included in the deployment package.

PROJECT_DIR: ClassVar[str] = '/var/task/project'

Mount path where the project directory is available within the Docker container.

__init__(*project: Project[AwsLambdaHookArgs]*, *, *client: Optional[DockerClient] = None*, *context: Optional[Union[CfnginContext, RunwayContext]] = None*) → None

Instantiate class.

This is a low-level method that requires the user to implement error handling. It is recommended to use *from_project()* instead of instantiating this class directly.

Parameters

- **project** – awslambda project.
- **client** – Pre-configured docker.client.DockerClient.
- **context** – CFNgin or Runway context object.

client: DockerClient

Docker client.

ctx: Union[*CfnginContext*, *RunwayContext*]

Context object.

options: *DockerOptions*

Hook arguments specific to Docker.

property bind_mounts: List[docker.types.services.Mount]

Bind mounts that will be used by the container.

property environment_variables: Dict[str, str]
Environment variables to pass to the Docker container.

This is a subset of the environment variables stored in the context object as some will cause issues if they are passed.

property image: Union[docker.models.images.Image, str]
Docker image that will be used.

Raises ValueError – Insufficient data to determine the desired Docker image.

property install_commands: List[str]
Commands to run to install dependencies.

property post_install_commands: List[str]
Commands to run after dependencies have been installed.

__new__(***kwargs*)

property pre_install_commands: List[str]
Commands to run before dependencies have been installed.

property runtime: Optional[str]
AWS Lambda runtime determined from the Docker container.

build_image(*docker_file: Path*, *, *name: Optional[str] = None*, *tag: Optional[str] = None*) → Image
Build Docker image from Dockerfile.

This method is exposed as a low-level interface. *image* should be used in place for this for most cases.

Parameters

- **docker_file** – Path to the Dockerfile to build. This path should be absolute, must exist, and must be a file.
- **name** – Name of the Docker image. The name should not contain a tag. If not provided, a default value is use.
- **tag** – Tag to apply to the image after it is built. If not provided, a default value of `latest` is used.

Returns Object representing the image that was built.

log_docker_msg_bytes(*stream: Iterator[bytes]*, *, *level: int = 20*) → List[str]
Log Docker output message from blocking generator that return bytes.

Parameters

- **stream** – Blocking generator that returns log messages as bytes.
- **level** – Log level to use when logging messages.

Returns List of log messages.

log_docker_msg_dict(*stream: Iterator[Dict[str, Any]]*, *, *level: int = 20*) → List[str]
Log Docker output message from blocking generator that return dict.

Parameters

- **stream** – Blocking generator that returns log messages as a dict.
- **level** – Log level to use when logging messages.

Returns list of log messages.

install() → None

Install dependencies using Docker.

Commands are run as they are defined in the following cached properties:

- *pre_install_commands*
- *install_commands*
- *post_install_commands*

pull_image(*name: str*, *, *force: bool = True*) → docker.models.images.Image

Pull a Docker image from a repository if it does not exist locally.

This method is exposed as a low-level interface. *image* should be used in place for this for most cases.

Parameters

- **name** – Name of the Docker image including tag.
- **force** – Always pull the image even if it exists locally. This will ensure that the latest version is always used.

Returns Object representing the image found locally or pulled from a repository.

run_command(*command: str*, *, *level: int = 20*) → List[str]

Execute equivalent of `docker container run`.

Parameters

- **command** – Command to be run.
- **level** – Log level to use when logging messages.

Raises *DockerExecFailedError* – Docker container returned a non-zero exit code.

Returns List of log messages.

classmethod from_project(*project: Project[AwsLambdaHookArgs]*) → Optional[_T]

Instantiate class from a project.

High-level method that wraps instantiation in error handling.

Parameters **project** – Project being processed.

Returns Object to handle dependency installation with Docker if Docker is available and not disabled.

Raises **DockerConnectionRefused** – Docker is not install or is unreachable.

runway.cfngin.hooks.awslambda.exceptions module

Exceptions for awslambda hooks.

exception runway.cfngin.hooks.awslambda.exceptions.**DeploymentPackageEmptyError**

Bases: *runway.cfngin.exceptions.CfnginError*

Deployment package is empty.

This can be caused by an incorrect source code directory or a gitignore rule unintentionally ignoring all source code.

Any empty deployment package is determined by checking the size of the archive file. If the size is <=22 (the size a zip file End of Central Directory Record) it has no contents.

__init__(*archive_file: Path*) → None

Instantiate class.

Parameters **archive_file** – The empty archive file.

archive_file: Path

The deployment package archive file.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.hooks.awslambda.exceptions.RuntimeMismatchError

Bases: *runway.cfngin.exceptions.CfnginError*

Required runtime does not match the detected runtime.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*expected_runtime: str*, *detected_runtime: str*) → None

Instantiate class.

Parameters

- **expected_runtime** – Explicitly defined runtime that was expected.
- **detected_runtime** – Runtime detected on the build system.

detected_runtime: str

Runtime detected on the build system.

expected_runtime: str

Explicitly defined runtime that was expected.

runway.cfngin.hooks.awslambda.source_code module

Source code.

class runway.cfngin.hooks.awslambda.source_code.SourceCode

Bases: object

Source code iterable.

__init__(*root_directory: StrPath*, ***, *gitignore_filter: Optional[igittigitt.IgnoreParser] = None*, *include_files_in_hash: Optional[Sequence[Path]] = None*, *project_root: Optional[StrPath] = None*) → None

Instantiate class.

Parameters

- **root_directory** – The root directory containing the source code.
- **gitignore_filter** – Object that has been pre-populated with rules/patterns to determine if a file should be ignored.
- **include_files_in_hash** – Files that should be included in hash calculation even if they are filtered by gitignore (e.g. `poetry.lock`).

- **project_root** – Optional project root if the source code is located within a larger project. This should only be used if the contents of value of `include_files_in_hash` contains paths that exist outside of the root directory. If this is provided, it must be a parent of the root directory.

gitignore_filter: `igittigitt.igittigitt.IgnoreParser`

Filter to use when zipping dependencies. If file/folder matches the filter, it should be ignored.

root_directory: `pathlib.Path`

The root directory containing the source code.

project_root: `pathlib.Path`

Top-level directory containing the project metadata files and source code root directory. The value can be the same as `root_directory`. If it is not, it must be a parent of `root_directory`.

property md5_hash: `str`

Calculate the md5 hash of the directory contents.

This can be resource intensive depending on the size of the project.

add_filter_rule(*pattern: str*) → None

Add rule to ignore filter.

Parameters **pattern** – The gitignore pattern to add to the filter.

__new__(***kwargs*)

sorted(*, *reverse: bool = False*) → List[pathlib.Path]

Sorted list of source code files.

Parameters **reverse** – Sort the list in reverse.

Returns Sorted list of source code files excluding those that match the ignore filter.

__eq__(*other: object*) → bool

Compare if self is equal to another object.

__fspath__() → Union[str, bytes]

Return the file system path representation of the object.

__iter__() → Iterator[pathlib.Path]

Iterate over the source code files.

Yields Files that do not match the ignore filter. Order in arbitrary.

__str__() → str

Return the string representation of the object.

__truediv__(*other: StrPath*) → Path

Create a new path object from source code's root directory.

runway.cfngin.hooks.awslambda.type_defs module

Type definitions.

class runway.cfngin.hooks.awslambda.type_defs.**AwsLambdaHookDeployResponseTypedDict**

Bases: typing_extensions.TypedDict

Dict output of runway.cfngin.hooks.awslambda.models.response.AwsLambdaHookDeployResponse using aliases.

runway.cfngin.hooks.docker package

Docker hook.

class runway.cfngin.hooks.docker.**LoginArgs**

Bases: *runway.utils.BaseModel*

Args passed to the docker.login hook.

dockercfg_path: Optional[str]

Path to a non-default Docker config file.

ecr: Optional[*ElasticContainerRegistry*]

Information describing an ECR registry.

email: Optional[str]

The email for the registry account.

password: str

The plaintext password for the registry account.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

registry: Optional[str]

URI of the registry to login to.

username: str

The registry username.

runway.cfngin.hooks.docker.**login**(*, *context:* runway.context.CfnginContext, ***kwargs: Any*) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

Docker login hook.

Replicates the functionality of `docker login` cli command.

kwargs are parsed by *LoginArgs*.

Subpackages

runway.cfngin.hooks.docker.image package

Docker image actions & argument parsers.

Replicates the functionality of `docker image` CLI commands.

class runway.cfngin.hooks.docker.image.**DockerImageBuildApiOptions**

Bases: *runway.utils.BaseModel*

Options for controlling Docker.

buildargs: Dict[str, Any]

Dict of build-time variables that will be passed to Docker.

custom_context: bool

Whether to use custom context when providing a file object.

extra_hosts: Dict[str, str]

Extra hosts to add to */etc/hosts* in the build containers. Defined as a mapping of hostname to IP address.

forcerm: bool

Always remove intermediate containers, even after unsuccessful builds.

isolation: Optional[str]

Isolation technology used during build.

network_mode: Optional[str]

Network mode for the run commands during build.

nocache: bool

Whether to use cache.

platform: Optional[str]

Set platform if server is multi-platform capable. Uses format `os[/arch[/variant]]`.

pull: bool

Whether to download any updates to the FROM image in the Dockerfile.

rm: bool

Whether to remove intermediate containers.

squash: bool

Whether to squash the resulting image layers into a single layer.

tag: Optional[str]

Optional name and tag to apply to the base image when it is built.

target: Optional[str]

Name of the build-stage to build in a multi-stage Dockerfile.

timeout: Optional[int]

HTTP timeout.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises AttributeError – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

use_config_proxy: `bool`

If `True` and if the Docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being built.

class runway.cfngin.hooks.docker.image.**ImageBuildArgs**

Bases: *runway.utils.BaseModel*

Args passed to image.build.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

ecr_repo: Optional[*ElasticContainerRegistryRepository*]

AWS Elastic Container Registry repository information. Providing this will automatically construct the repo URI. If provided, do not provide `repo`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

path: DirectoryPath

Path to the directory containing the Dockerfile.

dockerfile: str

Path within the build context to the Dockerfile.

repo: Optional[str]

URI of a non Docker Hub repository where the image will be stored.

docker: *DockerImageBuildApiOptions*

Options for `docker image build`.

tags: List[str]

List of tags to apply to the image.

class runway.cfngin.hooks.docker.image.**ImagePushArgs**

Bases: *runway.utils.BaseModel*

Args passed to image.push.

ecr_repo: Optional[*ElasticContainerRegistryRepository*]

AWS Elastic Container Registry repository information. Providing this will automatically construct the repo URI. If provided, do not provide `repo`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

image: Optional[*DockerImage*]

Image to push.

repo: Optional[str]

URI of a non Docker Hub repository where the image will be stored.

tags: List[str]

List of tags to push.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.

- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.docker.image.**ImageRemoveArgs**

Bases: *runway.utils.BaseModel*

Args passed to image.remove.

ecr_repo: Optional[*ElasticContainerRegistryRepository*]

AWS Elastic Container Registry repository information. Providing this will automatically construct the repo URI. If provided, do not provide `repo`.

If using a private registry, only `repo_name` is required. If using a public registry, `repo_name` and `registry_alias`.

force: bool

Whether to force the removal of the image.

image: Optional[*DockerImage*]

Image to push.

noprune: bool

Whether to delete untagged parents.

repo: Optional[str]

URI of a non Docker Hub repository where the image will be stored.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

tags: List[str]

List of tags to remove.

runway.cfngin.hooks.docker.image.**build**(***, *context:* runway.context.CfnginContext, ***kwargs: Any*) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

Docker image build hook.

Replicates the functionality of `docker image build` CLI command.

kwargs are parsed by *ImageBuildArgs*.

runway.cfngin.hooks.docker.image.**push**(*, *context:* runway.context.CfnginContext, ***kwargs:* *Any*) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

Docker image push hook.

Replicates the functionality of `docker image push` CLI command.

kwargs are parsed by *ImagePushArgs*.

runway.cfngin.hooks.docker.image.**remove**(*, *context:* runway.context.CfnginContext, ***kwargs:* *Any*) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

Docker image push remove.

Replicates the functionality of `docker image push` CLI command.

kwargs are parsed by *ImageRemoveArgs*.

Submodules

runway.cfngin.hooks.docker.data_models module

Hook data models.

These are makeshift data models for use until Runway v2 is released and pydantic can be used.

class runway.cfngin.hooks.docker.data_models.**ElasticContainerRegistry**

Bases: *runway.utils.BaseModel*

AWS Elastic Container Registry.

account_id: Optional[str]

AWS account ID that owns the registry being logged into.

alias: Optional[str]

If it is a public repository, the alias of the repository.

public: bool

Whether the repository is public.

region: Optional[str]

AWS region where the registry is located.

property fqn: str

Fully qualified ECR name.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.docker.data_models.**DockerImage**

Bases: *runway.utils.BaseModel*

Wrapper for docker.models.images.Image.

class Config

Bases: object

Model configuration.

__init__()

__new__(***kwargs*)

property id: str

ID of the image.

property repo: str

Repository URI.

property short_id: str

ID of the image truncated to 10 characters plus the sha256: prefix.

property tags: List[str]

List of image tags.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

property uri: *runway.utils.MutableMap*

Return a mapping of tag to image URI.

class runway.cfngin.hooks.docker.data_models.**ElasticContainerRegistryRepository**

Bases: *runway.utils.BaseModel*

AWS Elastic Container Registry (ECR) Repository.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

name: str

The name of the repository.

registry: *runway.cfngin.hooks.docker.data_models.ElasticContainerRegistry*

Information about an ECR registry.

property fqn: str

Fully qualified ECR repo name.

runway.cfngin.hooks.docker.hook_data module

Docker hook_data object.

class runway.cfngin.hooks.docker.hook_data.**DockerHookData**

Bases: *runway.utils.MutableMap*

Docker hook_data object.

property client: docker.client.DockerClient

Docker client.

__bool__() → bool

Implement evaluation of instances as a bool.

__contains__(*value: Any*) → bool

Implement evaluation of 'in' conditional.

__delitem__(*key: str*) → None

Implement deletion of self[key].

Parameters **key** – Attribute name to remove from the object.

Example

__getitem__(*key: str*) → Any

Implement evaluation of self[key].

Parameters **key** – Attribute name to return the value for.

Returns The value associated with the provided key/attribute name.

Raises AttributeError – If attribute does not exist on this object.

Example

__init__(***kwargs: Any*) → None

Initialize class.

Provided `kwargs` are added to the object as attributes.

Example

__iter__() → Iterator[Any]

Return iterator object that can iterate over all attributes.

Example

__len__() → int

Implement the built-in function len().

Example

__new__(***kwargs*)

__setitem__(*key: str*, *value: Any*) → None

Implement assignment to self[key].

Parameters

- **key** – Attribute name to associate with a value.
- **value** – Value of a key/attribute.

Example

__str__() → str

Return string representation of the object.

clear() → None. Remove all items from D.

clear_found_cache() → None

Clear _found_cache.

property data: Dict[str, Any]

Sanitized output of __dict__.

Removes anything that starts with _.

find(*query: str*, *default: Optional[Any] = None*, *ignore_cache: bool = False*) → Any

Find a value in the map.

Previously found queries are cached to increase search speed. The cached value should only be used if values are not expected to change.

Parameters

- **query** – A period delimited string that is split to search for nested values
- **default** – The value to return if the query was unsuccessful.

- **ignore_cache** – Ignore cached value.

get(*key: str*, *default: Optional[Any] = None*) → Any

Implement evaluation of self.get.

Parameters

- **key** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_context(*context: CfnginContext = None*) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

update_context(*context: None = None*) → None

Update context object with new the current DockerHookData.

values() → an object providing a view on D's values

classmethod from_cfngin_context(*context:* runway.context.CfnginContext) → *runway.cfngin.hooks.docker.hook_data.DockerHookData*

Get existing object or create a new one.

runway.cfngin.hooks.ecr package

AWS Elastic Container Registry (ECR) hook.

runway.cfngin.hooks.ecr.**purge_repository**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → Dict[str, str]

Purge all images from an ECR repository.

Parameters **context** – CFNgin context object.

runway.cfngin.hooks.ssm package

AWS SSM hooks.

Submodules

runway.cfngin.hooks.ssm.parameter module

AWS SSM Parameter Store hooks.

class runway.cfngin.hooks.ssm.parameter.**ArgsDataModel**

Bases: *runway.utils.BaseModel*

Parameter hook args.

allowed_pattern

A regular expression used to validate the parameter value.

Type Optional[str]

data_type

The data type for a String parameter. Supported data types include plain text and Amazon Machine Image IDs.

Type Optional[typing_extensions.Literal[aws:ec2:image, text]]

description

Information about the parameter.

Type Optional[str]

force

Skip checking the current value of the parameter, just put it. Can be used alongside `overwrite` to always update a parameter.

Type bool

key_id

The KMS Key ID that you want to use to encrypt a parameter. Either the default AWS Key Management Service (AWS KMS) key automatically assigned to your AWS account or a custom key. Required for parameters that use the `SecureString` data type.

Type Optional[str]

name

The fully qualified name of the parameter that you want to add to the system.

Type str

overwrite

Allow overwriting an existing parameter.

Type bool

policies

One or more policies to apply to a parameter. This field takes a JSON array.

Type Optional[str]

tags

Optional metadata that you assign to a resource.

Type Optional[List[*runway.cfngin.hooks.utils.TagDataModel*]]

tier

The parameter tier to assign to a parameter.

Type typing_extensions.Literal[Advanced, Intelligent-Tiering, Standard]

type

The type of parameter.

Type typing_extensions.Literal[String, StringList, *SecureString*]

value

The parameter value that you want to add to the system. Standard parameters have a value limit of 4 KB. Advanced parameters have a value limit of 8 KB.

Type Optional[str]

class Config

Bases: `object`

Model configuration.

__init__()

__new__(***kwargs*)

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: *Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.ssm.parameter.**SecureString**

Bases: runway.cfngin.hooks.ssm.parameter._Parameter

AWS SSM Parameter Store SecureString Parameter.

__new__(**kwargs*)

property client: SSMClient

AWS SSM client.

delete() → bool

Delete parameter.

get() → ParameterTypeDef

Get parameter.

get_current_tags() → List[*TagTypeDef*]

Get Tags currently applied to Parameter.

post_deploy() → runway.cfngin.hooks.ssm.parameter._PutParameterResultTypeDef

Run during the *post_deploy* stage.

post_destroy() → bool

Run during the *post_destroy* stage.

pre_deploy() → runway.cfngin.hooks.ssm.parameter._PutParameterResultTypeDef

Run during the *pre_deploy* stage.

pre_destroy() → bool

Run during the *pre_destroy* stage.

put() → runway.cfngin.hooks.ssm.parameter._PutParameterResultTypeDef

Put parameter.

update_tags() → None

Update tags.

__init__(*context:* runway.context.CfnginContext, *, *name: str*, **kwargs: *Any*) → None

Instantiate class.

Parameters

- **context** – CFNgin context object.
- **name** – The fully qualified name of the parameter that you want to add to the system.

runway.cfngin.hooks.staticsite package

Empty init for python import traversal.

Subpackages

runway.cfngin.hooks.staticsite.auth_at_edge package

Empty init for python import traversal.

Submodules

runway.cfngin.hooks.staticsite.auth_at_edge.callback_url_retriever module

Callback URL Retriever.

Dependency pre-hook responsible for ensuring correct callback urls are retrieved or a temporary one is used in it's place.

class runway.cfngin.hooks.staticsite.auth_at_edge.callback_url_retriever.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

stack_name: str

The name of the stack to check against.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

user_pool_arn: Optional[str]

The ARN of the User Pool to check for a client.

runway.cfngin.hooks.staticsite.auth_at_edge.callback_url_retriever.**get**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → Dict[str, Any]

Retrieve the callback URLs for User Pool Client Creation.

When the User Pool is created a Callback URL is required. During a post hook entitled `client_updater` these Callback URLs are updated to that of the Distribution. Before then we need to ensure that if a Client already exists that the URLs for that client are used to prevent any interruption of service during deploy.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.auth_at_edge.client_updater module

User Pool Client Updater.

Responsible for updating the User Pool Client with the generated distribution url + callback url paths.

class runway.cfngin.hooks.staticsite.auth_at_edge.client_updater.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

alternate_domains: List[str]

A list of any alternate domains that need to be listed with the primary distribution domain.

client_id: str

Client ID.

distribution_domain: str

Distribution domain.

oauth_scopes: List[str]

A list of all available validation scopes for oauth.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

redirect_path_sign_in: `str`

The redirect path after sign in.

redirect_path_sign_out: `str`

The redirect path after sign out.

supported_identity_providers: `List[str]`

Supported identity providers.

runway.cfngin.hooks.staticsite.auth_at_edge.client_updater.**get_redirect_uris**(*domains: List[str]*, *redirect_path_sign_in: str*, *redirect_path_sign_out: str*) → Dict[str, List[str]]

Create dict of redirect URIs for AppClient.

runway.cfngin.hooks.staticsite.auth_at_edge.client_updater.**update**(*context: runway.context.CfnginContext*, **__args: Any*, ***kwargs: Any*) → bool

Update the callback urls for the User Pool Client.

Required to match the redirect_uri being sent which contains our distribution and alternate domain names.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.auth_at_edge.domain_updater module

User Pool Client Domain Updater.

class runway.cfngin.hooks.staticsite.auth_at_edge.domain_updater.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

client_id: str

The ID of the Cognito User Pool Client.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises AttributeError – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.staticsite.auth_at_edge.domain_updater.**update**(*context: runway.context.CfnginContext*, **__args: Any*, ***kwargs: Any*) → Union[Dict[str, Any], bool]

Retrieve/Update the domain name of the specified client.

A domain name is required in order to make authorization and token requests. This prehook ensures we have one available, and if not we create one based on the user pool and client ids.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.auth_at_edge.domain_updater.**delete**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → Union[Dict[str, Any], bool]

Delete the domain if the user pool was created by Runway.

If a User Pool was created by Runway, and populated with a domain, that domain must be deleted prior to the User Pool itself being deleted or an error will occur. This process ensures that our generated domain name is deleted, or skips if not able to find one.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.auth_at_edge.domain_updater.**get_user_pool_domain**(*prefix: str*, *region: str*) → str

Return a user pool domain name based on the prefix received and region.

Parameters

- **prefix** – The domain prefix for the domain.
- **region** – The region in which the pool resides.

runway.cfngin.hooks.staticsite.auth_at_edge.lambda_config module

CFNgin prehook responsible for creation of Lambda@Edge functions.

class runway.cfngin.hooks.staticsite.auth_at_edge.lambda_config.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

bucket: str

S3 bucket name.

client_id: str

The ID of the Cognito User Pool Client.

cookie_settings: Dict[str, Any]

The settings for our customized cookies.

http_headers: Dict[str, Any]

The additional headers added to our requests.

nonce_signing_secret_param_name: str

SSM param name to store nonce signing secret.

oauth_scopes: List[str]

The validation scopes for our OAuth requests.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

redirect_path_refresh: str

The URL path for authorization refresh redirect (Correlates to the refresh auth lambda).

redirect_path_sign_in: str

The URL path to be redirected to after sign in (Correlates to the parse auth lambda).

redirect_path_sign_out: str

The URL path to be redirected to after sign out (Correlates to the root to be asked to resigning).

required_group: Optional[str]

Optional User Pool group to which access should be restricted.

runway.cfngin.hooks.staticsite.auth_at_edge.lambda_config.**write**(*context: CfnginContext*, *provider: Provider*, **__args: Any*, ***kwargs: Any*) → Dict[str, Any]

Writes/Uploads the configured lambdas for Auth@Edge.

Lambda@Edge does not have the ability to allow Environment variables at the time of this writing. In order to configure our lambdas with dynamic variables we first will go through and update a "shared" template with all of the configuration elements and add that to a temporary folder along with each of the individual Lambda@Edge functions. This temporary folder is then used with the CFNgin awsLambda hook to build the functions.

runway.cfngin.hooks.staticsite.auth_at_edge.lambda_config.**get_nonce_signing_secret**(*param_name: str*, *context: runway.context.CfnginContext*) → str

Retrieve signing secret, generating & storing it first if not present.

runway.cfngin.hooks.staticsite.auth_at_edge.lambda_config.**random_key**(*length: int = 16*) → str

Generate a random key of specified length from the allowed secret characters.

Parameters **length** – The length of the random key.

runway.cfngin.hooks.staticsite.auth_at_edge.user_pool_id_retriever module

Retrieve the ID of the Cognito User Pool.

class runway.cfngin.hooks.staticsite.auth_at_edge.user_pool_id_retriever.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

created_user_pool_id: Optional[str]

The ID of the created Cognito User Pool.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

user_pool_arn: Optional[str]

The ARN of the supplied User pool.

runway.cfngin.hooks.staticsite.auth_at_edge.user_pool_id_retriever.**get**(**__args: Any*, ***kwargs: Any*) → Dict[str, Any]

Retrieve the ID of the Cognito User Pool.

The User Pool can either be supplied via an ARN or by being generated. If the user has supplied an ARN that utilize that, otherwise retrieve the generated id. Used in multiple pre_hooks for Auth@Edge.

Arguments parsed by *HookArgs*.

Submodules

runway.cfngin.hooks.staticsite.build_staticsite module

CFNgin hook for building static website.

class runway.cfngin.hooks.staticsite.build_staticsite.**HookArgsOptions**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments `options` block.

build_output: Optional[str]

Path were the build static site will be stored locally before upload.

build_steps: List[Union[str, List[str], Dict[str, Union[str, List[str]]]]]

Steps to execute to build the static site.

name: str

Static site name.

namespace: str

Namespace of the static site.

path: str

Working directory/path to the static site's source code.

pre_build_steps: List[Union[str, List[str], Dict[str, Union[str, List[str]]]]]

Steps to run before building the static site.

source_hashing:
runway.module.staticsite.options.models.RunwayStaticSiteSourceHashingDataModel

Settings for tracking the hash of the source code between runs.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.staticsite.build_staticsite.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

artifact_bucket_rxref_lookup: **str**

Query for `RxrefLookup` to get artifact bucket.

options: *runway.cfngin.hooks.staticsite.build_staticsite.HookArgsOptions*

Hook `options` block.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.staticsite.build_staticsite.**zip_and_upload**(*app_dir: str*, *bucket: str*, *key: str*, *session: Optional[boto3.session.Session] = None*) → None

Zip built static site and upload to S3.

class runway.cfngin.hooks.staticsite.build_staticsite.**OptionsArgTypeDef**

Bases: typing_extensions.TypedDict

Options argument type definition.

runway.cfngin.hooks.staticsite.build_staticsite.**build**(*context: CfnginContext*, *provider: Provider*, ***, *options: Optional[OptionsArgTypeDef] = None*, ***kwargs: Any*) → Dict[str, Any]

Build static site.

Arguments parsed by *HookArgs*.

runway.cfngin.hooks.staticsite.cleanup module

Replicated Lambda Function cleanup warning.

class runway.cfngin.hooks.staticsite.cleanup.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

stack_relative_name: **str**

Name of the CloudFormation Stack as defined in the config file (no namespace).

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.staticsite.cleanup.**get_replicated_function_names**(*outputs: List[OutputTypeDef]*) → List[str]

Extract replicated function names from CFN outputs.

runway.cfngin.hooks.staticsite.cleanup.**warn**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → bool

Notify the user of Lambda functions to delete.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.upload_staticsite module

CFNgin hook for syncing static website to S3 bucket.

class runway.cfngin.hooks.staticsite.upload_staticsite.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

bucket_name: str

S3 bucket name.

cf_disabled: bool

Disable the use of CloudFront.

distribution_domain: str

Domain of the CloudFront distribution.

distribution_id: str

CloudFront distribution ID.

distribution_path: str

Path in the CloudFront distribution to invalidate.

extra_files: List[*runway.module.staticsite.options.models.RunwayStaticSiteExtraFileDataModel*]

Extra files to sync to the S3 bucket.

website_url: Optional[str]

S3 bucket website URL.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include

- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.staticsite.upload_staticsite.**get_archives_to_prune**(*archives: List[Dict[str, Any]]*, *hook_data: Dict[str, Any]*) → List[str]

Return list of keys to delete.

Parameters

- **archives** – The full list of file archives
- **hook_data** – CFNgin hook data

runway.cfngin.hooks.staticsite.upload_staticsite.**sync**(*context: runway.context.CfnginContext*, **__args: Any*, ***kwargs: Any*) → bool

Sync static website to S3 bucket.

Arguments parsed by *HookArgs*.

Parameters **context** – The context instance.

runway.cfngin.hooks.staticsite.upload_staticsite.**update_ssm_hash**(*context: CfnginContext*, *session: Session*) → bool

Update the SSM hash with the new tracking data.

Parameters

- **context** – Context instance.

- **session** – boto3 session.

runway.cfngin.hooks.staticsite.upload_staticsite.**invalidate_distribution**(*session:* Session, *, *domain: str = 'undefined'*, *identifier: str*, *path: str = '/*'*, ***_: Any*) → bool

Invalidate the current distribution.

Parameters

- **session** – The current CFNgin session.
- **domain** – The distribution domain.
- **identifier** – The distribution id.
- **path** – The distribution path.

runway.cfngin.hooks.staticsite.upload_staticsite.**prune_archives**(*context:* CfnginContext, *session:* Session) → bool

Prune the archives from the bucket.

Parameters

- **context** – The context instance.
- **session** – The CFNgin session.

runway.cfngin.hooks.staticsite.upload_staticsite.**auto_detect_content_type**(*filename: Optional[str]*) → Optional[str]

Auto detects the content type based on the filename.

Parameters **filename** – A filename to use to auto detect the content type.

Returns The content type of the file. None if the content type could not be detected.

runway.cfngin.hooks.staticsite.upload_staticsite.**get_content_type**(*extra_file:* runway.module.staticsite.options.models.RunwaySta → Optional[str]

Return the content type of the file.

Parameters **extra_file** – The extra file configuration.

Returns The content type of the extra file. If 'content_type' is provided then that is returned, otherwise it is auto detected based on the name.

runway.cfngin.hooks.staticsite.upload_staticsite.**get_content**(*extra_file:* runway.module.staticsite.options.models.RunwayStaticSite → Optional[str]

Get serialized content based on content_type.

Parameters **extra_file** – The extra file configuration.

Returns Serialized content based on the content_type.

runway.cfngin.hooks.staticsite.upload_staticsite.**calculate_hash_of_extra_files**(*extra_files: List[*runway.module.staticsite.op → str

Return a hash of all of the given extra files.

All attributes of the extra file object are included when hashing: name, content_type, content, and file data.

Parameters **extra_files** – The list of extra file configurations.

Returns The hash of all the files.

runway.cfngin.hooks.staticsite.upload_staticsite.**get_ssm_value**(*session: Session*, *name: str*) → Optional[str]

Get the ssm parameter value.

Parameters

- **session** – The boto3 session.
- **name** – The parameter name.

Returns The parameter value.

runway.cfngin.hooks.staticsite.upload_staticsite.**set_ssm_value**(*session: Session*, *name: str*, *value: Any*, *description: str = ''*) → None

Set the ssm parameter.

Parameters

- **session** – The boto3 session.
- **name** – The name of the parameter.
- **value** – The value of the parameter.
- **description** – A description of the parameter.

runway.cfngin.hooks.staticsite.upload_staticsite.**sync_extra_files**(*context: runway.context.CfnginContext*, *bucket: str*, *extra_files: List[runway.module.staticsite.options.models.Ru*
***kwargs: Any*) → List[str]

Sync static website extra files to S3 bucket.

Parameters

- **context** – The context instance.
- **bucket** – The static site bucket name.
- **extra_files** – List of files and file content that should be uploaded.

runway.cfngin.hooks.staticsite.utils module

Utility functions for website build/upload.

runway.cfngin.hooks.staticsite.utils.**calculate_hash_of_files**(*files: Iterable[StrPath]*, *root: Path*) → str

Return a hash of all of the given files at the given root.

Parameters

- **files** – file names to include in the hash calculation, relative to `root`.

- **root** – base directory to analyze files in.

Returns A hash of the hashes of the given files.

runway.cfngin.hooks.staticsite.utils.**get_hash_of_files**(*root_path: pathlib.Path*, *directories: Optional[List[Dict[str, Union[str, List[str]]]]] = None*) → str

Generate md5 hash of files.

Parameters

- **root_path** – Base directory where all paths will be relative to. This should already be resolve to an absolute path.
- **directories** – List of mappings that describe the paths to hash and files to exclude.

runway.cfngin.hooks.staticsite.utils.**get_ignorer**(*path: pathlib.Path*, *additional_exclusions: Optional[List[str]] = None*) → igittigitt.igittigitt.IgnoreParser

Create gitignore filter from directory `.gitignore` file.

Parameters

- **path** – Top-level directory that the gitignore filter will be created for. This directory and it's subdirectories will be searched for `.gitignore` files to use.
- **additional_exclusions** – Additional gitignore patterns to add.

Submodules

runway.cfngin.hooks.acm module

CFNgin hooks for AWS Certificate Manager.

class runway.cfngin.hooks.acm.**HookArgs**

Bases: *runway.cfngin.hooks.base.HookArgsBaseModel*

Hook arguments.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.acm.**Certificate**

Bases: *runway.cfngin.hooks.base.Hook*

Hook for managing a **AWS::CertificateManager::Certificate**.

Keyword Arguments

- **alt_names** (*Optional[List[str]]*) – Additional FQDNs to be included in the Subject Alternative Name extension of the ACM certificate. For example, you can add www.example.net to a certificate for which the domain field is www.example.com if users can reach your site by using either name.
- **domain** (*str*) – The fully qualified domain name (FQDN), such as www.example.com, with which you want to secure an ACM certificate. Use an asterisk (*) to create a wildcard certificate that protects several sites in the same domain. For example, *.example.com protects www.example.com, site.example.com, and images.example.com.
- **hosted_zone_id** (*str*) – The ID of the Route 53 Hosted Zone that contains the resource record sets that you want to change. This must exist in the same account that the certificate will be created in.
- **stack_name** (*Optional[str]*) – Provide a name for the stack used to create the certificate. If not provided, the domain is used (replacing `.` with `-`).
- **ttl** (*Optional[int]*) – The resource record cache time to live (TTL), in seconds. (*default:* 300)

Example

```
pre_deploy:
  example-wildcard-cert:
    path: runway.cfngin.hooks.acm.Certificate
    required: true
    args:
      domain: '*.example.com'
      hosted_zone_id: ${xref example-com::HostedZoneId}
```

ARGS_PARSER

alias of *runway.cfngin.hooks.acm.HookArgs*

__init__(*context:* CfnginContext, *provider:* Provider, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **context** – Context instance. (passed in by CFNgin)

- **provider** – Provider instance. (passed in by CFNgin)

`domain_changed`() → bool

Check to ensure domain has not changed for existing stack.

`get_certificate`(*interval: int = 5*) → str

Get the certificate being created by a CloudFormation.

Parameters **interval** – Number of seconds to wait between attempts.

Returns Certificate ARN.

`get_validation_record`(*cert_arn: Optional[str] = None*, ***, *interval: int = 5*, *status: str = 'PENDING_VALIDATION'*) → ResourceRecordTypeDef

Get validation record from the certificate being created.

Parameters

- **cert_arn** – ARN of the certificate to validate.
- **interval** – Number of seconds to wait between attempts.
- **status** – Validation status to look for when finding a validation record. Typically only "PENDING_VALIDATION" or "SUCCESS" will be used.

Raises `ValueError` – No pending or too many pending certificates.

`put_record_set`(*record_set: ResourceRecordTypeDef*) → None

Create/update a record set on a Route 53 Hosted Zone.

Parameters **record_set** – Record set to be added to Route 53.

`remove_validation_records`(*records: Optional[List[ResourceRecordTypeDef]] = None*) → None

Remove all record set entries used to validate an ACM Certificate.

Parameters **records** – List of validation records to remove from Route 53. This can be provided in cases were the certificate has been deleted during a rollback.

`update_record_set`(*record_set: ResourceRecordTypeDef*) → None

Update a validation record set when the cert has not changed.

Parameters **record_set** – Record set to be updated in Route 53.

`deploy`(*status: Optional[Status] = None*) → Dict[str, str]

Deploy an ACM Certificate.

`__new__`(***kwargs*)

`deploy_stack`(*stack: Optional[Stack] = None*, *wait: bool = False*) → *Status*

Deploy a stack.

Parameters

- **stack** – A stack to act on.
- **wait** – Wither to wait for the stack to complete before returning.

Returns Ending status of the stack.

`destroy`(*records: Optional[List[ResourceRecordTypeDef]] = None*, *skip_r53: bool = False*) → bool

Destroy an ACM certificate.

Parameters

- **records** – List of validation records to remove from Route 53. This can be provided in cases were the certificate has been deleted during a rollback.
- **skip_r53** – Skip the removal of validation records.

destroy_stack(*stack: Optional[Stack] = None*, *wait: bool = False*) → *Status*

Destroy a stack.

Parameters

- **stack** – A stack to act on.
- **wait** – Wither to wait for the stack to complete before returning.

Returns Ending status of the stack.

generate_stack(***kwargs: Any*) → *runway.cfngin.stack.Stack*

Create a CFNgin Stack object.

get_template_description(*suffix: Optional[str] = None*) → str

Generate a template description.

Parameters **suffix** – Suffix to append to the end of a CloudFormation template description.

property tags: troposphere.Tags

Return tags that should be applied to any resource being created.

post_deploy() → Dict[str, str]

Run during the **post_deploy** stage.

post_destroy() → bool

Run during the **post_destroy** stage.

pre_deploy() → Dict[str, str]

Run during the **pre_deploy** stage.

pre_destroy() → bool

Run during the **pre_destroy** stage.

runway.cfngin.hooks.aws_lambda module

AWS Lambda hook.

runway.cfngin.hooks.aws_lambda.**copydir**(*source: str*, *destination: str*, *includes: List[str]*, *excludes: Optional[List[str]] = None*, *follow_symlinks: bool = False*) → None

Extend the functionality of shutil.

Correctly copies files and directories in a source directory.

Parameters

- **source** – Source directory.
- **destination** – Destination directory.
- **includes** – Glob patterns for files to include.
- **excludes** – Glob patterns for files to exclude.
- **follow_symlinks** – If true, symlinks will be included in the resulting zip file.

runway.cfngin.hooks.aws_lambda.**find_requirements**(*root: str*) → Optional[Dict[str, bool]]

Identify Python requirement files.

Parameters **root** – Path that should be searched for files.

Returns Name of supported requirements file and whether it was found. If none are found, `None` is returned.

runway.cfngin.hooks.aws_lambda.**should_use_docker**(*dockerize_pip: Optional[typing.Union[bool, typing_extensions.Literal[false, False, no, No, non-linux, true, True, yes, Yes]]] = None*) → bool

Assess if Docker should be used based on the value of args.

Parameters **dockerize_pip** – Value to assess if Docker should be used for pip.

runway.cfngin.hooks.aws_lambda.**str2bool**(*v: str*)

Return boolean value of string.

runway.cfngin.hooks.aws_lambda.**handle_requirements**(*package_root: str*, *dest_path: str*, *requirements: Dict[str, bool]*, *pipenv_timeout: int = 300*, *python_path: Optional[str] = None*, *use_pipenv: bool = False*) → str

Use the correct requirements file.

Parameters

- **package_root** – Base directory containing a requirements file.
- **dest_path** – Where to output the requirements file if one needs to be created.
- **requirements** – Map of requirement file names and whether they exist.
- **pipenv_timeout** – Seconds to wait for a subprocess to complete.
- **python_path** – Explicit python interpreter to be used. Requirement file generators must be installed and executable using `-m` if provided.
- **use_pipenv** – Explicitly use pipenv to export a Pipfile as requirements.txt.

Returns Path to the final requirements.txt

Raises `NotImplementedError` – When a requirements file is not found. This should never be encountered but is included just in case.

runway.cfngin.hooks.aws_lambda.**dockerized_pip**(*work_dir: str*, *client: Optional[docker.client.DockerClient] = None*, *runtime: Optional[str] = None*, *docker_file: Optional[str] = None*, *docker_image: Optional[str] = None*, *python_dontwritebytecode: bool = False*, ***_: Any*) → None

Run pip with docker.

Parameters

- **work_dir** – Work directory for docker.
- **client** – Custom docker client.
- **runtime** – Lambda runtime. Must provide one of `runtime`, `docker_file`, or `docker_image`.
- **docker_file** – Path to a Dockerfile to build an image. Must provide one of `runtime`, `docker_file`, or `docker_image`.

- **docker_image** – Local or remote docker image to use. Must provide one of `runtime`, `docker_file`, or `docker_image`.
- **python_dontwritebytecode** – Don't write bytecode.

runway.cfngin.hooks.aws_lambda.**select_bucket_region**(*custom_bucket: Optional[str]*, *hook_region: Optional[str]*, *cfngin_bucket_region: Optional[str]*, *provider_region: str*) → str

Return the appropriate region to use when uploading functions.

Select the appropriate region for the bucket where lambdas are uploaded in.

Parameters

- **custom_bucket** – The custom bucket name provided by the *bucket* kwarg of the aws_lambda hook, if provided.
- **hook_region** – The contents of the *bucket_region* argument to the hook.
- **cfngin_bucket_region** – The contents of the `cfngin_bucket_region` global setting.
- **provider_region** – The region being used by the provider.

Returns The appropriate region string.

runway.cfngin.hooks.aws_lambda.**upload_lambda_functions**(*context: CfnginContext*, *provider: Provider*, ***kwargs: Any*)

Build Lambda payloads from user configuration and upload them to S3.

Constructs ZIP archives containing files matching specified patterns for each function, uploads the result to Amazon S3, then stores objects (of type `troposphere.awslambda.Code`) in the context's hook data, ready to be referenced in blueprints.

Configuration consists of some global options, and a dictionary of function specifications. In the specifications, each key indicating the name of the function (used for generating names for artifacts), and the value determines what files to include in the ZIP (see more details below).

Payloads are uploaded to either a custom bucket or the CFNgin default bucket, with the key containing it's checksum, to allow repeated uploads to be skipped in subsequent runs.

The configuration settings are documented as keyword arguments below.

Parameters

- **provider** – Provider instance. (passed in by CFNgin)
- **context** – Context instance. (passed in by CFNgin)

Keyword Arguments

- **bucket** (*Optional[str]*) – Custom bucket to upload functions to. Omitting it will cause the default CFNgin bucket to be used.
- **bucket_region** (*Optional[str]*) – The region in which the bucket should exist. If not given, the region will be either be that of the global `cfngin_bucket_region` setting, or else the region in use by the provider.
- **prefix** (*Optional[str]*) – S3 key prefix to prepend to the uploaded zip name.
- **follow_symlinks** (*Optional[bool]*) – Will determine if symlinks should be followed and included with the zip artifact. (*default:* `False`)
- **payload_acl** (*Optional[str]*) – The canned S3 object ACL to be applied to the uploaded payload. (*default: private*)

- **functions** (*Dict[str, Any]*) – Configurations of desired payloads to build. Keys correspond to function names, used to derive key names for the payload. Each value should itself be a dictionary, with the following data:

  docker_file (Optional[str]) Path to a local DockerFile that will be built and used for `dockerize_pip`. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

  docker_image (Optional[str]) Custom Docker image to use with `dockerize_pip`. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

  dockerize_pip (Optional[Union[str, bool]]) Whether to use Docker when preparing package dependencies with pip. Can be set to True/False or the special string 'non-linux' which will only run on non Linux systems. To use this option Docker must be installed.

  exclude (Optional[Union[str, List[str]]]) Pattern or list of patterns of files to exclude from the payload. If provided, any files that match will be ignored, regardless of whether they match an inclusion pattern.

  Commonly ignored files are already excluded by default, such as `.git`, `.svn`, `__pycache__`, `*.pyc`, `.gitignore`, etc.

  include (Optional[Union[str, List[str]]]) Pattern or list of patterns of files to include in the payload. If provided, only files that match these patterns will be included in the payload.

  Omitting it is equivalent to accepting all files that are not otherwise excluded.

  path (str) Base directory of the Lambda function payload content. If it not an absolute path, it will be considered relative to the directory containing the CFNgin configuration file in use.

  Files in this directory will be added to the payload ZIP, according to the include and exclude patterns. If no patterns are provided, all files in this directory (respecting default exclusions) will be used.

  Files are stored in the archive with path names relative to this directory. So, for example, all the files contained directly under this directory will be added to the root of the ZIP file.

  pipenv_lock_timeout (Optional[int]) Time in seconds to wait while creating lock file with pipenv.

  pipenv_timeout (Optional[int]) Time in seconds to wait while running pipenv.

  python_path (Optional[str]) Absolute path to a python interpreter to use for `pip`/`pipenv` actions. If not provided, the current python interpreter will be used for `pip` and `pipenv` will be used from the current `$PATH`.

  runtime (Optional[str]) Runtime of the AWS Lambda Function being uploaded. Used with `dockerize_pip` to automatically select the appropriate Docker image to use. Must provide exactly one of `docker_file`, `docker_image`, or `runtime`.

  use_pipenv (Optional[bool]) Explicitly use Pipfile/Pipfile.lock to prepare package dependencies even if a requirements.txt file is found.

Examples

```yaml
pre_deploy:
  - path: runway.cfngin.hooks.aws_lambda.upload_lambda_functions
    required: true
    enabled: true
    data_key: lambda
    args:
      bucket: custom-bucket
      follow_symlinks: true
      prefix: cloudformation-custom-resources/
      payload_acl: authenticated-read
      functions:
        MyFunction:
          path: ./lambda_functions
          dockerize_pip: non-linux
          use_pipenv: true
          runtime: python3.9
          include:
            - '*.py'
            - '*.txt'
          exclude:
            - '*.pyc'
            - test/
```

```python
from troposphere.awslambda import Function
from runway.cfngin.blueprints.base import Blueprint

class LambdaBlueprint(Blueprint):
    def create_template(self):
        code = self.context.hook_data['lambda']['MyFunction']

        self.template.add_resource(
            Function(
                'MyFunction',
                Code=code,
                Handler='my_function.handler',
                Role='...',
                Runtime='python2.7'
            )
        )
```

runway.cfngin.hooks.base module

Base class for CFNgin hooks.

class runway.cfngin.hooks.base.**HookArgsBaseModel**

Bases: *runway.utils.BaseModel*

Base model for hook args.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data

- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.base.**Hook**

Bases: *runway.cfngin.hooks.protocols.CfnginHookProtocol*

Base class for hooks.

Not all hooks need to be classes and not all classes need to be hooks.

args

Keyword arguments passed to the hook, loaded into a MutableMap object.

Type *HookArgsBaseModel*

blueprint

Blueprint generated by the hook if it will be deploying a stack.

Type Optional[*Blueprint*]

context

Context instance. (passed in by CFNgin)

Type *CfnginContext*

provider Provider instance.

Type passed in by CFNgin

stack

Stack object if the hook deploys a stack.

Type Optional[*Stack*]

stack_name

Name of the stack created by the hook if a stack is to be created.

Type str

ARGS_PARSER

Class used to parse arguments passed to the hook.

alias of *runway.cfngin.hooks.base.HookArgsBaseModel*

__init__(*context:* CfnginContext, *provider:* Provider, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **context** – Context instance. (passed in by CFNgin)
- **provider** – Provider instance. (passed in by CFNgin)

property tags: troposphere.Tags

Return tags that should be applied to any resource being created.

generate_stack(***kwargs: Any*) → *runway.cfngin.stack.Stack*

Create a CFNgin Stack object.

get_template_description(*suffix: Optional[str] = None*) → str

Generate a template description.

Parameters **suffix** – Suffix to append to the end of a CloudFormation template description.

deploy_stack(*stack: Optional[*Stack*] = None*, *wait: bool = False*) → *Status*

Deploy a stack.

Parameters

- **stack** – A stack to act on.
- **wait** – Wither to wait for the stack to complete before returning.

Returns Ending status of the stack.

destroy_stack(*stack: Optional[*Stack*] = None*, *wait: bool = False*) → *Status*

Destroy a stack.

Parameters

- **stack** – A stack to act on.
- **wait** – Wither to wait for the stack to complete before returning.

Returns Ending status of the stack.

post_deploy() → Any

Run during the **post_deploy** stage.

post_destroy() → Any

Run during the **post_destroy** stage.

pre_deploy() → Any

Run during the **pre_deploy** stage.

pre_destroy() → Any

Run during the **pre_destroy** stage.

__new__(***kwargs*)

class runway.cfngin.hooks.base.**HookDeployAction**

Bases: *runway.cfngin.actions.deploy.Action*

Deploy action that can be used from hooks.

__init__(*context:* CfnginContext, *provider:* Provider)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider** – The provider instance.

property provider: *Provider*

Override the inherited property to return the local provider.

__new__(***kwargs*)

static build_parameters(*stack:* Stack, *provider_stack: Optional[StackTypeDef] = None*) → List[ParameterTypeDef]

Build the CloudFormation Parameters for our stack.

Parameters

- **stack** – A CFNgin stack.
- **provider_stack** – An optional CFNgin provider object.

Returns The parameters for the given stack

build_provider() → *Provider*

Override the inherited method to always return local provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken after running the action.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken prior to running the action.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

property upload_disabled: **bool**

Whether the CloudFormation template should be uploaded to S3.

run(***kwargs: Any*) → *Status*

Run the action for one stack.

class runway.cfngin.hooks.base.**HookDestroyAction**

Bases: *runway.cfngin.hooks.base.HookDeployAction*

Destroy action that can be used from hooks.

__init__(*context:* CfnginContext, *provider:* Provider)

Instantiate class.

Parameters

- **context** – The context for the current run.
- **provider** – The provider instance.

__new__(***kwargs*)

static build_parameters(*stack:* Stack, *provider_stack: Optional[StackTypeDef] = None*) → List[ParameterTypeDef]

Build the CloudFormation Parameters for our stack.

Parameters

- **stack** – A CFNgin stack.
- **provider_stack** – An optional CFNgin provider object.

Returns The parameters for the given stack

build_provider() → *Provider*

Override the inherited method to always return local provider.

ensure_cfn_bucket() → None

CloudFormation bucket where templates will be stored.

execute(***kwargs: Any*) → None

Run the action with pre and post steps.

post_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken after running the action.

pre_run(*, *dump: Union[bool, str] = False*, *outline: bool = False*, ***_: Any*) → None

Any steps that need to be taken prior to running the action.

property provider: *Provider*

Override the inherited property to return the local provider.

s3_stack_push(*blueprint:* Blueprint, *force: bool = False*) → str

Push the rendered blueprint's template to S3.

Verifies that the template doesn't already exist in S3 before pushing.

Returns URL to the template in S3.

stack_template_url(*blueprint:* Blueprint) → str

S3 URL for CloudFormation template object.

property upload_disabled: bool

Whether the CloudFormation template should be uploaded to S3.

run(***kwargs: Any*) → *Status*

Run the action for one stack.

runway.cfngin.hooks.cleanup_s3 module

CFNgin hook for cleaning up resources prior to CFN stack deletion.

class runway.cfngin.hooks.cleanup_s3.**PurgeBucketHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for `purge_bucket`.

bucket_name: `str`

Name of the bucket to purge.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.cleanup_s3.**purge_bucket**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → bool

Delete objects in bucket.

runway.cfngin.hooks.cleanup_ssm module

CFNgin hook for cleaning up resources prior to CFN stack deletion.

class runway.cfngin.hooks.cleanup_ssm.**DeleteParamHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for `delete_param`.

parameter_name: `str`

Name of the bucket to purge.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.cleanup_ssm.**delete_param**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → bool

Delete SSM parameter.

runway.cfngin.hooks.command module

Command hook.

class runway.cfngin.hooks.command.**RunCommandHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for run_command.

capture: bool

If enabled, capture the command's stdout and stderr, and return them in the hook result.

command: Union[str, List[str]]

Command(s) to run.

env: Optional[Dict[str, str]]

Dictionary of environment variable overrides for the command context. Will be merged with the current environment.

ignore_status: bool

Don't fail the hook if the command returns a non-zero status.

interactive: bool

If enabled, allow the command to interact with stdin. Otherwise, stdin will be set to the null device.

quiet: bool

Redirect the command's stdout and stderr to the null device, silencing all output. Should not be enabled if `capture` is also enabled.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

stdin: Optional[str]

String to send to the stdin of the command. Implicitly disables `interactive`.

class runway.cfngin.hooks.command.**RunCommandResponseTypeDef**

Bases: `typing_extensions.TypedDict`

Response from run_command.

runway.cfngin.hooks.command.**run_command**(**__args: Any*, ***kwargs: Any*) → *runway.cfngin.hooks.command.RunCommandResponseTypeDef*

Run a custom command as a hook.

Arguments not parsed by the data model will be forwarded to the `subprocess.Popen` function. Interesting ones include: `cwd` and `shell`.

Examples

```yaml
pre_deploy:
  command_copy_environment:
    path: runway.cfngin.hooks.command.run_command
    required: true
    enabled: true
    data_key: copy_env
    args:
      command: ['cp', 'environment.template', 'environment']
  command_git_rev_parse:
    path: runway.cfngin.hooks.command.run_command
    required: true
    enabled: true
    data_key: get_git_commit
    args:
      command: ['git', 'rev-parse', 'HEAD']
      cwd: ./my-git-repo
      capture: true
  command_npm_install:
    path: runway.cfngin.hooks.command.run_command
    args:
      command: '`cd $PROJECT_DIR/project; npm install`'
      env:
        PROJECT_DIR: ./my-project
        shell: true
```

runway.cfngin.hooks.ecs module

AWS ECS hook.

class runway.cfngin.hooks.ecs.**CreateClustersHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for `create_clusters`.

clusters: List[str]

List of cluster names to create.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.ecs.**CreateClustersResponseTypeDef**

Bases: typing_extensions.TypedDict

Response from create_clusters.

runway.cfngin.hooks.ecs.**create_clusters**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → *runway.cfngin.hooks.ecs.CreateClustersResponseTypeDef*

Create ECS clusters.

Parameters **context** – CFNgin context object.

runway.cfngin.hooks.iam module

AWS IAM hook.

class runway.cfngin.hooks.iam.**CreateEcsServiceRoleHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for `create_ecs_service_role`.

role_name: *str*

Name of the role to create.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any], **kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.iam.**EnsureServerCertExistsHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for ensure_server_cert_exists.

cert_name: str

Name of the certificate that should exist.

path_to_certificate: Optional[str]

Path to certificate file.

path_to_chain: Optional[str]

Path to chain file.

path_to_private_key: Optional[str]

Path to private key file.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model

- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

prompt: bool

Whether to prompt to upload a certificate if one does not exist.

runway.cfngin.hooks.iam.**create_ecs_service_role**(*context: runway.context.CfnginContext*, **__args: Any*, ***kwargs: Any*) → bool

Create ecsServiceRole IAM role.

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using-service-linked-roles.html

Parameters **context** – Context instance. (passed in by CFNgin)

runway.cfngin.hooks.iam.**ensure_server_cert_exists**(*context: runway.context.CfnginContext*, **__args: Any*, ***kwargs: Any*) → Dict[str, str]

Ensure server cert exists.

Parameters **context** – CFNgin context object.

Returns Dict containing `status`, `cert_name`, and `cert_arn`.

runway.cfngin.hooks.keypair module

AWS EC2 keypair hook.

class runway.cfngin.hooks.keypair.**EnsureKeypairExistsHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for ensure_keypair_exists.

keypair: str

Name of the key pair to ensure exists.

public_key_path: Optional[str]

Path to a public key file to be imported instead of generating a new key. Incompatible with the SSM options, as the private key will not be available for storing.

ssm_key_id: Optional[str]

ID of a KMS key to encrypt the SSM parameter with. If omitted, the default key will be used.

ssm_parameter_name: Optional[str]

Path to an SSM store parameter to receive the generated private key instead of importing it or storing it locally.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises AttributeError – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.hooks.keypair.**KeyPairInfo**

Bases: typing_extensions.TypedDict

Value returned from get_existing_key_pair.

runway.cfngin.hooks.keypair.**get_existing_key_pair**(*ec2: EC2Client*, *keypair_name: str*) → Optional[*KeyPairInfo*]

Get existing keypair.

runway.cfngin.hooks.keypair.**import_key_pair**(*ec2: EC2Client*, *keypair_name: str*, *public_key_data: bytes*) → ImportKeyPairResultTypeDef

Import keypair.

runway.cfngin.hooks.keypair.**read_public_key_file**(*path: pathlib.Path*) → Optional[bytes]

Read public key file.

runway.cfngin.hooks.keypair.**create_key_pair_from_public_key_file**(*ec2: EC2Client*, *keypair_name: str*, *public_key_path: Path*) → Optional[*KeyPairInfo*]

Create keypair from public key file.

runway.cfngin.hooks.keypair.**create_key_pair_in_ssm**(*ec2: EC2Client*, *ssm: SSMClient*, *keypair_name: str*, *parameter_name: str*, *kms_key_id: Optional[str] = None*) → Optional[*KeyPairInfo*]

Create keypair in SSM.

runway.cfngin.hooks.keypair.**create_key_pair**(*ec2: EC2Client*, *keypair_name: str*) → KeyPairTypeDef

Create keypair.

runway.cfngin.hooks.keypair.**create_key_pair_local**(*ec2: EC2Client*, *keypair_name: str*, *dest_dir: Path*) → Optional[*KeyPairInfo*]

Create local keypair.

runway.cfngin.hooks.keypair.**interactive_prompt**(*keypair_name: str*) → Tuple[typing.Optional[typing_extensions.Literal[create, import]], typing.Optional[str]]

Interactive prompt.

runway.cfngin.hooks.keypair.**ensure_keypair_exists**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → *runway.cfngin.hooks.keypair.KeyPairInfo*

Ensure a specific keypair exists within AWS.

If the key doesn't exist, upload it.

runway.cfngin.hooks.protocols module

Protocols for structural typing.

For more information on protocols, refer to PEP 544.

class runway.cfngin.hooks.protocols.**CfnginHookArgsProtocol**

Bases: typing_extensions.Protocol

Protocol for CFNgin hook arguments class.

This class defines a structural interface for all CFNgin hook argument classes. It is recommended to use the provided base class in place of this when authoring a new argument class.

abstract get(*__name: str*) → Optional[Any]

abstract get(*__name: str*, *__default: Union[Any, runway.cfngin.hooks.protocols._T]*) → Union[Any, runway.cfngin.hooks.protocols._T]

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

__init__(**args*, ***kwargs*)

__new__(***kwargs*)

class runway.cfngin.hooks.protocols.**CfnginHookProtocol**

Bases: typing_extensions.Protocol

Protocol for CFNgin hooks.

This class defines a structural interface for all CFNgin hook classes. Classes used for hooks do not need to subclass this hook. They only need to implement a similar interface. While not required, it is still acceptable to subclass this class for full type checking of a hook class.

args: *runway.cfngin.hooks.protocols.CfnginHookArgsProtocol*

Arguments passed to the hook and parsed into an object.

abstract __init__(*context:* runway.context.CfnginContext, ***_kwargs: Any*) → None

Structural __init__ method.

This should not be called. Pylint will erroneously warn about "super-init-not-called" if using this class as a subclass. This should be disabled in-line until the bug reports for this issue is resolved.

abstract post_deploy() → Any

Run during the **post_deploy** stage.

Returns A "truthy" value if the hook was successful or a "falsy" value if the hook failed.

__new__(***kwargs*)

abstract post_destroy() → Any

Run during the **post_destroy** stage.

Returns A "truthy" value if the hook was successful or a "falsy" value if the hook failed.

abstract pre_deploy() → Any

Run during the **pre_deploy** stage.

Returns A "truthy" value if the hook was successful or a "falsy" value if the hook failed.

abstract pre_destroy() → Any

Run during the **pre_destroy** stage.

Returns A "truthy" value if the hook was successful or a "falsy" value if the hook failed.

runway.cfngin.hooks.route53 module

AWS Route 53 hook.

class runway.cfngin.hooks.route53.**CreateDomainHookArgs**

Bases: *runway.utils.BaseModel*

Hook arguments for `create_domain`.

domain: str

Domain name for the Route 53 hosted zone to be created.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.route53.**create_domain**(*context:* runway.context.CfnginContext, **__args: Any*, ***kwargs: Any*) → Dict[str, str]

Create a domain within route53.

Parameters **context** – CFNgin context object.

Returns Dict containing `domain` and `zone_id`.

runway.cfngin.hooks.utils module

Hook utils.

class runway.cfngin.hooks.utils.**BlankBlueprint**

Bases: *runway.cfngin.blueprints.base.Blueprint*

Blueprint that can be built programmatically.

create_template() → None

Create template without raising NotImplementedError.

__init__(*name: str*, *context:* runway.context.CfnginContext, *, *description: Optional[str] = None*, *mappings: Optional[Dict[str, Dict[str, Any]]] = None*, *template: Optional[troposphere.Template] = None*, ***_: Any*)

Instantiate class.

Parameters

- **name** – A name for the blueprint.
- **context** – Context the blueprint is being executed under.
- **description** – The description of the CloudFormation template that will be generated by this blueprint.
- **mappings** – CloudFormation Mappings to be used in the template during the rendering process.
- **template** – Optionally, provide a preexisting Template.

Changed in version 2.0.0: Added `template`.

Changed in version 2.0.0: Class only takes 2 positional arguments. The rest are now keyword arguments.

__new__(***kwargs*)

add_output(*name: str*, *value: Any*) → None

Add an output to the template.

Wrapper for `self.template.add_output(Output(name, Value=value))`.

Parameters

- **name** – The name of the output to create.
- **value** – The value to put in the output.

property cfn_parameters: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters.

property defined_variables: Dict[str, *BlueprintVariableTypeDef*]

Return a copy of `VARIABLES` to avoid accidental modification of the ClassVar.

Changed in version 2.0.0: Changed from a method to a property.

get_cfn_parameters() → Dict[str, Union[List[Any], str]]

Return a dictionary of variables with *type* `CFNType`.

Deprecated since version 2.0.0: Replaced by *cfn_parameters*.

Returns Variables that need to be submitted as CloudFormation Parameters.

get_output_definitions() → Dict[str, Dict[str, Any]]

Get the output definitions.

Deprecated since version 2.0.0: Replaced by *output_definitions*.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

get_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose *type* is an instance of *CFNType* will be returned as a CloudFormation Parameter.

Deprecated since version 2.0.0: Replaced by *parameter_definitions*.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

get_parameter_values() → Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

Deprecated since version 2.0.0: Replaced by *parameter_values*.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

get_required_parameter_definitions() → Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

Deprecated since version 2.0.0: Replaced by *required_parameter_definitions*.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

get_variables() → Dict[str, Any]

Return a dictionary of variables available to the template.

These variables will have been defined within *VARIABLES* or *self.defined_variables*. Any variable value that contains a lookup will have been resolved.

Deprecated since version 2.0.0: Replaced by *variables*.

Returns Variables available to the template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

import_mappings() → None

Import mappings from CFNgin config to the blueprint.

property output_definitions: Dict[str, Dict[str, Any]]

Get the output definitions.

New in version 2.0.0.

Returns Output definitions. Keys are output names, the values are dicts containing key/values for various output properties.

property parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Get the parameter definitions to submit to CloudFormation.

Any variable definition whose type is an instance of *CFNType* will be returned as a CloudFormation Parameter.

New in version 2.0.0.

Returns Parameter definitions. Keys are parameter names, the values are dicts containing key/values for various parameter properties.

property parameter_values: Dict[str, Union[List[Any], str]]

Return a dict of variables with type *CFNType*.

New in version 2.0.0.

Returns Variables that need to be submitted as CloudFormation Parameters. Will be a dictionary of <parameter name>: <parameter value>.

read_user_data(*user_data_path: str*) → str

Read and parse a user_data file.

Parameters **user_data_path** – Path to the userdata file.

render_template() → Tuple[str, str]

Render the Blueprint to a CloudFormation template.

property rendered: str

Return rendered blueprint.

property required_parameter_definitions: Dict[str, *BlueprintVariableTypeDef*]

Return all template parameters that do not have a default value.

New in version 2.0.0.

Returns Dict of required CloudFormation Parameters for the blueprint. Will be a dictionary of `<parameter name>: <parameter attributes>`.

property requires_change_set: bool

Return true if the underlying template has transforms.

reset_template() → None

Reset template.

resolve_variables(*provided_variables: List[*runway.variables.Variable*]*) → None

Resolve the values of the blueprint variables.

This will resolve the values of the *VARIABLES* with values from the env file, the config, and any lookups resolved.

Parameters **provided_variables** – List of provided variables.

set_template_description(*description: str*) → None

Add a description to the Template.

Parameters **description** – A description to be added to the resulting template.

setup_parameters() → None

Add any CloudFormation parameters to the template.

to_json(*variables: Optional[Dict[str, Any]] = None*) → str

Render the blueprint and return the template in json form.

Parameters **variables** – Dictionary providing/overriding variable values.

property variables: Dict[str, Any]

Return a Dict of variables available to the Template.

These variables will have been defined within `VARIABLES` or *defined_variables*. Any variable value that contains a Lookup will have been resolved.

New in version 2.0.0.

Returns Variables available to the Template.

Raises ***UnresolvedBlueprintVariables*** – If variables are unresolved.

property version: str

Template version.

class runway.cfngin.hooks.utils.**TagDataModel**

Bases: *runway.utils.BaseModel*

AWS Resource Tag data model.

class Config

Bases: `object`

Model configuration.

__init__()

__new__(***kwargs*)

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any], **kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.cfngin.hooks.utils.**full_path**(*path: str*) → str

Return full path.

runway.cfngin.hooks.utils.**handle_hooks**(*stage: str*, *hooks: List[*CfnginHookDefinitionModel*]*, *provider:* Provider, *context:* CfnginContext)

Handle pre/post_deploy hooks.

These are pieces of code that we want to run before/after deploying stacks.

Parameters

- **stage** – The current stage (pre_run, post_run, etc).
- **hooks** – Hooks to execute.
- **provider** – Provider instance.
- **context** – Context instance.

runway.cfngin.logger package

CFNgin logger.

class runway.cfngin.logger.**ColorFormatter**

Bases: `logging.Formatter`

Handles colorizing formatted log messages if color provided.

format(*record: logging.LogRecord*) → str

Format log message.

__init__(*fmt=None*, *datefmt=None*, *style='%'*, *validate=True*)

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument. If datefmt is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of '%', '{' or '$' to specify that you want to use one of %-formatting, `str.format()` ({}) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

__new__(***kwargs*)

converter()

localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

formatException(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

formatStack(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime(*record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime() is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

usesTime()

Check if the format uses the creation time of the record.

runway.cfngin.logger.**setup_logging**(*verbosity: int*, *formats: Optional[Dict[str, Any]] = None*)

Configure a proper logger based on verbosity and optional log formats.

Parameters

- **verbosity** – 0, 1, 2
- **formats** – Keys (`info`, `color`, `debug`) which may override the associated default log formats.

runway.cfngin.lookups package

CFNgin lookups.

runway.cfngin.lookups.**register_lookup_handler**(*lookup_type: str*, *handler_or_path: Union[str, Type[runway.lookups.handlers.base.LookupHandler]]*) → None

Register a lookup handler.

Parameters

- **lookup_type** – Name to register the handler under.
- **handler_or_path** – A function or a path to a handler.

runway.cfngin.lookups.**unregister_lookup_handler**(*lookup_type: str*) → None

Unregister the specified lookup type.

This is useful when testing various lookup types if you want to unregister the lookup type after the test runs.

Parameters **lookup_type** – Name of the lookup type to unregister.

Subpackages

runway.cfngin.lookups.handlers package

Import modules.

Submodules

runway.cfngin.lookups.handlers.ami module

AMI lookup.

class runway.cfngin.lookups.handlers.ami.**ArgsDataModel**

Bases: *runway.utils.BaseModel*

Arguments data model.

Any other arguments specified are sent as filters to the AWS API. For example, `architecture:x86_64` will add a filter.

executable_users: Optional[List[str]]

List of executable users.

owners: List[str]

At least one owner is required.

Should be `amazon`, `self`, or an AWS account ID.

region: Optional[str]

AWS region.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(**kwargs*)

__pretty__(*fmt: Callable[[Any], Any], **kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

exception runway.cfngin.lookups.handlers.ami.**ImageNotFound**

Bases: Exception

Image not found.

__init__(*search_string: str*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class runway.cfngin.lookups.handlers.ami.**AmiLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

AMI lookup.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises TypeError – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

TYPE_NAME: Final[typing_extensions.Literal[ami]] = 'ami'

Name that the Lookup is registered as.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to the lookup.

This overrides the default parsing to account for special requirements.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod handle(*value: str*, *context:* runway.context.CfnginContext, **_AmiLookup__args: Any*, ***_AmiLookup__kwargs: Any*) → str

Fetch the most recent AMI Id using a filter.

Parameters

- **value** – Parameter(s) given to this lookup.
- **context** – Context instance.

Example

The above fetches the most recent AMI where owner is self account or amazon and the ami name matches the regex described, the architecture will be either x64 or i386

You can also optionally specify the region in which to perform the AMI lookup.

runway.cfngin.lookups.handlers.awslambda module

Dedicated lookup for use with *AwsLambdaHook* based hooks.

To use this hook, there must be a *AwsLambdaHook* based hook defined in the *pre_deploy* section of the CFNgin configuration file. This hook must also define a *data_key* that is unique within the CFNgin configuration file (it can be reused in other CFNgin configuration files). The *data_key* is then passed to the lookup as it's input/query. This allows the lookup to function during a `runway plan`.

class runway.cfngin.lookups.handlers.awslambda.**AwsLambdaLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod get_deployment_package_data(*context:* CfnginContext, *data_key: str*) → *AwsLambdaHookDeployResponse*

Get the response of an AwsLambdaHook run.

Parameters

- **context** – CFNgin context object.
- **data_key** – The value of the `data_key` field as assigned in a Hook definition.

Returns The *AwsLambdaHook* response parsed into a data model. This will come from hook data if it exists or it will be calculated and added to hook data for future use.

Raises `TypeError` – The data stored in hook data does not align with the expected data model.

static get_required_hook_definition(*config:* CfnginConfig, *data_key: str*) → *CfnginHookDefinitionModel*

Get the required Hook definition from the CFNgin config.

Currently, this only supports finding the data_key pre_deploy.

Parameters

- **config** – CFNgin config being processed.
- **data_key** – The value of the `data_key` field as assigned in a Hook definition.

Returns The Hook definition set to use the provided `data_key`.

Raises `ValueError` – Either a Hook definition was not found for the provided `data_key` or, more than one was found.

classmethod handle(*value: str*, *context: Union[*CfnginContext, RunwayContext*]*, **_args: Any*, ***_kwargs: Any*) → *AwsLambdaHookDeployResponse*

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns The full `AwsLambdaHookDeployResponse` data model.

static init_hook_class(*context:* CfnginContext, *hook_def:* CfnginHookDefinitionModel) → *AwsLambdaHook*[Any]

Initialize AwsLambdaHook subclass instance.

Parameters

- **context** – CFNgin context object.
- **hook_def** – The *AwsLambdaHook* definition.

Returns The loaded AwsLambdaHook object.

class Code

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[*CfnginContext, RunwayContext*]*, **args: Any*, ***kwargs: Any*) → *Code*

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Function.Code`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters
- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters
- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class CodeSha256

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[*CfnginContext, RunwayContext*]*, **args: Any*, ***kwargs: Any*) → str

Retrieve metadata for an AWS Lambda deployment package.

Parameters
- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Version.CodeSha256`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters
- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises TypeError – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class CompatibleArchitectures

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → Optional[List[str]]

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::LayerVersion.CompatibleArchitectures`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters
- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:
1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters
- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters
- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class CompatibleRuntimes

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → Any

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::LayerVersion.CompatibleRuntimes`.

`__init__`()

`__new__`(***kwargs*)

`classmethod dependencies`(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

`classmethod format_results`(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *`load()`* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *`transform()`* if `transform` is provided.

`classmethod load`(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *`format_results()`*. If a lookup needs to handling loading data to process it before it enters *`format_results()`*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

`classmethod parse`(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

`classmethod transform`(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *`format_results()`*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.

- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class Content

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → *Content*

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::LayerVersion.Content`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.
Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters
- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class LicenseInfo

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[*CfnginContext, RunwayContext*]*, **args: Any*, ***kwargs: Any*) → Optional[str]

Retrieve metadata for an AWS Lambda deployment package.

Parameters
- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::LayerVersion.LicenseInfo`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters
- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises **TypeError** – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:
1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters
- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters
- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class Runtime

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → str

Retrieve metadata for an AWS Lambda deployment package.

Parameters
- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Function.Runtime`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters
- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.

- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class S3Bucket

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → str

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Function.Code.S3Bucket` or `AWS::Lambda::LayerVersion.Content.S3Bucket`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters
- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:
1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters
- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters
- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class S3Key

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → str

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Function.Code.S3Key` or `AWS::Lambda::LayerVersion.Content.S3Key`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class S3ObjectVersion

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup for AwsLambdaHook responses.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **args: Any*, ***kwargs: Any*) → Optional[str]

Retrieve metadata for an AWS Lambda deployment package.

Parameters

- **value** – Value to resolve.
- **context** – The current context object.

Returns Value that can be passed into CloudFormation property `AWS::Lambda::Function.Code.S3ObjectVersion` or `AWS::Lambda::LayerVersion.Content.S3ObjectVersion`.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises **TypeError** – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.default module

Lookup to provide a default value.

class runway.cfngin.lookups.handlers.default.**DefaultLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Lookup to provide a default value.

TYPE_NAME: Final[typing_extensions.Literal[default]] = 'default'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Optional[CfnginContext] = None*, ***_: Any*) → Any

Use a value from the environment or fall back to a default value.

Allows defaults to be set at the config file level.

Parameters

- **value** – Parameter(s) given to this lookup. `<env_var>::<default value>`
- **context** – Context instance.

Example

```
Groups: ${default app_security_groups::sg-12345,sg-67890}
```

If `app_security_groups` is defined in the environment, its defined value will be returned. Otherwise, `sg-12345,sg-67890` will be the returned value.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.dynamodb module

DynamoDB lookup.

class runway.cfngin.lookups.handlers.dynamodb.**ArgsDataModel**

Bases: *runway.utils.BaseModel*

Arguments data model.

region: Optional[str]

AWS region.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.lookups.handlers.dynamodb.**QueryDataModel**

Bases: *runway.utils.BaseModel*

Arguments data model.

attribute: str

The attribute to be returned by this lookup. Supports additional syntax to retrieve a nested value.

partition_key: str

The DynamoDB Table's partition key.

partition_key_value: str

The value of the partition key to query the database.

table_name: str

Name of the DynamoDB Table to query.

property item_key: Dict[str, Dict[typing_extensions.Literal[B, N, S], Any]]

Value to pass to boto3 `.get_item()` call as the `Key` argument.

Raises **ValueError** – The value of `partition_key_value` doesn't match the required regex and so it can't be parsed.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(**kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, **kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.lookups.handlers.dynamodb.**DynamodbLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

DynamoDB lookup.

TYPE_NAME: Final[typing_extensions.Literal[dynamodb]] = 'dynamodb'

Name that the Lookup is registered as.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to the lookup.

This overrides the default parsing to account for special requirements.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

Raises **ValueError** – The value provided does not appear to contain the name of a DynamoDB Table. The name of a Table is required.

classmethod parse_query(*value: str*) → *runway.cfngin.lookups.handlers.dynamodb.QueryDataModel*

Parse query string to extract. Does not support arguments in `value`.

Raises **ValueError** – The argument provided does not match the expected format defined with a regex pattern.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises **TypeError** – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context:* runway.context.CfnginContext, **_DynamodbLookup__args: Any*, ***_DynamodbLookup__kwargs: Any*) → Any

Get a value from a DynamoDB table.

Parameters

- **value** – Parameter(s) given to this lookup. `[<region>:]<tablename>@<partitionkey>:<keyvalue>.<keyvalue>...`
- **context** – Context instance.

Raises `ValueError` – The value provided to the lookup resulted in an error.

Note: The region is optional, and defaults to the environment's `AWS_DEFAULT_REGION` if not specified.

---

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

class runway.cfngin.lookups.handlers.dynamodb.**ParsedLookupKey**

Bases: `typing_extensions.TypedDict`

Return value of _lookup_key_parse.

runway.cfngin.lookups.handlers.envvar module

Environment variable lookup.

class runway.cfngin.lookups.handlers.envvar.**EnvvarLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Environment variable lookup.

TYPE_NAME: Final[typing_extensions.Literal[envvar]] = 'envvar'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, ***_: Any*) → str

Retrieve an environment variable.

Parameters **value** – Parameter(s) given to this lookup.

Example

```
# With CFNgin we would reference the environment variable like this:
conf_key: ${envvar ENV_VAR_NAME}
```

You can optionally store the value in a file, ie:

```
$ cat envvar_value.txt
ENV_VAR_NAME
```

and reference it within CFNgin (NOTE: the path should be relative to the CFNgin config file):

```
conf_key: ${envvar file://envvar_value.txt}

# Both of the above would resolve to
conf_key: ENV_VALUE
```

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.file module

File lookup.

class runway.cfngin.lookups.handlers.file.**ArgsDataModel**

Bases: *runway.utils.BaseModel*

Arguments data model.

codec: str

Codec that will be used to parse and/or manipulate the data.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod **__try_update_forward_refs__**(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod **construct**(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.cfngin.lookups.handlers.file.**FileLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

File lookup.

TYPE_NAME: Final[typing_extensions.Literal[file]] = 'file'

Name that the Lookup is registered as.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to the lookup.

This overrides the default parsing to account for special requirements.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

Raises `ValueError` – The value provided does not match the expected regex.

classmethod handle(*value: str*, ***_: Any*) → Any

Translate a filename into the file contents.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.file.**parameterized_codec**(*raw: str*, *b64: typing_extensions.Literal[False] = False*) → troposphere.GenericHelperFn

runway.cfngin.lookups.handlers.file.**parameterized_codec**(*raw: str*, *b64: typing_extensions.Literal[True] = False*) → troposphere.Base64

Parameterize a string, possibly encoding it as Base64 afterwards.

Parameters

- **raw** – String to be processed. Byte strings will be interpreted as UTF-8.
- **b64** – Whether to wrap the output in a Base64 CloudFormation call.

Returns Output to be included in a CloudFormation template.

Return type troposphere.AWSHelperFn

runway.cfngin.lookups.handlers.file.**yaml_codec**(*raw: str*, *parameterized: bool = False*) → Any

YAML codec.

runway.cfngin.lookups.handlers.file.**json_codec**(*raw: str*, *parameterized: bool = False*) → Any

JSON codec.

runway.cfngin.lookups.handlers.hook_data module

Hook data lookup.

class runway.cfngin.lookups.handlers.hook_data.**HookDataLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Hook data lookup.

TYPE_NAME: Final[typing_extensions.Literal[hook_data]] = 'hook_data'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.

- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *`load()`* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *`transform()`* if `transform` is provided.

classmethod handle(*value: str*, *context:* runway.context.CfnginContext, ***_: Any*) → Any

Return the data from `hook_data`.

Parameters

- **value** – Parameter(s) given to this lookup.
- **context** – Context instance.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *`format_results()`*. If a lookup needs to handling loading data to process it before it enters *`format_results()`*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *`format_results()`*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.kms module

AWS KMS lookup.

class runway.cfngin.lookups.handlers.kms.**KmsLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

AWS KMS lookup.

TYPE_NAME: Final[typing_extensions.Literal[kms]] = 'kms'

Name that the Lookup is registered as.

classmethod legacy_parse(*value: str*) → Tuple[str, Dict[str, str]]

Retain support for legacy lookup syntax.

Format of value:

```
<region>@<encrypted-blob>
```

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context:* runway.context.CfnginContext, ***_: Any*) → str

Decrypt the specified value with a master key in KMS.

Parameters

- **value** – Parameter(s) given to this lookup.
- **context** – Context instance.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.output module

AWS CloudFormation Output lookup.

class runway.cfngin.lookups.handlers.output.**OutputQuery**

Bases: NamedTuple

Output query NamedTuple.

stack_name: str

Alias for field number 0

output_name: str

Alias for field number 1

__init__()

static __new__(*_cls*, *stack_name: str*, *output_name: str*)

Create new instance of OutputQuery(stack_name, output_name)

count(*value*, */*)

Return number of occurrences of value.

index(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

class runway.cfngin.lookups.handlers.output.**OutputLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

AWS CloudFormation Output lookup.

TYPE_NAME: Final[typing_extensions.Literal[output]] = 'output'

Name that the Lookup is registered as.

classmethod legacy_parse(*value: str*) → Tuple[*runway.cfngin.lookups.handlers.output.OutputQuery*, Dict[str, str]]

Retain support for legacy lookup syntax.

Format of value: <relative-stack-name>::<OutputName>

classmethod handle(*value: str*, *context:* runway.context.CfnginContext, ***_: Any*) → str

Fetch an output from the designated stack.

Parameters

- **value** – Parameter(s) given to this lookup. `<relative-stack-name>.<OutputName>`
- **context** – Context instance.

Returns Output from the specified stack.

Raises

- ***OutputDoesNotExist*** – Output not found for Stack.
- ***StackDoesNotExist*** – Stack not found for the name provided.

classmethod dependencies(*lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

Parameters **lookup_query** – Parameter(s) given to this lookup.

Returns Stack names this lookup depends on.

__init__()

__new__(***kwargs*)

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.output.**deconstruct**(*value: str*) → *runway.cfngin.lookups.handlers.output.OutputQuery*

Deconstruct the value.

runway.cfngin.lookups.handlers.rxref module

Handler for fetching outputs from a stack in the current namespace.

class runway.cfngin.lookups.handlers.rxref.**RxrefLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Rxref lookup.

TYPE_NAME: Final[typing_extensions.Literal[rxref]] = 'rxref'

Name that the Lookup is registered as.

classmethod legacy_parse(*value: str*) → Tuple[*runway.cfngin.lookups.handlers.output.OutputQuery*, Dict[str, str]]

Retain support for legacy lookup syntax.

Format of value: <relative-stack-name>::<OutputName>

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: CfnginContext*, *provider: Provider*, ***_: Any*) → Any

Fetch an output from the designated stack in the current namespace.

The `output` lookup supports fetching outputs from stacks created within a single config file. Sometimes it's useful to fetch outputs from stacks created outside of the current config file but using the same namespace. `rxref` supports this by using the *runway.context.CfnginContext* to expand the fqn of the stack.

Parameters

- **value** – Parameter(s) given to this lookup. *"<relative-stack-name>.<OutputName>`*
- **context** – Context instance.
- **provider** – Provider instance.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.split module

Split lookup.

class runway.cfngin.lookups.handlers.split.**SplitLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Split lookup.

TYPE_NAME: Final[typing_extensions.Literal[split]] = 'split'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, ***_: Any*) → List[str]

Split the supplied string on the given delimiter, providing a list.

Parameters **value** – Parameter(s) given to this lookup.

Format of value:

```
<delimiter>::<value>
```

Example

```
Subnets: ${split ,::subnet-1,subnet-2,subnet-3}
```

Would result in the variable *Subnets* getting a list consisting of:

```
["subnet-1", "subnet-2", "subnet-3"]
```

This is particularly useful when getting an output from another stack that contains a list. For example, the standard vpc blueprint outputs the list of Subnets it creates as a pair of Outputs (`PublicSubnets`, `PrivateSubnets`) that are comma separated, so you could use this in your config:

```
Subnets: ${split ,::${output vpc.PrivateSubnets}}
```

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.cfngin.lookups.handlers.xref module

Handler for fetching outputs from fully qualified stacks.

class runway.cfngin.lookups.handlers.xref.**XrefLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Xref lookup.

TYPE_NAME: Final[typing_extensions.Literal[xref]] = 'xref'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *provider:* Provider, ***_: Any*) → str

Fetch an output from the designated, fully qualified stack.

The *output* handler supports fetching outputs from stacks created within a single config file. Sometimes it's useful to fetch outputs from stacks created outside of the current config file. *xref* supports this by **not** using the *runway.context.CfnginContext* to expand the fqn of the stack.

Parameters

- **value** – Parameter(s) given to this lookup. `<stack_name>::<output_name>`
- **provider** – Provider instance.

Returns Output from the specified stack.

Example

```
conf_value: ${xref fully-qualified-stack-name::SomeOutputName}
```

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

Submodules

runway.cfngin.lookups.registry module

CFNgin lookup registry.

runway.cfngin.lookups.registry.**register_lookup_handler**(*lookup_type: str*, *handler_or_path: Union[str, Type[runway.lookups.handlers.base.LookupHandler]]*) → None

Register a lookup handler.

Parameters

- **lookup_type** – Name to register the handler under.
- **handler_or_path** – A function or a path to a handler.

runway.cfngin.lookups.registry.**unregister_lookup_handler**(*lookup_type: str*) → None

Unregister the specified lookup type.

This is useful when testing various lookup types if you want to unregister the lookup type after the test runs.

Parameters **lookup_type** – Name of the lookup type to unregister.

runway.cfngin.providers package

Import modules.

Subpackages

runway.cfngin.providers.aws package

Import modules.

Submodules

runway.cfngin.providers.aws.default module

Default AWS Provider.

runway.cfngin.providers.aws.default.**get_cloudformation_client**(*session: boto3.Session*) → CloudFormationClient

Get CloudFormation boto3 client.

runway.cfngin.providers.aws.default.**get_output_dict**(*stack: StackTypeDef*) → Dict[str, str]

Return a dict of key/values for the outputs for a given CF stack.

Parameters **stack** – The stack object to get outputs from.

Returns A dictionary with key/values for each output on the stack.

runway.cfngin.providers.aws.default.**s3_fallback**(*fqn: str*, *template: Template*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, *method: Callable[..., Any]*, *change_set_name: Optional[str] = None*, *service_role: Optional[str] = None*) → Any

Falling back to legacy CFNgin S3 bucket region for templates.

runway.cfngin.providers.aws.default.**get_change_set_name**() → str

Return a valid Change Set Name.

The name has to satisfy the following regex:

```
[a-zA-Z][-a-zA-Z0-9]*
```

And must be unique across all change sets.

runway.cfngin.providers.aws.default.**requires_replacement**(*changeset: List[ChangeTypeDef]*) → List[ChangeTypeDef]

Return the changes within the changeset that require replacement.

Parameters **changeset** – List of changes

runway.cfngin.providers.aws.default.**output_full_changeset**(*full_changeset: Optional[List[ChangeTypeDef]] = None*, *params_diff: Optional[List[DictValue[Any, Any]]] = None*, *answer: Optional[str] = None*, *fqn: Optional[str] = None*) → None

Optionally output full changeset.

Parameters

- **full_changeset** – A list of the full changeset that will be output if the user specifies verbose.
- **params_diff** – A list of DictValue detailing the differences between two parameters returned by *runway.cfngin.actions.diff.diff_dictionaries()*.
- **answer** – Predetermined answer to the prompt if it has already been answered or inferred.
- **fqn** – Fully qualified name of the stack.

runway.cfngin.providers.aws.default.**ask_for_approval**(*full_changeset: Optional[List[ChangeTypeDef]] = None*, *params_diff: Optional[List[*DictValue*[Any, Any]]] = None*, *include_verbose: bool = False*, *fqn: Optional[str] = None*) → None

Prompt the user for approval to execute a change set.

Parameters

- **full_changeset** – A list of the full changeset that will be output if the user specifies verbose.
- **params_diff** – A list of DictValue detailing the differences between two parameters returned by *runway.cfngin.actions.diff.diff_dictionaries()*
- **include_verbose** – Boolean for whether or not to include the verbose option.
- **fqn** – fully qualified name of the stack.

Raises *CancelExecution* – If approval no given.

runway.cfngin.providers.aws.default.**output_summary**(*fqn: str*, *action: str*, *changeset: List[ChangeTypeDef]*, *params_diff: List[*DictValue*[Any, Any]]*, *replacements_only: bool = False*) → None

Log a summary of the changeset.

Parameters

- **fqn** – Fully qualified name of the stack.
- **action** – Action to include in the log message.
- **changeset** – AWS changeset.
- **params_diff** – A list of dictionaries detailing the differences between two parameters returned by *runway.cfngin.actions.diff.diff_dictionaries()*
- **replacements_only** – Boolean for whether or not we only want to list replacements.

runway.cfngin.providers.aws.default.**format_params_diff**(*params_diff: List[*runway.cfngin.actions.diff.DictValue*[Any, Any]]*) → str

Wrap *runway.cfngin.actions.diff.format_params_diff()* for testing.

runway.cfngin.providers.aws.default.**summarize_params_diff**(*params_diff: List[*runway.cfngin.actions.diff.DictValue*[Any, Any]]*) → str

Summarize parameter diff.

runway.cfngin.providers.aws.default.**wait_till_change_set_complete**(*cfn_client: CloudFormationClient*, *change_set_id: str*, *try_count: int = 25*, *sleep_time: float = 0.5*, *max_sleep: float = 3*) → DescribeChangeSetOutputTypeDef

Check state of a changeset, returning when it is in a complete state.

Since changesets can take a little bit of time to get into a complete state, we need to poll it until it does so. This will try to get the state `try_count` times, waiting `sleep_time` * 2 seconds between each try up to the `max_sleep` number of seconds. If, after that time, the changeset is not in a complete state it fails. These default settings will wait a little over one minute.

Parameters

- **cfn_client** – Used to query CloudFormation.
- **change_set_id** – The unique changeset id to wait for.
- **try_count** – Number of times to try the call.
- **sleep_time** – Time to sleep between attempts.
- **max_sleep** – Max time to sleep during backoff

runway.cfngin.providers.aws.default.**create_change_set**(*cfn_client: CloudFormationClient*, *fqn: str*, *template: Template*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, *change_set_type: str = 'UPDATE'*, *service_role: Optional[str] = None*) → Tuple[List[ChangeTypeDef], str]

Create CloudFormation change set.

runway.cfngin.providers.aws.default.**check_tags_contain**(*actual: List[TagTypeDef]*, *expected: List[TagTypeDef]*) → bool

Check if a set of AWS resource tags is contained in another.

Every tag key in `expected` must be present in `actual`, and have the same value. Extra keys in *actual* but not in `expected` are ignored.

Parameters

- **actual** – Set of tags to be verified, usually from the description of a resource. Each item must be a `dict` containing `Key` and `Value` items.
- **expected** – Set of tags that must be present in `actual` (in the same format).

runway.cfngin.providers.aws.default.**generate_cloudformation_args**(*stack_name: str*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, *template: Template*, *capabilities: Optional[List[str]] = None*, *change_set_type: Optional[str] = None*, *service_role: Optional[str] = None*, *stack_policy: Optional[Template] = None*, *change_set_name: Optional[str] = None*) → Dict[str, Any]

Generate the args for common CloudFormation API interactions.

This is used for `create_stack`/`update_stack`/`create_change_set` calls in CloudFormation.

Parameters

- **stack_name** – The fully qualified stack name in Cloudformation.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.
- **template** – The template object.

- **capabilities** – A list of capabilities to use when updating Cloudformation.
- **change_set_type** – An optional change set type to use with create_change_set.
- **service_role** – An optional service role to use when interacting with Cloudformation.
- **stack_policy** – A template object representing a stack policy.
- **change_set_name** – An optional change set name to use with create_change_set.

runway.cfngin.providers.aws.default.**generate_stack_policy_args**(*stack_policy: Optional[Template] = None*) → Dict[str, str]

Convert a stack policy object into keyword args.

Parameters **stack_policy** – A template object representing a stack policy.

class runway.cfngin.providers.aws.default.**ProviderBuilder**

Bases: `object`

Implements a Memorized ProviderBuilder for the AWS provider.

__init__(*, *region: Optional[str] = None*, ***kwargs: Any*) → None

Instantiate class.

build(*, *profile: Optional[str] = None*, *region: Optional[str] = None*) → *runway.cfngin.providers.aws.default.Provider*

Get or create the provider for the given region and profile.

__new__(***kwargs*)

class runway.cfngin.providers.aws.default.**Provider**

Bases: *runway.cfngin.providers.base.BaseProvider*

AWS CloudFormation Provider.

__init__(*session: boto3.Session*, *, *interactive: bool = False*, *recreate_failed: bool = False*, *region: Optional[str] = None*, *replacements_only: bool = False*, *service_role: Optional[str] = None*)

Instantiate class.

get_stack(*stack_name: str*, **_args: Any*, ***_kwargs: Any*) → StackTypeDef

Get stack.

static get_stack_status(*stack: StackTypeDef*, **_args: Any*, ***_kwargs: Any*) → str

Get stack status.

static get_stack_status_reason(*stack: StackTypeDef*) → Optional[str]

Get stack status reason.

is_stack_being_destroyed(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'being destroyed'.

is_stack_completed(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'complete'.

is_stack_destroy_possible(*stack: StackTypeDef*) → bool

Whether the status of the stack is able to be cleanly deleted.

is_stack_in_progress(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'in progress'.

is_stack_destroyed(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'deleted'.

is_stack_recreatable(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'recreating'.

is_stack_rolling_back(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'rolling back'.

is_stack_failed(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates it is 'failed'.

is_stack_in_review(*stack: StackTypeDef*) → bool

Whether the status of the stack indicates if 'review in progress'.

tail_stack(*stack: Stack*, *cancel: threading.Event*, *action: Optional[str] = None*, *log_func: Optional[Callable[[StackEventTypeDef], None]] = None*, *retries: Optional[int] = None*) → None

Tail the events of a stack.

get_delete_failed_status_reason(*stack_name: str*) → Optional[str]

Process events and return latest delete failed reason.

Parameters **stack_name** – Name of a CloudFormation Stack.

Returns Reason for the Stack's DELETE_FAILED status if one can be found.

get_event_by_resource_status(*stack_name: str*, *status: str*, ***, *chronological: bool = True*) → Optional[StackEventTypeDef]

Get Stack Event of a given set of resource status.

Parameters

- **stack_name** – Name of a CloudFormation Stack.
- **status** – Resource status to look for.
- **chronological** – Whether to sort events in chronological order before looking for the desired status.

Returns The first Stack Event matching the given status.

get_events(*stack_name: str*, *chronological: bool = True*) → Iterable[StackEventTypeDef]

Get the events in batches and return in chronological order.

get_rollback_status_reason(*stack_name: str*) → Optional[str]

Process events and returns latest roll back reason.

Parameters **stack_name** – Name of a CloudFormation Stack.

Returns Reason for the Stack's rollback status if one can be found.

tail(*stack_name: str, cancel: threading.Event, log_func: Callable[[StackEventTypeDef], None] = <staticmethod object>, sleep_time: int = 5, include_initial: bool = True*) → None

Show and then tail the event log.

destroy_stack(*stack: StackTypeDef*, ***, *action: str = 'destroy'*, *approval: Optional[str] = None*, *force_interactive: bool = False*, ***kwargs: Any*) → None

Destroy a CloudFormation Stack.

Parameters

- **stack** – Stack to be destroyed.
- **action** – Name of the action being executed. This impacts the log message used.
- **approval** – Response to approval prompt.
- **force_interactive** – Always ask for approval.

`create_stack`(*fqn: str*, *template: Template*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, ***, *force_change_set: bool = False*, *stack_policy: Optional[Template] = None*, *termination_protection: bool = False*, *timeout: Optional[int] = None*, ***kwargs: Any*) → None

Create a new Cloudformation stack.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **template** – A Template object to use when creating the stack.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.
- **force_change_set** – Whether or not to force change set use.
- **stack_policy** – A template object representing a stack policy.
- **termination_protection** – End state of the stack's termination protection.
- **timeout** – The amount of time that can pass before the stack status becomes `CREATE_FAILED`.

`select_update_method`(*force_interactive: bool*, *force_change_set: bool*) → Callable[[...], None]

Select the correct update method when updating a stack.

Parameters

- **force_interactive** – Whether or not to force interactive mode no matter what mode the provider is in.
- **force_change_set** – Whether or not to force change set use.

Returns The correct object method to use when updating.

Return type function

`prepare_stack_for_update`(*stack: StackTypeDef*, *tags: List[TagTypeDef]*) → bool

Prepare a stack for updating.

It may involve deleting the stack if is has failed it's initial creation. The deletion is only allowed if:

- The stack contains all the tags configured in the current context;
- The stack is in one of the statuses considered safe to re-create
- `recreate_failed` is enabled, due to either being explicitly enabled by the user, or because interactive mode is on.

Parameters

- **stack** – A stack object returned from get_stack
- **tags** – List of expected tags that must be present in the stack if it must be re-created

Returns True if the stack can be updated, False if it must be re-created

update_stack(*fqn: str*, *template: Template*, *old_parameters: List[ParameterTypeDef]*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, *force_interactive: bool = False*, *force_change_set: bool = False*, *stack_policy: Optional[Template] = None*, *termination_protection: bool = False*, ***kwargs: Any*) → None

Update a Cloudformation stack.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **template** – A Template object to use when updating the stack.
- **old_parameters** – A list of dictionaries that defines the parameter list on the existing Cloudformation stack.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.
- **force_interactive** – A flag that indicates whether the update should be interactive. If set to True, interactive mode will be used no matter if the provider is in interactive mode or not. False will follow the behavior of the provider.
- **force_change_set** – A flag that indicates whether the update must be executed with a change set.
- **stack_policy** – A template object representing a stack policy.
- **termination_protection** – End state of the stack's termination protection.

update_termination_protection(*fqn: str*, *termination_protection: bool*) → None

Update a Stack's termination protection if needed.

Runs before the normal stack update process.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **termination_protection** – End state of the stack's termination protection.

deal_with_changeset_stack_policy(*fqn: str*, *stack_policy: Optional[Template] = None*) → None

Set a stack policy when using changesets.

ChangeSets don't allow you to set stack policies in the same call to update them. This sets it before executing the changeset if the stack policy is passed in.

Parameters

- **fqn** – Fully qualified name of the stack.
- **stack_policy** – A template object representing a stack policy.

interactive_destroy_stack(*fqn: str*, *approval: Optional[str] = None*, ***kwargs: Any*) → None

Delete a CloudFormation stack in interactive mode.

Parameters

- **fqn** – A fully qualified stack name.
- **approval** – Response to approval prompt.

interactive_update_stack(*fqn: str*, *template: Template*, *old_parameters: List[ParameterTypeDef]*, *parameters: List[ParameterTypeDef]*, *stack_policy: Template*, *tags: List[TagTypeDef]*) → None

Update a Cloudformation stack in interactive mode.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **template** – A Template object to use when updating the stack.
- **old_parameters** – A list of dictionaries that defines the parameter list on the existing Cloudformation stack.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **stack_policy** – A template object representing a stack policy.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.

noninteractive_destroy_stack(*fqn: str*, ***_kwargs: Any*) → None

Delete a CloudFormation stack without interaction.

Parameters **fqn** – A fully qualified stack name.

__new__(***kwargs*)

get_output(*stack: str*, *output: str*) → str

Abstract method.

noninteractive_changeset_update(*fqn: str*, *template: Template*, *old_parameters: List[ParameterTypeDef]*, *parameters: List[ParameterTypeDef]*, *stack_policy: Optional[Template]*, *tags: List[TagTypeDef]*) → None

Update a Cloudformation stack using a change set.

This is required for stacks with a defined Transform (i.e. SAM), as the default `update_stack` API cannot be used with them.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **template** – A Template object to use when updating the stack.
- **old_parameters** – A list of dictionaries that defines the parameter list on the existing Cloudformation stack.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **stack_policy** – A template object representing a stack policy.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.

select_destroy_method(*force_interactive: bool*) → Callable[[...], None]

Select the correct destroy method for destroying a stack.

Parameters **force_interactive** – Always ask for approval.

Returns Interactive or non-interactive method to be invoked.

default_update_stack(*fqn: str*, *template: Template*, *old_parameters: List[ParameterTypeDef]*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*, *stack_policy: Optional[Template] = None*) → None

Update a Cloudformation stack in default mode.

Parameters

- **fqn** – The fully qualified name of the Cloudformation stack.
- **template** – A Template object to use when updating the stack.
- **old_parameters** – A list of dictionaries that defines the parameter list on the existing Cloudformation stack.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.
- **stack_policy** – A template object representing a stack policy.

static get_stack_name(*stack: StackTypeDef*) → str

Get stack name.

static get_stack_tags(*stack: StackTypeDef*) → List[*TagTypeDef*]

Get stack tags.

get_outputs(*stack_name: str*, **_args: Any*, ***_kwargs: Any*) → Dict[str, str]

Get stack outputs.

static get_output_dict(*stack: StackTypeDef*) → Dict[str, str]

Get stack outputs dict.

get_stack_info(*stack: StackTypeDef*) → Tuple[str, Dict[str, Union[List[str], str]]]

Get the template and parameters of the stack currently in AWS.

get_stack_changes(*stack: Stack*, *template: Template*, *parameters: List[ParameterTypeDef]*, *tags: List[TagTypeDef]*) → Dict[str, str]

Get the changes from a ChangeSet.

Parameters

- **stack** – The stack to get changes.
- **template** – A Template object to compared to.
- **parameters** – A list of dictionaries that defines the parameter list to be applied to the Cloudformation stack.
- **tags** – A list of dictionaries that defines the tags that should be applied to the Cloudformation stack.

Returns Stack outputs with inferred changes.

static params_as_dict(*parameters_list: List[ParameterTypeDef]*) → Dict[str, Union[List[str], str]]

Parameters as dict.

Submodules

runway.cfngin.providers.base module

Provider base class.

runway.cfngin.providers.base.**not_implemented**(*method: str*) → None

Wrap NotImplimentedError with a formatted message.

class runway.cfngin.providers.base.**BaseProviderBuilder**

Bases: object

ProviderBuilder base class.

build(*region: Optional[str] = None*) → Any

Abstract method.

__init__()

__new__(***kwargs*)

class runway.cfngin.providers.base.**BaseProvider**

Bases: object

Provider base class.

get_stack(*stack_name: str*, **args: Any*, ***kwargs: Any*) → Any

Abstract method.

get_outputs(*stack_name: str*, **args: Any*, ***kwargs: Any*) → Any

Abstract method.

get_output(*stack: str*, *output: str*) → str

Abstract method.

__init__()

__new__(***kwargs*)

class runway.cfngin.providers.base.**Template**

Bases: object

CloudFormation stack template, which could be optionally uploaded to s3.

Presence of the url attribute indicates that the template was uploaded to S3, and the uploaded template should be used for `CreateStack/UpdateStack` calls.

__init__(*url: Optional[str] = None*, *body: Optional[str] = None*) → None

Instantiate class.

__new__(***kwargs*)

Submodules

runway.cfngin.awscli_yamlhelper module

Copy of `awscli.customizations.cloudformation.yamlhelper.py`.

`runway.cfngin.awscli_yamlhelper.`**`intrinsics_multi_constructor`**(*loader: yaml.loader.Loader*, *tag_prefix: str*, *node: yaml.nodes.Node*) → Dict[str, Any]

YAML constructor to parse CloudFormation intrinsics.

This will return a dictionary with key being the intrinsic name

`runway.cfngin.awscli_yamlhelper.`**`yaml_dump`**(*dict_to_dump: Dict[str, Any]*) → str

Dump the dictionary as a YAML document.

`runway.cfngin.awscli_yamlhelper.`**`yaml_parse`**(*yamlstr: str*) → Dict[str, Any]

Parse a yaml string.

runway.cfngin.cfngin module

CFNgin entrypoint.

class `runway.cfngin.cfngin.`**`CFNgin`**

Bases: `object`

Control CFNgin.

`concurrency`

Max number of CFNgin stacks that can be deployed concurrently. If the value is `0`, will be constrained based on the underlying graph.

Type int

`interactive`

Whether or not to prompt the user before taking action.

Type bool

`parameters`

Combination of the parameters provided when initializing the class and any environment files that are found.

Type *runway.utils.MutableMap*

`recreate_failed`

Destroy and re-create stacks that are stuck in a failed state from an initial deployment when updating.

Type bool

`region`

The AWS region where CFNgin is currently being executed.

Type str

`sys_path`

Working directory.

Type pathlib.Path

tail

Whether or not to display all CloudFormation events in the terminal.

Type bool

__init__(*ctx:* runway.context.RunwayContext, *parameters: Optional[Dict[str, Any]] = None*, *sys_path: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters

- **ctx** – Runway context object.
- **parameters** – Parameters from Runway.
- **sys_path** – Working directory.

property env_file: *runway.utils.MutableMap*

Contents of a CFNgin environment file.

deploy(*force: bool = False*, *sys_path: Optional[pathlib.Path] = None*) → None

Run the CFNgin deploy action.

Parameters

- **force** – Explicitly enable the action even if an environment file is not found.
- **sys_path** – Explicitly define a path to work in. If not provided, `self.sys_path` is used.

destroy(*force: bool = False*, *sys_path: Optional[pathlib.Path] = None*) → None

Run the CFNgin destroy action.

Parameters

- **force** – Explicitly enable the action even if an environment file is not found.
- **sys_path** – Explicitly define a path to work in. If not provided, `self.sys_path` is used.

init(*force: bool = False*, *sys_path: Optional[pathlib.Path] = None*) → None

Initialize environment.

load(*config_path: pathlib.Path*) → *runway.context.CfnginContext*

Load a CFNgin config into a context object.

Parameters **config_path** – Valid path to a CFNgin config file.

__new__(***kwargs*)

plan(*force: bool = False*, *sys_path: Optional[pathlib.Path] = None*)

Run the CFNgin plan action.

Parameters

- **force** – Explicitly enable the action even if an environment file is not found.
- **sys_path** – Explicitly define a path to work in. If not provided, `self.sys_path` is used.

should_skip(*force: bool = False*) → bool

Determine if action should be taken or not.

Parameters **force** – If `True`, will always return `False` meaning the action should not be skipped.

classmethod find_config_files(*exclude: Optional[List[str]] = None*, *sys_path: Optional[pathlib.Path] = None*) → List[pathlib.Path]

Find CFNgin config files.

Parameters

- **exclude** – List of file names to exclude. This list is appended to the global exclude list.
- **sys_path** – Explicitly define a path to search for config files.

Returns Paths to config files that were found.

runway.cfngin.environment module

CFNgin environment file parsing.

runway.cfngin.environment.**parse_environment**(*raw_environment: str*) → Dict[str, Any]

Parse environment file contents.

Parameters **raw_environment** – Environment file read into a string.

runway.cfngin.exceptions module

CFNgin exceptions.

exception runway.cfngin.exceptions.**CfnginError**

Bases: *runway.exceptions.RunwayError*

Base class for custom exceptions raised by Runway.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**CancelExecution**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when we want to cancel executing the plan.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**CfnginBucketAccessDenied**

Bases: *runway.cfngin.exceptions.CfnginError*

Access denied to CFNgin bucket.

This can occur when the bucket exists in another AWS account and/or the credentials being used do not have adequate permissions to access the bucket.

__init__(*, *bucket_name: str*) → None

Instantiate class.

Parameters **bucket_name** – Name of the CFNgin bucket.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**CfnginBucketNotFound**

Bases: *runway.cfngin.exceptions.CfnginError*

CFNgin bucket specified or default bucket being used but it does not exist.

This can occur when using a custom stack to deploy the CFNgin bucket but the custom stack does not create bucket that is expected.

__init__(*, *bucket_name: str*) → None

Instantiate class.

Parameters **bucket_name** – Name of the CFNgin bucket.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**CfnginBucketRequired**

Bases: *runway.cfngin.exceptions.CfnginError*

CFNgin bucket is required to use a feature but it not provided/disabled.

__init__(*, *config_path: Optional[AnyPath] = None*, *reason: Optional[str] = None*) → None

Instantiate class.

Parameters

- **config_path** – Path to the CFNgin config file.
- **reason** – Reason why CFNgin bucket is needed.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**CfnginOnlyLookupError**

Bases: *runway.cfngin.exceptions.CfnginError*

Attempted to use a CFNgin lookup outside of CFNgin.

__init__(*lookup_name: str*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**ChangesetDidNotStabilize**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the applying a changeset fails.

__init__(*change_set_id: str*) → None

Instantiate class.

Parameters **change_set_id** – The changeset that failed.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**GraphError**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the graph is invalid (e.g. acyclic dependencies).

__init__(*exception: Exception*, *stack: str*, *dependency: str*) → None

Instantiate class.

Parameters

- **exception** – The exception that was raised by the invalid graph.
- **stack** – Name of the stack causing the error.
- **dependency** – Name of the dependency causing the error.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**ImproperlyConfigured**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when a component is improperly configured.

__init__(*kls: Any*, *error: Exception*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **kls** – The class that was improperly configured.
- **error** – The exception that was raised when trying to use cls.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**InvalidConfig**

Bases: *runway.cfngin.exceptions.CfnginError*

Provided config file is invalid.

__init__(*errors: Union[str, List[Union[Exception, str]]]*) → None

Instantiate class.

Parameters **errors** – Errors or error messages that are raised to identify that a config is invalid.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**InvalidDockerizePipConfiguration**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the provided configuration for dockerized pip is invalid.

__init__(*msg: str*) → None

Instantiate class.

Parameters **msg** – The reason for the error being raised.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**InvalidUserdataPlaceholder**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when a placeholder name in raw_user_data is not valid.

E.g `${100}` would raise this.

__init__(*blueprint_name: str*, *exception_message: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **blueprint_name** – Name of the blueprint with invalid userdata placeholder.
- **exception_message** – Message from the exception that was raised while parsing the user-data.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**MissingEnvironment**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when an environment lookup is used but the key doesn't exist.

__init__(*key: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **key** – The key that was used but doesn't exist in the environment.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**MissingParameterException**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised if a required parameter with no default is missing.

__init__(*parameters: List[str]*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **parameters** – A list of the parameters that are missing.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**MissingVariable**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when a variable with no default is not provided a value.

__init__(*blueprint_name: str*, *variable_name: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **blueprint_name** – Name of the blueprint.
- **variable_name** – Name of the variable missing a value.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PipError**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when pip returns a non-zero exit code.

__init__() → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PipenvError**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when pipenv returns a non-zero exit code.

__init__() → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PersistentGraphCannotLock**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the persistent graph in S3 cannot be locked.

__init__(*reason: str*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PersistentGraphCannotUnlock**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the persistent graph in S3 cannot be unlocked.

__init__(*reason: Union[Exception, str]*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PersistentGraphLocked**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the persistent graph in S3 is lock.

The action being executed requires it to be unlocked before attempted.

__init__(*, *message: Optional[str] = None*, *reason: Optional[str] = None*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PersistentGraphLockCodeMismatch**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the provided persistent graph lock code does not match.

The code used to unlock the persistent graph must match the s3 object lock code.

__init__(*provided_code: str*, *s3_code: Optional[str]*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.**PersistentGraphUnlocked**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when the persistent graph in S3 is unlock.

The action being executed requires it to be locked before attempted.

__init__(*message: Optional[str] = None*, *reason: Optional[str] = None*) → None
Instantiate class.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.PlanFailed
Bases: *runway.cfngin.exceptions.CfnginError*

Raised if any step of a plan fails.

__init__(*failed_steps: List[Step]*, **args: Any*, ***kwargs: Any*) → None
Instantiate class.

Parameters **failed_steps** – The steps that failed.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.StackDidNotChange
Bases: *runway.cfngin.exceptions.CfnginError*

Raised when there are no changes to be made by the provider.

__init__(**args: Any*, ***kwargs: Any*) → None
Instantiate class.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.StackDoesNotExist
Bases: *runway.cfngin.exceptions.CfnginError*

Raised when a stack does not exist in AWS.

__init__(*stack_name: str*, **args: Any*, ***kwargs: Any*) → None
Instantiate class.

Parameters **stack_name** – Name of the stack that does not exist.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.StackUpdateBadStatus
Bases: *runway.cfngin.exceptions.CfnginError*

Raised if the state of a stack can't be handled.

__init__(*stack_name: str*, *stack_status: str*, *reason: str*, **args: Any*, ***kwargs: Any*) → None
Instantiate class.

Parameters

- **stack_name** – Name of the stack.

- **stack_status** – The stack's status.
- **reason** – The reason for the current status.

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.StackFailed

Bases: runway.cfngin.exceptions.CfnginError

Raised when a stack action fails.

Primarily used with hooks that act on stacks.

__init__(*stack_name: str*, *status_reason: Optional[str] = None*) → None

Instantiate class.

Parameters

- **stack_name** – Name of the stack.
- **status_reason** – The reason for the current status.

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.cfngin.exceptions.UnableToExecuteChangeSet

Bases: runway.cfngin.exceptions.CfnginError

Raised if changeset execution status is not `AVAILABLE`.

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*stack_name: str*, *change_set_id: str*, *execution_status: str*) → None

Instantiate class.

Parameters

- **stack_name** – Name of the stack.
- **change_set_id** – The changeset that failed.
- **execution_status** – The value of the changeset's `ExecutionStatus` attribute.

exception runway.cfngin.exceptions.UnhandledChangeSetStatus

Bases: runway.cfngin.exceptions.CfnginError

Raised when creating a changeset failed for an unhandled reason.

Handled failure reasons include: no changes

__new__(**kwargs)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*stack_name: str*, *change_set_id: str*, *status: str*, *status_reason: str*) → None

Instantiate class.

Parameters

- **stack_name** – Name of the stack.
- **change_set_id** – The changeset that failed.
- **status** – The state that could not be handled.
- **status_reason** – Cause of the current state.

exception runway.cfngin.exceptions.**UnresolvedBlueprintVariable**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when trying to use a variable before it has been resolved.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*blueprint_name: str*, *variable:* Variable, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **blueprint_name** – Name of the blueprint that tried to use the unresolved variables.
- **variable** – The unresolved variable.

exception runway.cfngin.exceptions.**UnresolvedBlueprintVariables**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when trying to use variables before they has been resolved.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*blueprint_name: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **blueprint_name** – Name of the blueprint that tried to use the unresolved variables.

exception runway.cfngin.exceptions.**ValidatorError**

Bases: *runway.cfngin.exceptions.CfnginError*

Used for errors raised by custom validators of blueprint variables.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*variable: str*, *validator: str*, *value: str*, *exception: Optional[Exception] = None*) → None

Instantiate class.

Parameters

- **variable** – The variable that failed validation.

- **validator** – The validator that was not passed.
- **value** – The value of the variable that did not pass the validator.
- **exception** – The exception raised by the validator.

`__str__`()

Return the exception's message when converting to a string.

exception runway.cfngin.exceptions.**VariableTypeRequired**

Bases: *runway.cfngin.exceptions.CfnginError*

Raised when a variable defined in a blueprint is missing a type.

`__new__`(***kwargs*)

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`__init__`(*blueprint_name: str*, *variable_name: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **blueprint_name** – Name of the blueprint.
- **variable_name** – Name of the variable missing a type.

runway.cfngin.plan module

CFNgin plan, plan components, and functions for interacting with a plan.

runway.cfngin.plan.**json_serial**(*obj: Set[runway.cfngin.plan._T]*) → List[runway.cfngin.plan._T]

runway.cfngin.plan.**json_serial**(*obj: Union[Dict[Any, Any], int, List[Any], str]*) → NoReturn

Serialize json.

Parameters **obj** – A python object.

Example

json.dumps(data, default=json_serial)

runway.cfngin.plan.**merge_graphs**(*graph1:* runway.cfngin.plan.Graph, *graph2:* runway.cfngin.plan.Graph) → *runway.cfngin.plan.Graph*

Combine two Graphs into one, retaining steps.

Parameters

- **graph1** – Graph that `graph2` will be merged into.
- **graph2** – Graph that will be merged into `graph1`.

class runway.cfngin.plan.**Step**

Bases: `object`

State machine for executing generic actions related to stacks.

fn

Function to run to execute the step. This function will be ran multiple times until the step is "done".

Type Optional[Callable[..., Any]]

last_updated

Time when the step was last updated.

Type float

logger

Logger for logging messages about the step.

Type PrefixAdaptor

stack

the stack associated with this step

Type *Stack*

status

The status of step.

Type *Status*

watch_func

Function that will be called to "tail" the step action.

Type Optional[Callable[..., Any]]

__init__(*stack:* runway.cfngin.stack.Stack, *, *fn: Optional[Callable[[...], Any]] = None*, *watch_func: Optional[Callable[[...], Any]] = None*) → None

Instantiate class.

Parameters

- **stack** – The stack associated with this step
- **fn** – Function to run to execute the step. This function will be ran multiple times until the step is "done".
- **watch_func** – Function that will be called to "tail" the step action.

run() → bool

Run this step until it has completed or been skipped.

property name: str

Name of the step.

This is equal to the name of the stack it operates on.

property requires: Set[str]

Return a list of step names this step depends on.

property required_by: Set[str]

Return a list of step names that depend on this step.

property completed: bool

Return True if the step is in a COMPLETE state.

property skipped: bool

Return True if the step is in a SKIPPED state.

property failed: bool

Return True if the step is in a FAILED state.

property done: bool

Return True if the step is finished.

To be True, status must be either COMPLETE, SKIPPED or FAILED)

property ok: bool

Return True if the step is finished (either COMPLETE or SKIPPED).

property submitted: bool

Return True if the step is SUBMITTED, COMPLETE, or SKIPPED.

set_status(*status:* Status) → None

Set the current step's status.

Parameters **status** – The status to set the step to.

complete() → None

Shortcut for set_status(COMPLETE).

log_step() → None

Construct a log message for a set and log it to the UI.

skip() → None

Shortcut for set_status(SKIPPED).

submit() → None

Shortcut for set_status(SUBMITTED).

classmethod from_stack_name(*stack_name: str*, *context:* CfnginContext, *requires: Optional[Union[List[str], Set[str]]] = None*, *fn: Optional[Callable[..., Status]] = None*, *watch_func: Optional[Callable[..., Any]] = None*) → *Step*

Create a step using only a stack name.

Parameters

- **stack_name** – Name of a CloudFormation stack.
- **context** – Context object. Required to initialize a "fake" *runway.cfngin.stack.Stack*.
- **requires** – Stacks that this stack depends on.
- **fn** – The function to run to execute the step. This function will be ran multiple times until the step is "done".
- **watch_func** – an optional function that will be called to "tail" the step action.

classmethod from_persistent_graph(*graph_dict: Union[Dict[str, List[str]], Dict[str, Set[str]], OrderedDict[str, Set[str]]]*, *context:* CfnginContext, *fn: Optional[Callable[..., Status]] = None*, *watch_func: Optional[Callable[..., Any]] = None*) → List[*Step*]

Create a steps for a persistent graph dict.

Parameters

- **graph_dict** – A graph dict.
- **context** – Context object. Required to initialize a "fake" *runway.cfngin.stack.Stack*.

- **requires** – Stacks that this stack depends on.
- **fn** – The function to run to execute the step. This function will be ran multiple times until the step is "done".
- **watch_func** – an optional function that will be called to "tail" the step action.

__repr__() → str

Object represented as a string.

__str__() → str

Object displayed as a string.

__new__(***kwargs*)

class runway.cfngin.plan.**Graph**

Bases: object

Graph represents a graph of steps.

The *Graph* helps organize the steps needed to execute a particular action for a set of *runway.cfngin.stack.Stack* objects. When initialized with a set of steps, it will first build a Directed Acyclic Graph from the steps and their dependencies.

Example

```
>>> dag = DAG()
>>> a = Step("a", fn=deploy)
>>> b = Step("b", fn=deploy)
>>> dag.add_step(a)
>>> dag.add_step(b)
>>> dag.connect(a, b)
```

__init__(*steps: Optional[Dict[str,* runway.cfngin.plan.Step*]] = None*, *dag:* *Optional[*runway.cfngin.dag.DAG*] = None*) → None

Instantiate class.

Parameters

- **steps** – Dict with key of step name and value of *Step* for steps to initialize the Graph with. Note that if this is provided, a pre-configured *runway.cfngin.dag.DAG* that already includes these steps should also be provided..
- **dag** – An optional *runway.cfngin.dag.DAG* object. If one is not provided, a new one will be initialized.

add_step(*step:* runway.cfngin.plan.Step, *add_dependencies: bool = False*, *add_dependents: bool = False*) → None

Add a step to the graph.

Parameters

- **step** – The step to be added.
- **add_dependencies** – Connect steps that need to be completed before this step.
- **add_dependents** – Connect steps that require this step.

add_step_if_not_exists(*step:* runway.cfngin.plan.Step, *add_dependencies: bool = False*, *add_dependents: bool = False*) → None

Try to add a step to the graph.

Can be used when failure to add is acceptable.

Parameters

- **step** – The step to be added.
- **add_dependencies** – Connect steps that need to be completed before this step.
- **add_dependents** – Connect steps that require this step.

add_steps(*steps: List[*runway.cfngin.plan.Step*]*) → None

Add a list of steps.

Parameters **steps** – The step to be added.

pop(*step:* runway.cfngin.plan.Step, *default: Optional[Any] = None*) → Any

Remove a step from the graph.

Parameters

- **step** – The step to remove from the graph.
- **default** – Returned if the step could not be popped

connect(*step: str*, *dep: str*) → None

Connect a dependency to a step.

Parameters

- **step** – Step name to add a dependency to.
- **dep** – Name of dependent step.

transitive_reduction() → None

Perform a transitive reduction on the underlying DAG.

The transitive reduction of a graph is a graph with as few edges as possible with the same reachability as the original graph.

See https://en.wikipedia.org/wiki/Transitive_reduction

walk(*walker: Callable[[*runway.cfngin.dag.DAG*, Callable[[str], Any]], Any]*, *walk_func: Callable[[*runway.cfngin.plan.Step*], Any]*) → Any

Walk the steps of the graph.

Parameters

- **walker** – Function used to walk the steps.
- **walk_func** – Function called with a `Step` as the only argument for each step of the plan.

downstream(*step_name: str*) → List[*runway.cfngin.plan.Step*]

Return the direct dependencies of the given step.

transposed() → *runway.cfngin.plan.Graph*

Return a "transposed" version of this graph.

Useful for walking in reverse.

filtered(*step_names: List[str]*) → *runway.cfngin.plan.Graph*

Return a "filtered" version of this graph.

Parameters **step_names** – Steps to filter.

topological_sort() → List[*runway.cfngin.plan.Step*]

Perform a topological sort of the underlying DAG.

to_dict() → OrderedDict[str, Set[str]]

Return the underlying DAG as a dictionary.

dumps(*indent: Optional[int] = None*) → str

Output the graph as a json serialized string for storage.

Parameters **indent** – Number of spaces for each indentation.

classmethod from_dict(*graph_dict: Union[Dict[str, List[str]], Dict[str, Set[str]], OrderedDict[str, Set[str]]]*, *context:* runway.context.CfnginContext) → *runway.cfngin.plan.Graph*

Create a Graph from a graph dict.

Parameters

- **graph_dict** – The dictionary used to create the graph.
- **context** – Required to init stacks.

classmethod from_steps(*steps: List[*runway.cfngin.plan.Step*]*) → *runway.cfngin.plan.Graph*

Create a Graph from Steps.

Parameters **steps** – Steps used to create the graph.

__str__() → str

Object displayed as a string.

__new__(***kwargs*)

class runway.cfngin.plan.**Plan**

Bases: object

A convenience class for working on a Graph.

context

Context object.

Type Optional[*CfnginContext*]

description

Plan description.

Type str

graph

Graph of the plan.

Type *Graph*

id

UUID for the plan.

Type uuid.UUID

reverse

The graph has been transposed for walking in reverse.

Type bool

require_unlocked

Require the persistent graph to be unlocked before executing steps.

Type bool

__init__(*description: str*, *graph: Graph*, *context: Optional[CfnginContext] = None*, *reverse: bool = False*, *require_unlocked: bool = True*) → None

Initialize class.

Parameters

- **description** – Description of what the plan is going to do.
- **graph** – Local graph used for the plan.
- **context** – Context object.
- **reverse** – Transpose the graph for walking in reverse.
- **require_unlocked** – Require the persistent graph to be unlocked before executing steps.

__new__(***kwargs*)

outline(*level: int = 20*, *message: str = ''*)

Print an outline of the actions the plan is going to take.

The outline will represent the rough ordering of the steps that will be taken.

Parameters

- **level** – a valid log level that should be used to log the outline
- **message** – a message that will be logged to the user after the outline has been logged.

dump(*, *directory: str*, *context: CfnginContext*, *provider: Optional[Provider] = None*) → Any

Output the rendered blueprint for all stacks in the plan.

Parameters

- **directory** – Directory where files will be created.
- **context** – Current CFNgin context.
- **provider** – Provider to use when resolving the blueprints.

execute(**args: Any*, ***kwargs: Any*)

Walk each step in the underlying graph.

Raises

- ***PersistentGraphLocked*** – Raised if the persistent graph is locked prior to execution and this session did not lock it.
- ***PlanFailed*** – Raised if any of the steps fail.

walk(*walker: Callable[[...], Any]*) → Any

Walk each step in the underlying graph, in topological order.

Parameters **walker** – a walker function to be passed to *runway.cfngin.dag.DAG* to walk the graph.

property lock_code: str

Code to lock/unlock the persistent graph.

property steps: List[*runway.cfngin.plan.Step*]

Return a list of all steps in the plan.

property step_names: List[str]

Return a list of all step names.

keys() → List[str]

Return a list of all step names.

runway.cfngin.session_cache module

CFNgin session caching.

runway.cfngin.session_cache.**get_session**(*region: Optional[str] = None*, *profile: Optional[str] = None*, *access_key: Optional[str] = None*, *secret_key: Optional[str] = None*, *session_token: Optional[str] = None*) → boto3.session.Session

Create a thread-safe boto3 session.

Parameters

- **region** – The region for the session.
- **profile** – The profile for the session.
- **access_key** – AWS Access Key ID.
- **secret_key** – AWS secret Access Key.
- **session_token** – AWS session token.

Returns A thread-safe boto3 session.

runway.cfngin.stack module

CFNgin stack.

class runway.cfngin.stack.**Stack**

Bases: object

Represents gathered information about a stack to be built/updated.

definition

The stack definition from the config.

Type *CfnginStackDefinitionModel*

enabled

Whether this stack is enabled

Type bool

force

Whether to force updates on this stack.

Type bool

fqn

Fully qualified name of the stack. Combines the stack name and current namespace.

Type str

in_progress_behavior

The behavior for when a stack is in `CREATE_IN_PROGRESS` or `UPDATE_IN_PROGRESS`.

Type Optional[Literal['wait']]

locked

Whether or not the stack is locked.

Type bool

logging

Whether logging is enabled.

Type bool

mappings

Cloudformation mappings passed to the blueprint.

Type Dict[str, Dict[str, Dict[str, Any]]]

name

Name of the stack taken from the definition.

Type str

outputs

CloudFormation Stack outputs.

Type Dict[str, Any]

protected

Whether this stack is protected.

Type bool

termination_protection

The state of termination protection to apply to the stack.

Type bool

variables

Variables for the stack.

Type List[*Variable*]

__init__(*definition:* CfnginStackDefinitionModel, *context:* CfnginContext, *, *variables: Optional[Dict[str, Any]] = None*, *mappings: Dict[str, Dict[str, Dict[str, Any]]] = None*, *locked: bool = False*, *force: bool = False*, *enabled: bool = True*, *protected: bool = False*)

Instantiate class.

Parameters

- **definition** – A stack definition.
- **context** – Current context for deploying the stack.
- **variables** – Variables for the stack.
- **mappings** – Cloudformation mappings passed to the blueprint.

- **locked** – Whether or not the stack is locked.
- **force** – Whether to force updates on this stack.
- **enabled** – Whether this stack is enabled
- **protected** – Whether this stack is protected.

__new__(***kwargs*)

property required_by: Set[str]

Return a list of stack names that depend on this stack.

property requires: Set[str]

Return a list of stack names this stack depends on.

property stack_policy: Optional[str]

Return the Stack Policy to use for this stack.

property blueprint: *Blueprint*

Return the blueprint associated with this stack.

property tags: Dict[str, Any]

Return the tags that should be set on this stack.

Includes both the global tags, as well as any stack specific tags or overrides.

property parameter_values: Dict[str, Any]

Return all CloudFormation Parameters for the stack.

CloudFormation Parameters can be specified via Blueprint Variables with a *runway.cfngin.blueprints.variables.types.CFNType* type.

Returns Dictionary of <parameter name>: <parameter value>.

property all_parameter_definitions: Dict[str, Any]

Return all parameters in the blueprint/template.

property required_parameter_definitions: Dict[str, Any]

Return all CloudFormation Parameters without a default value.

resolve(*context:* CfnginContext, *provider: Optional[*Provider*] = None*) → None

Resolve the Stack variables.

This resolves the Stack variables and then prepares the Blueprint for rendering by passing the resolved variables to the Blueprint.

Parameters

- **context** – CFNgin context.
- **provider** – Subclass of the base provider.

set_outputs(*outputs: Dict[str, Any]*) → None

Set stack outputs to the provided value.

Parameters **outputs** – CloudFormation Stack outputs.

__repr__() → str

Object represented as a string.

runway.cfngin.status module

CFNgin statuses.

class runway.cfngin.status.**Status**

Bases: object

CFNgin status base class.

name

Name of the status.

Type str

code

Status code.

Type int

reason

Reason for the status.

Type Optional[str]

__init__(*name: str*, *code: int*, *reason: Optional[str] = None*) → None

Instantiate class.

Parameters

- **name** – Name of the status.
- **code** – Status code.
- **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**CompleteStatus**

Bases: *runway.cfngin.status.Status*

Status name of 'complete' with code of '2'.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**FailedStatus**

Bases: *runway.cfngin.status.Status*

Status name of 'failed' with code of '4'.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**PendingStatus**

Bases: *runway.cfngin.status.Status*

Status name of 'pending' with code of '0'.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**SkippedStatus**

Bases: *runway.cfngin.status.Status*

Status name of 'skipped' with code of '3'.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**SubmittedStatus**

Bases: *runway.cfngin.status.Status*

Status name of 'submitted' with code of '1'.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**DidNotChangeStatus**

Bases: *runway.cfngin.status.SkippedStatus*

Skipped status with a reason of 'nochange'.

__eq__(*other: Any*) → bool

Compare if self is equal to another object.

__ge__(*other: Any*) → bool

Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool

Compare if self is greater than another object.

__init__(*reason: Optional[str] = None*) → None

Instantiate class.

Parameters **reason** – Reason for the status.

__le__(*other: Any*) → bool

Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool

Compare if self is less than another object.

__ne__(*other: Any*) → bool

Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**DoesNotExistInCloudFormation**

Bases: *runway.cfngin.status.SkippedStatus*

Skipped status with a reason of 'does not exist in cloudformation'.

__eq__(*other: Any*) → bool
Compare if self is equal to another object.

__ge__(*other: Any*) → bool
Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool
Compare if self is greater than another object.

__init__(*reason: Optional[str] = None*) → None
Instantiate class.

Parameters **reason** – Reason for the status.

__le__(*other: Any*) → bool
Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool
Compare if self is less than another object.

__ne__(*other: Any*) → bool
Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**NotSubmittedStatus**
Bases: *runway.cfngin.status.SkippedStatus*

Skipped status with a reason of 'disabled'.

__eq__(*other: Any*) → bool
Compare if self is equal to another object.

__ge__(*other: Any*) → bool
Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool
Compare if self is greater than another object.

__init__(*reason: Optional[str] = None*) → None
Instantiate class.

Parameters **reason** – Reason for the status.

__le__(*other: Any*) → bool
Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool
Compare if self is less than another object.

__ne__(*other: Any*) → bool
Compare if self is not equal to another object.

__new__(***kwargs*)

class runway.cfngin.status.**NotUpdatedStatus**
Bases: *runway.cfngin.status.SkippedStatus*

Skipped status with a reason of 'locked'.

__eq__(*other: Any*) → bool
Compare if self is equal to another object.

__ge__(*other: Any*) → bool
Compare if self is greater than equal to another object.

__gt__(*other: Any*) → bool
Compare if self is greater than another object.

__init__(*reason: Optional[str] = None*) → None
Instantiate class.

Parameters **reason** – Reason for the status.

__le__(*other: Any*) → bool
Compare if self is less than or equal to another object.

__lt__(*other: Any*) → bool
Compare if self is less than another object.

__ne__(*other: Any*) → bool
Compare if self is not equal to another object.

__new__(***kwargs*)

runway.cfngin.tokenize_userdata module

Resources to tokenize userdata.

runway.cfngin.tokenize_userdata.**cf_tokenize**(*raw_userdata: str*) → List[str]
Parse UserData for Cloudformation helper functions.

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html

http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html

It breaks apart the given string at each recognized function (see HELPERS global variable) and instantiates the helper function objects in place of those.

Parameters **raw_userdata** – Unparsed userdata data string.

Returns A list of string parts that is useful when used with `troposphere.Join()` and `troposphere.Base64()` to produce userdata.

Example

runway.cfngin.ui module

CFNgin UI manipulation.

runway.cfngin.ui.**get_raw_input**(*message: str*) → str
Just a wrapper for `input()` for testing purposes.

class runway.cfngin.ui.**UI**
Bases: `ContextManager[UI]`

Used internally from terminal output in a multithreaded environment.

Ensures that two threads don't write over each other while asking a user for input (e.g. in interactive mode).

__init__() → None

Instantiate class.

log(*lvl: int, msg: Union[Exception, str], *args: Any, logger: Union[logging.Logger, logging.LoggerAdapter[Any]] = <RunwayLogger runway.cfngin.ui (WARNING)>, **kwargs: Any*) → None

Log the message if the current thread owns the underlying lock.

Parameters

- **lvl** – Log level.
- **msg** – String template or exception to use for the log record.
- **logger** – Specific logger to log to.

info(*msg: str, *args: Any, logger: Union[logging.Logger, logging.LoggerAdapter[Any]] = <RunwayLogger runway.cfngin.ui (WARNING)>, **kwargs: Any*) → None

Log the line if the current thread owns the underlying lock.

Parameters

- **msg** – String template or exception to use for the log record.
- **logger** – Specific logger to log to.

ask(*message: str*) → str

Collect input from a user in a multithreaded environment.

This wraps the built-in input function to ensure that only 1 thread is asking for input from the user at a give time. Any process that tries to log output to the terminal will be blocked while the user is being prompted.

getpass(*prompt: str, stream: Optional[TextIO] = None*) → str

Wrap getpass to lock the UI.

__enter__() → *runway.cfngin.ui.UI*

Enter the context manager.

__exit__(*exc_type: Optional[Type[BaseException]], exc_value: Optional[BaseException], traceback: Optional[TracebackType]*) → None

Exit the context manager.

__new__(***kwargs*)

runway.cfngin.utils module

CFNgin utilities.

runway.cfngin.utils.**camel_to_snake**(*name: str*) → str

Convert CamelCase to snake_case.

Parameters **name** (*str*) – The name to convert from CamelCase to snake_case.

Returns Converted string.

Return type str

runway.cfngin.utils.**convert_class_name**(*kls: type*) → str

Get a string that represents a given class.

Parameters **kls** – The class being analyzed for its name.

Returns The name of the given kls.

runway.cfngin.utils.**parse_zone_id**(*full_zone_id: str*) → str

Parse the returned hosted zone id and returns only the ID itself.

runway.cfngin.utils.**get_hosted_zone_by_name**(*client: Route53Client*, *zone_name: str*) → Optional[str]

Get the zone id of an existing zone by name.

Parameters

- **client** – The connection used to interact with Route53's API.
- **zone_name** – The name of the DNS hosted zone to create.

Returns The Id of the Hosted Zone.

runway.cfngin.utils.**get_or_create_hosted_zone**(*client: Route53Client*, *zone_name: str*) → str

Get the Id of an existing zone, or create it.

Parameters

- **client** – The connection used to interact with Route53's API.
- **zone_name** – The name of the DNS hosted zone to create.

Returns The Id of the Hosted Zone.

class runway.cfngin.utils.**SOARecordText**

Bases: object

Represents the actual body of an SOARecord.

__init__(*record_text: str*) → None

Instantiate class.

__str__() → str

Convert an instance of this class to a string.

__new__(***kwargs*)

class runway.cfngin.utils.**SOARecord**

Bases: object

Represents an SOA record.

__init__(*record: ResourceRecordSetTypeDef*) → None

Instantiate class.

__new__(***kwargs*)

runway.cfngin.utils.**get_soa_record**(*client: Route53Client*, *zone_id: str*, *zone_name: str*) → *SOARecord*

Get the SOA record for zone_name from zone_id.

Parameters

- **client** – The connection used to interact with Route53's API.
- **zone_id** – The AWS Route53 zone id of the hosted zone to query.
- **zone_name** – The name of the DNS hosted zone to create.

Returns An object representing the parsed SOA record returned from AWS Route53.

runway.cfngin.utils.**create_route53_zone**(*client: Route53Client*, *zone_name: str*) → str

Create the given zone_name if it doesn't already exists.

Also sets the SOA negative caching TTL to something short (300 seconds).

Parameters

- **client** – The connection used to interact with Route53's API.
- **zone_name** – The name of the DNS hosted zone to create.

Returns The zone id returned from AWS for the existing, or newly created zone.

runway.cfngin.utils.**yaml_to_ordered_dict**(*stream: str*, *loader: typing.Union[typing.Type[yaml.loader.Loader], typing.Type[yaml.loader.SafeLoader]] = <class 'yaml.loader.SafeLoader'>*) → OrderedDict[str, Any]

yaml.load alternative with preserved dictionary order.

Parameters

- **stream** – YAML string to load.
- **loader** – PyYAML loader class. Defaults to safe load.

runway.cfngin.utils.**uppercase_first_letter**(*string_: str*) → str

Return string with first character upper case.

runway.cfngin.utils.**cf_safe_name**(*name: str*) → str

Convert a name to a safe string for a CloudFormation resource.

Given a string, returns a name that is safe for use as a CloudFormation Resource. (ie: Only alphanumeric characters)

runway.cfngin.utils.**read_value_from_path**(*value: str*, *, *root_path: Optional[pathlib.Path] = None*) → str

Enable translators to read values from files.

The value can be referred to with the *file://* prefix.

Example

```
conf_key: ${kms file://kms_value.txt}
```

runway.cfngin.utils.**get_client_region**(*client: Any*) → str

Get the region from a boto3 client.

Parameters **client** – The client to get the region from.

Returns AWS region string.

runway.cfngin.utils.**get_s3_endpoint**(*client: Any*) → str

Get the s3 endpoint for the given boto3 client.

Parameters **client** – The client to get the endpoint from.

Returns The AWS endpoint for the client.

runway.cfngin.utils.**s3_bucket_location_constraint**(*region: Optional[str]*) → Optional[str]

Return the appropriate LocationConstraint info for a new S3 bucket.

When creating a bucket in a region OTHER than us-east-1, you need to specify a LocationConstraint inside the CreateBucketConfiguration argument. This function helps you determine the right value given a given client.

Parameters **region** – The region where the bucket will be created in.

Returns The string to use with the given client for creating a bucket.

runway.cfngin.utils.**ensure_s3_bucket**(*s3_client: S3Client*, *bucket_name: str*, *bucket_region: Optional[str] = None*, ***, *create: bool = True*, *persist_graph: bool = False*) → None

Ensure an s3 bucket exists, if it does not then create it.

Parameters

- **s3_client** – An s3 client used to verify and create the bucket.
- **bucket_name** – The bucket being checked/created.
- **bucket_region** – The region to create the bucket in. If not provided, will be determined by s3_client's region.
- **create** – Whether to create the bucket if it does not exist.
- **persist_graph** – Check bucket for recommended settings. If creating a new bucket, it will be created with recommended settings.

runway.cfngin.utils.**parse_cloudformation_template**(*template: str*) → Dict[str, Any]

Parse CFN template string.

Leverages the vendored aws-cli yamlhelper to handle JSON or YAML templates.

Parameters **template** – The template body.

runway.cfngin.utils.**is_within_directory**(*directory: Path | str*, *target: str*) → bool

Check if file is in directory.

Determines if the provided path is within a specific directory or its subdirectories.

Parameters

- **directory** (*Union[Path, str]*) – Path of the directory we're checking.
- **target** (*str*) – Path of the file we're checking for containment.

Returns True if the target is in the directory or subdirectories, False otherwise.

Return type bool

runway.cfngin.utils.**safe_tar_extract**(*tar: tarfile.TarFile*, *path: Path | str = '.'*, *members: list[tarfile.TarInfo] | None = None*, ***, *numeric_owner: bool = False*)

Safely extract the contents of a tar file to a specified directory.

This code is modified from a PR provided to Runway project to address CVE-2007-4559.

Parameters

- **tar** (*TarFile*) – The tar file object that will be extracted.
- **path** (*Union[Path, str], optional*) – The directory to extract the tar into.
- **members** (*List[TarInfo] | None, optional*) – List of TarInfo objects to extract.
- **numeric_owner** (*bool, optional*) – Enable usage of owner and group IDs when extracting.

Raises Exception – If any tar file tries to go outside the specified area.

class runway.cfngin.utils.**Extractor**

Bases: object

Base class for extractors.

__init__(*archive: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters **archive** (*str*) – Archive path.

set_archive(*dir_name: pathlib.Path*) → None

Update archive filename to match directory name & extension.

Parameters **dir_name** – Archive directory name

__new__(***kwargs*)

class runway.cfngin.utils.**TarExtractor**

Bases: *runway.cfngin.utils.Extractor*

Extracts tar archives.

extract(*destination: pathlib.Path*) → None

Extract the archive.

__init__(*archive: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters **archive** (*str*) – Archive path.

__new__(***kwargs*)

set_archive(*dir_name: pathlib.Path*) → None

Update archive filename to match directory name & extension.

Parameters **dir_name** – Archive directory name

class runway.cfngin.utils.**TarGzipExtractor**

Bases: *runway.cfngin.utils.Extractor*

Extracts compressed tar archives.

extract(*destination: pathlib.Path*) → None

Extract the archive.

__init__(*archive: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters **archive** (*str*) – Archive path.

__new__(***kwargs*)

set_archive(*dir_name: pathlib.Path*) → None

Update archive filename to match directory name & extension.

Parameters **dir_name** – Archive directory name

class runway.cfngin.utils.**ZipExtractor**

Bases: *runway.cfngin.utils.Extractor*

Extracts zip archives.

extract(*destination: pathlib.Path*) → None

Extract the archive.

__init__(*archive: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters **archive** (*str*) – Archive path.

__new__(***kwargs*)

set_archive(*dir_name: pathlib.Path*) → None

Update archive filename to match directory name & extension.

Parameters **dir_name** – Archive directory name

class runway.cfngin.utils.**SourceProcessor**

Bases: object

Makes remote python package sources available in current environment.

__init__(*sources:* CfnginPackageSourcesDefinitionModel, *cache_dir: Path*) → None

Process a config's defined package sources.

Parameters

- **sources** – Package sources from CFNgin config dictionary.
- **cache_dir** – Path where remote sources will be cached.

create_cache_directories() → None

Ensure that SourceProcessor cache directories exist.

get_package_sources() → None

Make remote python packages available for local use.

fetch_local_package(*config:* LocalCfnginPackageSourceDefinitionModel) → None

Make a local path available to current CFNgin config.

Parameters **config** – Package source config.

fetch_s3_package(*config:* S3CfnginPackageSourceDefinitionModel) → None

Make a remote S3 archive available for local use.

Parameters **config** – Package source config.

fetch_git_package(*config:* GitCfnginPackageSourceDefinitionModel) → None

Make a remote git repository available for local use.

Parameters **config** – Package source config.

update_paths_and_config(*config: Union[*GitCfnginPackageSourceDefinitionModel, LocalCfnginPackageSourceDefinitionModel, S3CfnginPackageSourceDefinitionModel*]*, *pkg_dir_name: str*, *pkg_cache_dir: Optional[Path] = None*) → None

Handle remote source defined sys.paths & configs.

Parameters

- **config** – Package source config.
- **pkg_dir_name** – Directory name of the CFNgin archive.
- **pkg_cache_dir** – Fully qualified path to CFNgin cache cache directory.

static git_ls_remote(*uri: str*, *ref: str*) → str

Determine the latest commit id for a given ref.

Parameters

- **uri** – Git URI.
- **ref** – Git ref.

static determine_git_ls_remote_ref(*config:* GitCfnginPackageSourceDefinitionModel) → str

Determine the ref to be used with the "git ls-remote" command.

Parameters **config** – Git package source config.

Returns A branch reference or "HEAD".

determine_git_ref(*config:* GitCfnginPackageSourceDefinitionModel) → str

Determine the ref to be used for `git checkout`.

Parameters **config** – Git package source config.

Returns A commit id or tag name.

static sanitize_uri_path(*uri: str*) → str

Take a URI and converts it to a directory safe path.

Parameters **uri** – URI to sanitize.

Returns Directory name for the supplied uri.

sanitize_git_path(*uri: str*, *ref: Optional[str] = None*) → str

Take a git URI and ref and converts it to a directory safe path.

Parameters

- **uri** – Git URI. (e.g. `git@github.com:foo/bar.git`)
- **ref** – Git ref to be appended to the path.

Returns Directory name for the supplied uri

__new__(***kwargs*)

runway.cfngin.utils.**stack_template_key_name**(*blueprint:* Blueprint) → str

Given a blueprint, produce an appropriate key name.

Parameters **blueprint** – The blueprint object to create the key from.

Returns Key name resulting from blueprint.

runway.config package

CFNgin config.

class runway.config.**BaseConfig**

Bases: `object`

Base class for configurations.

__init__(*data:* BaseModel, *, *path: Optional[Path] = None*) → None

Instantiate class.

Parameters

- **data** – The data model of the config file.
- **path** – Path to the config file.

dump(*, *by_alias: bool = False*, *exclude: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *exclude_unset: bool = True*, *include: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None*) → str

Dump model to a YAML string.

Parameters

- **by_alias** – Whether field aliases should be used as keys in the returned dictionary.
- **exclude** – Fields to exclude from the returned dictionary.
- **exclude_defaults** – Whether fields which are equal to their default values (whether set or otherwise) should be excluded from the returned dictionary.
- **exclude_none** – Whether fields which are equal to None should be excluded from the returned dictionary.
- **exclude_unset** – Whether fields which were not explicitly set when creating the model should be excluded from the returned dictionary.
- **include** – Fields to include in the returned dictionary.

classmethod find_config_file(*path: pathlib.Path*) → Optional[pathlib.Path]

Find a config file in the provided path.

Parameters **path** – The path to search for a config file.

__new__(***kwargs*)

class runway.config.**CfnginConfig**

Bases: *runway.config.BaseConfig*

Python representation of a CFNgin config file.

This is used internally by CFNgin to parse and validate a YAML formatted CFNgin configuration file, but can also be used in scripts to generate a CFNgin config file before handing it off to CFNgin to deploy/destroy.

Example:

```
from runway.cfngin.config import dump, Config, Stack

vpc = Stack({
    "name": "vpc",
    "class_path": "blueprints.VPC"})

config = Config()
config.namespace = "prod"
config.stacks = [vpc]

print dump(config)
```

EXCLUDE_REGEX = 'runway(\\..*)?\\.(yml|yaml)'

Regex for file names to exclude when looking for config files.

EXCLUDE_LIST = ['bitbucket-pipelines.yml', 'buildspec.yml', 'docker-compose.yml']

Explicit files names to ignore when looking for config files.

__new__(***kwargs*)

dump(**, by_alias: bool = False, exclude: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None, exclude_defaults: bool = False, exclude_none: bool = False, exclude_unset: bool = True, include: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None*) → str

Dump model to a YAML string.

Parameters

- **by_alias** – Whether field aliases should be used as keys in the returned dictionary.
- **exclude** – Fields to exclude from the returned dictionary.
- **exclude_defaults** – Whether fields which are equal to their default values (whether set or otherwise) should be excluded from the returned dictionary.
- **exclude_none** – Whether fields which are equal to None should be excluded from the returned dictionary.
- **exclude_unset** – Whether fields which were not explicitly set when creating the model should be excluded from the returned dictionary.
- **include** – Fields to include in the returned dictionary.

__init__(*data:* runway.config.models.cfngin.CfnginConfigDefinitionModel, **, path: Optional[pathlib.Path] = None, work_dir: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters

- **data** – The data model of the config file.
- **path** – Path to the config file.
- **work_dir** – Working directory.

cfngin_bucket: Optional[str]

Bucket to use for CFNgin resources. (e.g. CloudFormation templates). May be an empty string.

cfngin_bucket_region: Optional[str]

Explicit region to use for *CfnginConfig.cfngin_bucket*

cfngin_cache_dir: Path

Local directory to use for caching.

log_formats: Dict[str, str]

Custom formatting for log messages.

lookups: Dict[str, str]

Register custom lookups.

mappings: Dict[str, Dict[str, Dict[str, Any]]]

Mappings that will be added to all stacks.

namespace: *str*

Namespace to prepend to everything.

namespace_delimiter: *str*

Character used to separate *CfnginConfig.namespace* and anything it prepends.

package_sources: *CfnginPackageSourcesDefinitionModel*

Remote source locations.

persistent_graph_key: *Optional[str] = None*

S3 object key were the persistent graph is stored.

post_deploy: *List[CfnginHookDefinitionModel]*

Hooks to run after a deploy action.

post_destroy: *List[CfnginHookDefinitionModel]*

Hooks to run after a destroy action.

pre_deploy: *List[CfnginHookDefinitionModel]*

Hooks to run before a deploy action.

pre_destroy: *List[CfnginHookDefinitionModel]*

Hooks to run before a destroy action.

service_role: *Optional[str]*

IAM role for CloudFormation to use.

stacks: *List[CfnginStackDefinitionModel]*

Stacks to be processed.

sys_path: *Optional[Path]*

Relative or absolute path to use as the work directory.

tags: *Optional[Dict[str, str]]*

Tags to apply to all resources.

template_indent: *int*

Spaces to use per-indent level when outputting a template to json.

load() → None

Load config options into the current environment/session.

classmethod find_config_file(*path: Optional[pathlib.Path] = None*, *, *exclude: Optional[List[str]] = None*) → List[pathlib.Path]

Find a config file in the provided path.

Parameters

- **path** – The path to search for a config file.
- **exclude** – List of file names to exclude. This list is appended to the global exclude list.

Raises

- *ConfigNotFound* – Could not find a config file in the provided path.
- **ValueError** – More than one config file found in the provided path.

classmethod parse_file(*, *path: Optional[pathlib.Path] = None*, *file_path: Optional[pathlib.Path] = None*, *parameters: Optional[MutableMapping[str, Any]] = None*, *work_dir: Optional[pathlib.Path] = None*, ***kwargs: Any*) → *runway.config.CfnginConfig*

Parse a YAML file to create a config object.

Parameters

- **path** – The path to search for a config file.
- **file_path** – Exact path to a file to parse.
- **parameters** – Values to use when resolving a raw config.
- **work_dir** – Explicit working directory.

Raises *ConfigNotFound* – Provided config file was not found.

classmethod parse_obj(*obj: Any*, ***, *path: Optional[pathlib.Path] = None*, *work_dir: Optional[pathlib.Path] = None*) → *runway.config.CfnginConfig*

Parse a python object.

Parameters

- **obj** – A python object to parse as a CFNgin config.
- **path** – The path to the config file that was parsed into the object.
- **work_dir** – Working directory.

classmethod parse_raw(*data: str*, ***, *parameters: Optional[MutableMapping[str, Any]] = None*, *path: Optional[pathlib.Path] = None*, *skip_package_sources: bool = False*, *work_dir: Optional[pathlib.Path] = None*) → *runway.config.CfnginConfig*

Parse raw data.

Parameters

- **data** – The raw data to parse.
- **parameters** – Values to use when resolving a raw config.
- **path** – The path to search for a config file.
- **skip_package_sources** – Skip processing package sources.
- **work_dir** – Explicit working directory.

classmethod process_package_sources(*raw_data: str*, ***, *parameters: Optional[MutableMapping[str, Any]] = None*, *work_dir: Optional[pathlib.Path] = None*) → str

Process the package sources defined in a rendered config.

Parameters

- **raw_data** – Raw configuration data.
- **cache_dir** – Directory to use when caching remote sources.
- **parameters** – Values to use when resolving a raw config.
- **work_dir** – Explicit working directory.

static resolve_raw_data(*raw_data: str*, ***, *parameters: Optional[MutableMapping[str, Any]] = None*) → str

Resolve raw data.

Parameters

- **raw_data** – Raw configuration data.
- **parameters** – Values to use when resolving a raw config.

Raises *MissingEnvironment* – A value required by the config was not provided in parameters.

class runway.config.**RunwayConfig**

Bases: *runway.config.BaseConfig*

Python representation of a Runway config file.

__new__(***kwargs*)

dump(**, by_alias: bool = False, exclude: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None, exclude_defaults: bool = False, exclude_none: bool = False, exclude_unset: bool = True, include: Optional[Union[AbstractSet[Union[int, str]], Mapping[Union[int, str], Any]]] = None*) → str

Dump model to a YAML string.

Parameters

- **by_alias** – Whether field aliases should be used as keys in the returned dictionary.
- **exclude** – Fields to exclude from the returned dictionary.
- **exclude_defaults** – Whether fields which are equal to their default values (whether set or otherwise) should be excluded from the returned dictionary.
- **exclude_none** – Whether fields which are equal to None should be excluded from the returned dictionary.
- **exclude_unset** – Whether fields which were not explicitly set when creating the model should be excluded from the returned dictionary.
- **include** – Fields to include in the returned dictionary.

__init__(*data:* runway.config.models.runway.RunwayConfigDefinitionModel, **, path: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters

- **data** – The data model of the config file.
- **path** – Path to the config file.

classmethod find_config_file(*path: pathlib.Path*) → pathlib.Path

Find a config file in the provided path.

Parameters **path** – The path to search for a config file.

Raises

- *ConfigNotFound* – Could not find a config file in the provided path.
- **ValueError** – More than one config file found in the provided path.

classmethod parse_file(**, path: Optional[pathlib.Path] = None, file_path: Optional[pathlib.Path] = None, **kwargs: Any*) → *runway.config.RunwayConfig*

Parse a YAML file to create a config object.

Parameters

- **path** – The path to search for a config file.
- **file_path** – Exact path to a file to parse.

Raises

- *ConfigNotFound* – Provided config file was not found.
- ValueError – path and file_path were both excluded.

classmethod parse_obj(*obj: Any*, *, *path: Optional[pathlib.Path] = None*) → *runway.config.RunwayConfig*

Parse a python object into a config object.

Parameters

- **obj** – The object to be parsed.
- **path** – Path to the file the object was parsed from.

Subpackages

runway.config.components package

Runway & CFNgin config components.

Subpackages

runway.config.components.runway package

Runway config components.

class runway.config.components.runway.**CfnLintRunwayTestDefinition**

Bases: *runway.config.components.runway._test_def.RunwayTestDefinition*[*runway.config.models.runway._builtin_tests.CfnLintRunwayTestDefinitionModel*]

Runway cfn-lint test definition.

__init__(*data:* runway.config.models.runway._builtin_tests.CfnLintRunwayTestDefinitionModel) → None

Instantiate class.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._test_def.CfnLintRunwayTestDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- AttributeError – Object does not contain an attribute for the name provided.

- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

static __new__(*cls*, *data: runway.config.components.runway._test_def._DataModel*) → *runway.config.components.runway._test_def.RunwayTestDefinition*[runway.config.components.runway._test_def._

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, *pre_process: bool = False*, *variables: Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

class runway.config.components.runway.**RunwayDeploymentDefinition**

Bases: *runway.config.components.runway.base.ConfigComponentDefinition*

Runway deployment definition.

__init__(*data:* runway.config.models.runway.RunwayDeploymentDefinitionModel) → None

Instantiate class.

property menu_entry: str

Return menu entry representation of this deployment.

property modules:
List[*runway.config.components.runway._module_def.RunwayModuleDefinition*]

List of Runway modules.

reverse()

Reverse the order of modules and regions.

set_modules(*modules:*
*List[Union[*runway.config.components.runway._module_def.RunwayModuleDefinition, runway.config.models.runway.RunwayModuleDefinitionModel*]]*) → None

Set the value of modules.

Parameters **modules** – A list of modules.

Raises **TypeError** – The provided value does not match the required types.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

__new__(***kwargs*)

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises AttributeError – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, *pre_process: bool = False*, *variables:* *Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

classmethod parse_obj(*obj: List[Dict[str, Any]]*) → List[*runway.config.components.runway._deployment_def.RunwayDeploymentDefinition*]

classmethod parse_obj(*obj: Union[List[*ConfigProperty*], Set[*ConfigProperty*], Tuple[*ConfigProperty*, ...]]*) → List[*runway.config.components.runway._deployment_def.RunwayDeploymentDefinition*]

classmethod parse_obj(*obj: Union[Dict[str, Any],* ConfigProperty*]*) → *runway.config.components.runway._deployment_def.RunwayDeploymentDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

class runway.config.components.runway.**RunwayModuleDefinition**

Bases: *runway.config.components.runway.base.ConfigComponentDefinition*

Runway module definition.

__init__(*data:* runway.config.models.runway.RunwayModuleDefinitionModel) → None

Instantiate class.

property child_modules: List[*runway.config.components.runway._module_def.RunwayModuleDefinition*]

List of child modules.

property is_parent: bool

Assess if the modules contains child modules (e.g. run in parallel).

property menu_entry: str

Return menu entry representation of this module.

reverse()

Reverse the order of child/parallel modules.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

__new__(***kwargs*)

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._module_def.RunwayModuleDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

resolve(*context: RunwayContext*, ***, *pre_process: bool = False*, *variables: Optional[RunwayVariablesDefinition] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

class runway.config.components.runway.**RunwayTestDefinition**

Bases: `Generic`[runway.config.components.runway._test_def._DataModel], *runway.config.components.runway.base.ConfigComponentDefinition*

Runway test definition.

__init__(*data: runway.config.components.runway._test_def._DataModel*) → None

Instantiate class.

static __new__(*cls*, *data: runway.config.components.runway._test_def._DataModel*) → *runway.config.components.runway._test_def.RunwayTestDefinition*[runway.config.components.runway._test_def._

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._test_def.RunwayTestDefinition*[runway.config.components.runway._

Parse a python object into this class.

Parameters **obj** – The object to parse.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, *pre_process: bool = False*, *variables:* *Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

class runway.config.components.runway.**RunwayVariablesDefinition**

Bases: *runway.utils.MutableMap*

Runway variables definition.

__init__(*data:* runway.config.models.runway.RunwayVariablesDefinitionModel) → None

Instantiate class.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._variables_def.RunwayVariablesDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

__bool__() → bool

Implement evaluation of instances as a bool.

__contains__(*value: Any*) → bool

Implement evaluation of 'in' conditional.

__delitem__(*key: str*) → None

Implement deletion of self[key].

Parameters **key** – Attribute name to remove from the object.

Example

__getitem__(*key: str*) → Any

Implement evaluation of self[key].

Parameters **key** – Attribute name to return the value for.

Returns The value associated with the provided key/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

Example

__iter__() → Iterator[Any]

Return iterator object that can iterate over all attributes.

Example

__len__() → int

Implement the built-in function len().

Example

__new__(***kwargs*)

__setitem__(*key: str*, *value: Any*) → None

Implement assignment to self[key].

Parameters

- **key** – Attribute name to associate with a value.
- **value** – Value of a key/attribute.

Example

__str__() → str

Return string representation of the object.

clear() → None. Remove all items from D.

clear_found_cache() → None

Clear _found_cache.

property data: Dict[str, Any]

Sanitized output of __dict__.

Removes anything that starts with _.

find(*query: str*, *default: Optional[Any] = None*, *ignore_cache: bool = False*) → Any

Find a value in the map.

Previously found queries are cached to increase search speed. The cached value should only be used if values are not expected to change.

Parameters

- **query** – A period delimited string that is split to search for nested values
- **default** – The value to return if the query was unsuccessful.
- **ignore_cache** – Ignore cached value.

get(*key: str*, *default: Optional[Any] = None*) → Any

Implement evaluation of self.get.

Parameters

- **key** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class runway.config.components.runway.**ScriptRunwayTestDefinition**

Bases: *runway.config.components.runway._test_def.RunwayTestDefinition*[*runway.config.models.runway._builtin_tests.ScriptRunwayTestDefinitionModel*]

Runway script test definition.

__init__(*data:* runway.config.models.runway._builtin_tests.ScriptRunwayTestDefinitionModel) → None

Instantiate class.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._test_def.ScriptRunwayTestDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

static __new__(*cls*, *data: runway.config.components.runway._test_def._DataModel*) → *runway.config.components.runway._test_def.RunwayTestDefinition*[runway.config.components.runway._test_def.

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, pre_process: bool = False*, *variables:* *Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

class runway.config.components.runway.**YamlLintRunwayTestDefinition**

Bases: *runway.config.components.runway._test_def.RunwayTestDefinition*[*runway.config.models.runway._builtin_tests.YamlLintRunwayTestDefinitionModel*]

Runway yamllint test definition.

__init__(*data:* runway.config.models.runway._builtin_tests.YamlLintRunwayTestDefinitionModel) → None

Instantiate class.

classmethod parse_obj(*obj: Any*) → *runway.config.components.runway._test_def.YamlLintRunwayTestDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

static __new__(*cls*, *data: runway.config.components.runway._test_def._DataModel*) → *runway.config.components.runway._test_def.RunwayTestDefinition*[runway.config.components.runway._test_def.

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, *pre_process: bool = False*, *variables: Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

Submodules

runway.config.components.runway.base module

Runway config base definition.

class runway.config.components.runway.base.**ConfigComponentDefinition**

Bases: abc.ABC

Base class for Runway config components.

__init__(*data:* ConfigProperty) → None

Instantiate class.

property data: Dict[str, Any]

Return the underlying data as a dict.

get(*name: str*, *default: Optional[Any] = None*) → None

Get a value or return default if it is not found.

Parameters

- **name** – The value to look for.
- **default** – Returned if no other value is found.

resolve(*context:* RunwayContext, *, *pre_process: bool = False*, *variables:* *Optional[*RunwayVariablesDefinition*] = None*) → None

Resolve variables.

Parameters

- **context** – Runway context object.
- **pre_process** – Whether to only resolve pre-process fields.
- **variables** – Object containing values to resolve the `var` lookup.

abstract classmethod parse_obj(*obj: Any*) → *runway.config.components.runway.base.ConfigComponentDefinition*

Parse a python object into this class.

Parameters **obj** – The object to parse.

__contains__(*name: str*) → bool

Implement evaluation of 'in' conditional.

__getattr__(*name: str*)

Implement evaluation of self.name.

Parameters **name** – The value to look for.

Raises

- **AttributeError** – Object does not contain an attribute for the name provided.
- ***UnresolvedVariable*** – The value being access is a variable and it has not been resolved yet.

__getitem__(*name: str*)

Implement evaluation of self[name].

Parameters **name** – The value to look for.

Raises **KeyError** – Object does not contain a field of the name provided.

__new__(***kwargs*)

__setattr__(*name: str*, *value: Any*) → None

Implement evaluation of self.name = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

Raises **AttributeError** – The name being set is a property without a setter.

__setitem__(*name: str*, *value: Any*) → None

Implement evaluation of self[name] = value.

When setting an attribute, the value is set on the underlying data model. The exception to this is if the name starts with an underscore.

Parameters

- **name** – The value to set.
- **value** – The value to assigned to the field.

runway.config.models package

Runway & CFNgin config models.

Subpackages

runway.config.models.cfngin package

CFNgin config models.

class runway.config.models.cfngin.**CfnginConfigDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin config definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

classmethod parse_file(*path: Union[str, Path]*, *, *content_type: Optional[str] = None*, *encoding: str = 'utf8'*, *proto: Optional[Protocol] = None*, *allow_pickle: bool = False*) → Model

Parse a file.

classmethod parse_raw(*b: Union[bytes, str]*, *, *content_type: Optional[str] = None*, *encoding: str = 'utf8'*, *proto: Optional[Protocol] = None*, *allow_pickle: bool = False*) → Model

Parse raw data.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**CfnginHookDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin hook definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(**kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

object_pairs_hook is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**CfnginPackageSourcesDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin package sources definition.

git

Package source located in a git repo.

Type List[*runway.config.models.cfngin._package_sources.GitCfnginPackageSourceDefinitionModel*]

local

Package source located on a local disk.

Type List[*runway.config.models.cfngin._package_sources.LocalCfnginPackageSourceDefinitionModel*]

s3

Package source located in AWS S3.

Type List[*runway.config.models.cfngin._package_sources.S3CfnginPackageSourceDefinitionModel*]

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**CfnginStackDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin stack definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration options.

static schema_extra(*schema: Dict[str, Any]*) → None

Process the schema after it has been generated.

Schema is modified in place. Return value is ignored.

https://pydantic-docs.helpmanual.io/usage/schema/#schema-customization

__init__()

__new__(**kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**GitCfnginPackageSourceDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a git package source definition.

Package source located in a git repository.

branch

Branch name.

Type Optional[str]

commit

Commit hash.

Type Optional[str]

configs

List of CFNgin config paths to execute.

Type List[str]

paths

List of paths to append to `sys.path`.

Type List[str]

tag

Git tag.

Type Optional[str]

uri

Remote git repo URI.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(**kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**LocalCfnginPackageSourceDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin local package source definition.

Package source located on a local disk.

configs

List of CFNgin config paths to execute.

Type List[str]

paths

List of paths to append to `sys.path`.

Type List[str]

source

Source.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, **kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

object_pairs_hook is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

parse_float, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

parse_int, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

parse_constant, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom JSONDecoder subclass, specify it with the cls kwarg; otherwise JSONDecoder is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. self[name] = value).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.cfngin.**S3CfnginPackageSourceDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a CFNgin S3 package source definition.

Package source located in AWS S3.

bucket

AWS S3 bucket name.

Type str

configs

List of CFNgin config paths to execute.

Type List[str]

key

Object key. The object should be a zip file.

Type str

paths

List of paths to append to `sys.path`.

Type List[str]

requester_pays

AWS S3 requester pays option.

Type bool

use_latest

Use the latest version of the object.

Type bool

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.config.models.runway package

Runway config models.

class runway.config.models.runway.**CfnLintRunwayTestArgs**

Bases: *runway.config.models.base.ConfigProperty*

Model for the args of a cfn-lint test.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**CfnLintRunwayTestDefinitionModel**

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Model for a cfn-lint test definition.

class Config

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

static __new__(*cls*, ***kwargs: Any*) → *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayAssumeRoleDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway assume role definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayConfigDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Runway configuration definition model.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(**kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

classmethod parse_file(*path: Union[str, Path]*, *, *content_type: Optional[str] = None*, *encoding: str = 'utf8'*, *proto: Optional[Protocol] = None*, *allow_pickle: bool = False*) → Model

Parse a file.

classmethod parse_raw(*b: Union[bytes, str]*, *, *content_type: Optional[str] = None*, *encoding: str = 'utf8'*, *proto: Optional[Protocol] = None*, *allow_pickle: bool = False*) → Model

Parse raw data.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayDeploymentDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway deployment definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

static schema_extra(*schema: Dict[str, Any]*) → None

Process the schema after it has been generated.

Schema is modified in place. Return value is ignored.

https://pydantic-docs.helpmanual.io/usage/schema/#schema-customization

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayDeploymentRegionDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway deployment region definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayFutureDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for the Runway future definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(**, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayModuleDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway module definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayTestDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway test definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(**, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

static __new__(*cls*, ***kwargs: Any*) → *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayVariablesDefinitionModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for a Runway variable definition.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: *Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**RunwayVersionField**

Bases: packaging.specifiers.SpecifierSet

Extends packaging.specifiers.SpecifierSet for use with pydantic.

classmethod __get_validators__() → Generator[Callable[[...], Any], None, None]

Yield one of more validators with will be called to validate the input.

Each validator will receive, as input, the value returned from the previous validator.

classmethod __modify_schema__(*field_schema: Dict[str, Any]*) → None

Mutate the field schema in place.

This is only called when output JSON schema from a model.

__and__(*other: Union[packaging.specifiers.SpecifierSet, str]*) → packaging.specifiers.SpecifierSet

Return a SpecifierSet which is a combination of the two sets.

Parameters **other** – The other object to combine with.

```
>>> SpecifierSet(">=1.0.0,!=1.0.1") & '<=2.0.0,!=2.0.1'
<SpecifierSet('!=1.0.1,!=2.0.1,<=2.0.0,>=1.0.0')>
>>> SpecifierSet(">=1.0.0,!=1.0.1") & SpecifierSet('<=2.0.0,!=2.0.1')
<SpecifierSet('!=1.0.1,!=2.0.1,<=2.0.0,>=1.0.0')>
```

__contains__(*item: Union[packaging.version.Version, str]*) → bool

Return whether or not the item is contained in this specifier.

Parameters **item** – The item to check for.

This is used for the `in` operator and behaves the same as *contains()* with no `prereleases` argument passed.

```
>>> "1.2.3" in SpecifierSet(">=1.0.0,!=1.0.1")
True
>>> Version("1.2.3") in SpecifierSet(">=1.0.0,!=1.0.1")
True
>>> "1.0.1" in SpecifierSet(">=1.0.0,!=1.0.1")
False
>>> "1.3.0a1" in SpecifierSet(">=1.0.0,!=1.0.1")
False
>>> "1.3.0a1" in SpecifierSet(">=1.0.0,!=1.0.1", prereleases=True)
True
```

__eq__(*other: object*) → bool

Whether or not the two SpecifierSet-like objects are equal.

Parameters **other** – The other object to check against.

The value of *prereleases* is ignored.

```
>>> SpecifierSet(">=1.0.0,!=1.0.1") == SpecifierSet(">=1.0.0,!=1.0.1")
True
>>> (SpecifierSet(">=1.0.0,!=1.0.1", prereleases=False) ==
...  SpecifierSet(">=1.0.0,!=1.0.1", prereleases=True))
```

(continues on next page)

(continued from previous page)

```
True
>>> SpecifierSet(">=1.0.0,!=1.0.1") == ">=1.0.0,!=1.0.1"
True
>>> SpecifierSet(">=1.0.0,!=1.0.1") == SpecifierSet(">=1.0.0")
False
>>> SpecifierSet(">=1.0.0,!=1.0.1") == SpecifierSet(">=1.0.0,!=1.0.2")
False
```

__init__(*specifiers: str = ''*, *prereleases: Optional[bool] = None*) → None

Initialize a SpecifierSet instance.

Parameters

- **specifiers** – The string representation of a specifier or a comma-separated list of specifiers which will be parsed and normalized before use.
- **prereleases** – This tells the SpecifierSet if it should accept prerelease versions if applicable or not. The default of `None` will autodetect it from the given specifiers.

Raises **InvalidSpecifier** – If the given `specifiers` are not parseable than this exception will be raised.

__iter__() → Iterator[packaging.specifiers.Specifier]

Returns an iterator over all the underlying `Specifier` instances in this specifier set.

```
>>> sorted(SpecifierSet(">=1.0.0,!=1.0.1"), key=str)
[<Specifier('!=1.0.1')>, <Specifier('>=1.0.0')>]
```

__len__() → int

Returns the number of specifiers in this specifier set.

__new__(***kwargs*)

__repr__() → str

A representation of the specifier set that shows all internal state.

Note that the ordering of the individual specifiers within the set may not match the input string.

```
>>> SpecifierSet('>=1.0.0,!=2.0.0')
<SpecifierSet('!=2.0.0,>=1.0.0')>
>>> SpecifierSet('>=1.0.0,!=2.0.0', prereleases=False)
<SpecifierSet('!=2.0.0,>=1.0.0', prereleases=False)>
>>> SpecifierSet('>=1.0.0,!=2.0.0', prereleases=True)
<SpecifierSet('!=2.0.0,>=1.0.0', prereleases=True)>
```

__str__() → str

A string representation of the specifier set that can be round-tripped.

Note that the ordering of the individual specifiers within the set may not match the input string.

```
>>> str(SpecifierSet(">=1.0.0,!=1.0.1"))
'!=1.0.1,>=1.0.0'
>>> str(SpecifierSet(">=1.0.0,!=1.0.1", prereleases=False))
'!=1.0.1,>=1.0.0'
```

contains(*item: Union[packaging.version.Version, str]*, *prereleases: Optional[bool] = None*, *installed: Optional[bool] = None*) → bool

Return whether or not the item is contained in this SpecifierSet.

Parameters

- **item** – The item to check for, which can be a version string or a `Version` instance.
- **prereleases** – Whether or not to match prereleases with this SpecifierSet. If set to `None` (the default), it uses *prereleases* to determine whether or not prereleases are allowed.

```
>>> SpecifierSet(">=1.0.0,!=1.0.1").contains("1.2.3")
True
>>> SpecifierSet(">=1.0.0,!=1.0.1").contains(Version("1.2.3"))
True
>>> SpecifierSet(">=1.0.0,!=1.0.1").contains("1.0.1")
False
>>> SpecifierSet(">=1.0.0,!=1.0.1").contains("1.3.0a1")
False
>>> SpecifierSet(">=1.0.0,!=1.0.1", prereleases=True).contains("1.3.0a1")
True
>>> SpecifierSet(">=1.0.0,!=1.0.1").contains("1.3.0a1", prereleases=True)
True
```

filter(*iterable: Iterable[packaging.specifiers.UnparsedVersionVar]*, *prereleases: Optional[bool] = None*) → Iterator[packaging.specifiers.UnparsedVersionVar]

Filter items in the given iterable, that match the specifiers in this set.

Parameters

- **iterable** – An iterable that can contain version strings and `Version` instances. The items in the iterable will be filtered according to the specifier.
- **prereleases** – Whether or not to allow prereleases in the returned iterator. If set to `None` (the default), it will be intelligently decide whether to allow prereleases or not (based on the *prereleases* attribute, and whether the only versions matching are prereleases).

This method is smarter than just `filter(SpecifierSet(...).contains, [...])` because it implements the rule from **PEP 440** that a prerelease item SHOULD be accepted if no other versions match the given specifier.

```
>>> list(SpecifierSet(">=1.2.3").filter(["1.2", "1.3", "1.5a1"]))
['1.3']
>>> list(SpecifierSet(">=1.2.3").filter(["1.2", "1.3", Version("1.4")]))
['1.3', <Version('1.4')>]
>>> list(SpecifierSet(">=1.2.3").filter(["1.2", "1.5a1"]))
[]
>>> list(SpecifierSet(">=1.2.3").filter(["1.3", "1.5a1"], prereleases=True))
['1.3', '1.5a1']
>>> list(SpecifierSet(">=1.2.3", prereleases=True).filter(["1.3", "1.5a1"]))
['1.3', '1.5a1']
```

An "empty" SpecifierSet will filter items based on the presence of prerelease versions in the set.

```
>>> list(SpecifierSet("").filter(["1.3", "1.5a1"]))
['1.3']
>>> list(SpecifierSet("").filter(["1.5a1"]))
```

(continues on next page)

(continued from previous page)

```
['1.5a1']
>>> list(SpecifierSet("", prereleases=True).filter(["1.3", "1.5a1"]))
['1.3', '1.5a1']
>>> list(SpecifierSet("").filter(["1.3", "1.5a1"], prereleases=True))
['1.3', '1.5a1']
```

property prereleases: Optional[bool]

Whether or not pre-releases as a whole are allowed.

This can be set to either `True` or `False` to explicitly enable or disable prereleases or it can be set to `None` (the default) to use default semantics.

class runway.config.models.runway.ScriptRunwayTestArgs

Bases: *runway.config.models.base.ConfigProperty*

Model for the args of a script test.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**ScriptRunwayTestDefinitionModel**

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Model for a script test definition.

class Config

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

static __new__(*cls*, ***kwargs: Any*) → *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.**YamlLintRunwayTestDefinitionModel**

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Model for a yamllint test definition.

class Config

Bases: *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

static __new__(*cls*, ***kwargs: Any*) → *runway.config.models.runway._builtin_tests.RunwayTestDefinitionModel*

Create a new instance of a class.

Returns Correct subclass of RunwayTestDefinition for the given data.

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

Subpackages

runway.config.models.runway.options package

Runway module options.

Submodules

runway.config.models.runway.options.cdk module

Runway AWS Cloud Development Kit Module options.

class runway.config.models.runway.options.cdk.**RunwayCdkModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway AWS Cloud Development Kit Module options.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.config.models.runway.options.k8s module

Runway Kubernetes Module options.

class runway.config.models.runway.options.k8s.**RunwayK8sModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Kubernetes Module options.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises AttributeError – If attribute does not exist on this object.

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(**kwargs*)

__pretty__(*fmt: Callable[[Any], Any], **kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(**localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, **values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.config.models.runway.options.serverless module

Runway Serverless Framework Module options.

class
runway.config.models.runway.options.serverless.**RunwayServerlessPromotezipOptionDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Serverless module promotezip option.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__bool__() → bool

Evaluate the boolean value of the object instance.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.options.serverless.**RunwayServerlessModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Serverless Framework Module options.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.config.models.runway.options.terraform module

Runway Terraform Module options.

class runway.config.models.runway.options.terraform.**RunwayTerraformArgsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Terraform Module args option.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.options.terraform.**RunwayTerraformBackendConfigDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Terraform Module terraform_backend_config option.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__bool__() → bool

Evaluate the boolean value of the object instance.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.config.models.runway.options.terraform.**RunwayTerraformModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway Terraform Module options.

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

Submodules

runway.config.models.base module

Base models & other objects.

class runway.config.models.base.**ConfigProperty**

Bases: *runway.utils.BaseModel*

Base class for Runway configuration properties.

class Config

Bases: pydantic.config.BaseConfig

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.config.models.utils module

Runway & CFNgin config model utilities.

runway.config.models.utils.**convert_null_values**(*v: Any*) → Any

Convert a "null" string into type(None).

runway.config.models.utils.**resolve_path_field**(*v: Optional[pathlib.Path]*) → Optional[pathlib.Path]

Resolve sys_path.

runway.config.models.utils.**validate_string_is_lookup**(*v: Any*) → Any

Validate value against regex if it's a string to ensure its a lookup.

runway.context package

Context objects.

class runway.context.**CfnginContext**

Bases: runway.context._base.BaseContext

CFNgin context object.

bucket_region

Region where the S3 Bucket is located. The S3 Bucket being the Bucket configured for staging CloudFormation Templates.

Type str

config

CFNgin configuration file that has been resolved & parsed into a python object.

Type *CfnginConfig*

config_path

Path to the configuration file that has been resolved, parsed and made accessible via this object.

Type Path

env

Deploy environment object containing information about the current deploy environment.

Type *DeployEnvironment*

force_stacks

List of stacks to force.

Type List[str]

hook_data

Values returned by hooks that are stored based on the `data_key` defined for the hook. Returned values are only stored if a `data_key` was provided AND the return value is a dict or `pydantic.BaseModel`.

Type Dict[str, Any]

logger

Custom logger to use when logging messages.

Type Union[PrefixAdaptor, RunwayLogger]

parameters

Parameters passed from Runway or read from a file.

Type MutableMapping[str, Any]

stack_names

List of Stack names to operate on. If value is falsy, all Stacks defined in the config will be operated on.

Type List[str]

__init__(*, *config: typing.Optional[runway.config.CfnginConfig] = None*, *config_path: typing.Optional[pathlib.Path] = None*, *deploy_environment: typing.Optional[runway.core.components._deploy_environment.DeployEnvironment] = None*, *force_stacks: typing.Optional[typing.List[str]] = None*, *logger: typing.Union[runway._logging.PrefixAdaptor, runway._logging.RunwayLogger] = <RunwayLogger runway.context._cfngin (WARNING)>*, *parameters: typing.Optional[typing.MutableMapping[str, typing.Any]] = None*, *stack_names: typing.Optional[typing.List[str]] = None*, *work_dir: typing.Optional[pathlib.Path] = None*, ***_: typing.Any*) → None

Instantiate class.

Parameters

- **config** – The CFNgin configuration being operated on.
- **config_path** – Path to the config file that was provided.
- **deploy_environment** – The current deploy environment.
- **force_stacks** – A list of stacks to force work on. Used to work on locked stacks.
- **logger** – Custom logger.

- **parameters** – Parameters passed from Runway or read from a file.
- **stack_names** – A list of stack_names to operate on. If not passed, all stacks defined in the config will be operated on.
- **work_dir** – Working directory used by Runway.

property base_fqn: str

Return `namespace` sanitized for use as an S3 Bucket name.

property bucket_name: Optional[str]

Return `cfngin_bucket` from config, calculated name, or None.

property mappings: Dict[str, Dict[str, Dict[str, Any]]]

Return `mappings` from config.

property namespace: str

Return `namespace` from config.

property namespace_delimiter: str

Return `namespace_delimiter` from config or default.

property persistent_graph_location: *PersistentGraphLocation*

Location of the persistent graph in s3.

property persistent_graph_locked: bool

Check if persistent graph is locked.

property persistent_graph_lock_code: Optional[str]

Code used to lock the persistent graph S3 object.

property persistent_graph_tags: Dict[str, str]

Cache of tags on the persistent graph object.

property persistent_graph: Optional[*runway.cfngin.plan.Graph*]

Graph if a persistent graph is being used.

Will create an "empty" object in S3 if one is not found.

property s3_bucket_verified: bool

Check CFNgin bucket exists and you have access.

If the CFNgin bucket does not exist, will try to create one.

property s3_client: S3Client

AWS S3 client.

property stacks_dict: Dict[str, *runway.cfngin.stack.Stack*]

Construct a dict of `{stack.fqn: Stack}` for easy access to stacks.

__new__(***kwargs*)

property boto3_credentials: *runway.type_defs.Boto3CredentialsTypeDef*

Return a dict of boto3 credentials.

property current_aws_creds: *EnvVarsAwsCredentialsTypeDef*

AWS credentials from self.env_vars.

get_session(*, *aws_access_key_id: Optional[str] = None*, *aws_secret_access_key: Optional[str] = None*, *aws_session_token: Optional[str] = None*, *profile: Optional[str] = None*, *region: Optional[str] = None*) → boto3.session.Session

Create a thread-safe boto3 session.

If `profile` is provided, it will take priority.

If no credential arguments are passed, will attempt to find them in environment variables.

Parameters

- **aws_access_key_id** – AWS Access Key ID.
- **aws_secret_access_key** – AWS secret Access Key.
- **aws_session_token** – AWS session token.
- **profile** – The profile for the session.
- **region** – The region for the session.

Returns A thread-safe boto3 session.

property is_interactive: bool

Whether the user should be prompted or not.

Determined by the existed of `CI` in the environment.

property is_noninteractive: bool

Whether the user should be prompted or not.

Determined by the existed of `CI` in the environment. Inverse of `is_interactive` property.

property stacks: List[*runway.cfngin.stack.Stack*]

Stacks for the current action.

sys_info: *SystemInfo*

Information about the current system being used to run Runway.

work_dir: Path

Working directory used by Runway. Should always be a directory named `.runway`.

property tags: Dict[str, str]

Return `tags` from config.

property template_indent: int

Return `template_indent` from config or default.

property upload_to_s3: bool

Check if S3 should be used for caching/persistent graph.

copy() → *runway.context.CfnginContext*

Copy the contents of this object into a new instance.

get_fqn(*name: Optional[str] = None*) → str

Return the fully qualified name of an object within this context.

If the name passed already appears to be a fully qualified name, it will be returned with no further processing.

get_stack(*name: str*) → Optional[*runway.cfngin.stack.Stack*]

Get a stack by name.

Parameters **name** – Name of a Stack as defined in the config.

lock_persistent_graph(*lock_code: str*) → None

Locks the persistent graph in s3.

Parameters **lock_code** – The code that will be used to lock the S3 object.

Raises

- *runway.cfngin.exceptions.PersistentGraphLocked* –
- *runway.cfngin.exceptions.PersistentGraphCannotLock* –

put_persistent_graph(*lock_code: str*) → None

Upload persistent graph to s3.

Parameters **lock_code** (*str*) – The code that will be used to lock the S3 object.

Raises

- *runway.cfngin.exceptions.PersistentGraphUnlocked* –
- *runway.cfngin.exceptions.PersistentGraphLockCodeMismatch* –

set_hook_data(*key: str*, *data: Any*) → None

Set hook data for the given key.

Parameters

- **key** – The key to store the hook data in.
- **data** – A dictionary of data to store, as returned from a hook.

unlock_persistent_graph(*lock_code: str*) → bool

Unlocks the persistent graph in s3.

Parameters **lock_code** – The code that will be used to lock the S3 object.

Raises *runway.cfngin.exceptions.PersistentGraphCannotUnlock* –

class runway.context.**RunwayContext**

Bases: runway.context._base.BaseContext

Runway context object.

__init__(*, *command: Optional[RunwayActionTypeDef] = None*, *deploy_environment: Optional[DeployEnvironment] = None*, *logger: Union[PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.context._runway (WARNING)>*, *work_dir: Optional[Path] = None*, ***_: Any*) → None

Instantiate class.

Parameters

- **command** – Runway command/action being run.
- **deploy_environment** – The current deploy environment.
- **logger** – Custom logger.
- **work_dir** – Working directory used by Runway.

command: Optional[RunwayActionTypeDef]

Runway command/action being run.

property no_color: bool

Whether to explicitly disable color output.

Primarily applies to IaC being wrapped by Runway.

property use_concurrent: bool

Whether to use concurrent.futures or not.

Noninteractive is required for concurrent execution to prevent weird user-input behavior.

Python 3 is required because backported futures has issues with ProcessPoolExecutor.

__new__(***kwargs*)

property boto3_credentials: *runway.type_defs.Boto3CredentialsTypeDef*

Return a dict of boto3 credentials.

copy() → *runway.context.RunwayContext*

Copy the contents of this object into a new instance.

property current_aws_creds: *EnvVarsAwsCredentialsTypeDef*

AWS credentials from self.env_vars.

get_session(*, *aws_access_key_id: Optional[str] = None*, *aws_secret_access_key: Optional[str] = None*, *aws_session_token: Optional[str] = None*, *profile: Optional[str] = None*, *region: Optional[str] = None*) → boto3.session.Session

Create a thread-safe boto3 session.

If `profile` is provided, it will take priority.

If no credential arguments are passed, will attempt to find them in environment variables.

Parameters

- **aws_access_key_id** – AWS Access Key ID.
- **aws_secret_access_key** – AWS secret Access Key.
- **aws_session_token** – AWS session token.
- **profile** – The profile for the session.
- **region** – The region for the session.

Returns A thread-safe boto3 session.

property is_interactive: bool

Whether the user should be prompted or not.

Determined by the existed of `CI` in the environment.

property is_noninteractive: bool

Whether the user should be prompted or not.

Determined by the existed of `CI` in the environment. Inverse of `is_interactive` property.

env: *DeployEnvironment*

Object containing information about the environment being deployed to.

logger: Union[PrefixAdaptor, RunwayLogger]
Custom logger.

sys_info: *SystemInfo*
Information about the current system being used to run Runway.

work_dir: Path
Working directory used by Runway. Should always be a directory named `.runway`.

echo_detected_environment() → None
Print a helper note about how the environment was determined.

Submodules

runway.context.sys_info module

System Information.

class runway.context.sys_info.**OsInfo**
Bases: object

Information about the operating system running on the current system.

static __new__(*cls*, **args: Any*, ***kwargs: Any*) → *runway.context.sys_info.OsInfo*
Create a new instance of class.

This class is a singleton so it will always return the same instance.

property is_darwin: bool
Operating system is Darwin.

property is_linux: bool
Operating system is Linux.

property is_macos: bool
Operating system is macOS.

Does not differentiate between macOS and Darwin.

property is_posix: bool
Operating system is posix.

property is_windows: bool
Operating system is Windows.

property name: str
Operating system name set to lowercase for consistency.

classmethod clear_singleton() → None
Clear singleton instances.

Intended to only be used for running tests.

__init__()

class runway.context.sys_info.**SystemInfo**

Bases: object

Information about the system running Runway.

static **__new__**(*cls*, **args: Any*, ***kwargs: Any*) → *runway.context.sys_info.SystemInfo*

Create a new instance of class.

This class is a singleton so it will always return the same instance.

property **is_frozen**: bool

Whether or not Runway is running from a frozen package (Pyinstaller).

property **os**: *runway.context.sys_info.OsInfo*

Operating system information.

classmethod **clear_singleton**() → None

Clear singleton instances.

Intended to only be used for running tests.

__init__()

runway.context.type_defs module

Context type definitions.

class runway.context.type_defs.**PersistentGraphLocation**

Bases: typing_extensions.TypedDict

CFNgin persistent graph location.

runway.core package

Core Runway API.

class runway.core.**Runway**

Bases: object

Runway's core functionality.

__init__(*config:* runway.config.RunwayConfig, *context:* runway.context.RunwayContext) → None

Instantiate class.

Parameters

- **config** – Runway config.
- **context** – Runway context.

deploy(*deployments: Optional[List[*RunwayDeploymentDefinition*]] = None*) → None

Deploy action.

Parameters **deployments** – List of deployments to run. If not provided, all deployments in the config will be run.

destroy(*deployments: Optional[List[RunwayDeploymentDefinition]] = None*) → None

Destroy action.

Parameters **deployments** – List of deployments to run. If not provided, all deployments in the config will be run in reverse.

get_env_vars(*deployments: Optional[List[RunwayDeploymentDefinition]] = None*) → Dict[str, Any]

Get env_vars defined in the config.

Parameters **deployments** – List of deployments to get env_vars from.

Returns Resolved env_vars from the deployments.

init(*deployments: Optional[List[RunwayDeploymentDefinition]] = None*) → None

Init action.

Parameters **deployments** – List of deployments to run. If not provided, all deployments in the config will be run.

plan(*deployments: Optional[List[RunwayDeploymentDefinition]] = None*) → None

Plan action.

Parameters **deployments** – List of deployments to run. If not provided, all deployments in the config will be run.

static reverse_deployments(*deployments: List[RunwayDeploymentDefinition]*) → List[*RunwayDeploymentDefinition*]

Reverse deployments and the modules within them.

Parameters **deployments** – List of deployments to reverse.

Returns Deployments and modules in reverse order.

test() → None

Run tests defined in the config.

__new__(***kwargs*)

Subpackages

runway.core.components package

Core Runway components.

class runway.core.components.**DeployEnvironment**

Bases: *runway.mixins.DelCachedPropMixin*

Runway deploy environment.

__init__(**, environ: Optional[Dict[str, str]] = None, explicit_name: Optional[str] = None, ignore_git_branch: bool = False, root_dir: Optional[pathlib.Path] = None*) → None

Instantiate class.

Parameters

- **environ** – Environment variables.
- **explicit_name** – Explicitly provide the deploy environment name.

- **ignore_git_branch** – Ignore the git branch when determining the deploy environment name.
- **root_dir** – Root directory of the project.

property aws_credentials: *runway.type_defs.EnvVarsAwsCredentialsTypeDef*

Get AWS credentials from environment variables.

property aws_profile: Optional[str]

Get AWS profile from environment variables.

property aws_region: str

Get AWS region from environment variables.

property branch_name: Optional[str]

Git branch name.

property ci: bool

Return CI status.

Returns bool

property debug: bool

Get debug setting from the environment.

__new__(***kwargs*)

property ignore_git_branch: bool

Whether to ignore git branch when determining name.

property max_concurrent_cfngin_stacks: int

Max number of CFNgin stacks that can be deployed concurrently.

This property can be set by exporting `RUNWAY_MAX_CONCURRENT_CFNGIN_STACKS`. If no value is specified, the value will be constrained based on the underlying graph.

Returns Value from environment variable or `0`.

property max_concurrent_modules: int

Max number of modules that can be deployed to concurrently.

This property can be set by exporting `RUNWAY_MAX_CONCURRENT_MODULES`. If no value is specified, `min(61, os.cpu_count())` is used.

On Windows, this must be equal to or lower than `61`.

IMPORTANT: When using `parallel_regions` and `child_modules` together, please consider the nature of their relationship when manually setting this value. (`parallel_regions * child_modules`)

Returns Value from environment variable or `min(61, os.cpu_count())`

property max_concurrent_regions: int

Max number of regions that can be deployed to concurrently.

This property can be set by exporting `RUNWAY_MAX_CONCURRENT_REGIONS`. If no value is specified, `min(61, os.cpu_count())` is used.

On Windows, this must be equal to or lower than `61`.

IMPORTANT: When using `parallel_regions` and `child_modules` together, please consider the nature of their relationship when manually setting this value. (`parallel_regions * child_modules`)

Returns Value from environment variable or `min(61, os.cpu_count())`

property name: str

Deploy environment name.

property verbose: bool

Get verbose setting from the environment.

copy() → *runway.core.components._deploy_environment.DeployEnvironment*

Copy the contents of this object into a new instance.

Returns New instance with the same contents.

Return type *DeployEnvironment*

log_name() → None

Output name to log.

class runway.core.components.**Deployment**

Bases: object

Runway deployment.

__init__(*context:* RunwayContext, *definition:* RunwayDeploymentDefinition, *future: Optional[*RunwayFutureDefinitionModel*] = None*, *variables: Optional[*RunwayVariablesDefinition*] = None*) → None

Instantiate class.

Parameters

- **context** – Runway context object.
- **definition** – A single deployment definition.
- **future** – Future functionality configuration.
- **variables** – Runway variables.

property assume_role_config: Dict[str, Union[bool, int, str]]

Parse the definition to get assume role arguments.

property env_vars_config: Dict[str, str]

Parse the definition to get the correct env_vars configuration.

property regions: List[str]

List of regions this deployment is associated with.

property use_async: bool

Whether to use asynchronous method.

deploy() → None

Deploy the deployment.

High level method for running a deployment.

destroy() → None

Destroy the deployment.

High level method for running a deployment.

init() → None

Initialize/bootstrap deployment.

High level method for running a deployment.

plan() → None

Plan for the next deploy of the deployment.

High level method for running a deployment.

run(*action: RunwayActionTypeDef*, *region: str*) → None

Run a single deployment in a single region.

Low level API access to run a deployment object.

Parameters

- **action** – Action to run (deploy, destroy, plan, etc.)
- **region** – AWS region to run in.

validate_account_credentials(*context: Optional[RunwayContext] = None*) → None

Exit if requested deployment account doesn't match credentials.

Parameters **context** – Context object.

Raises **SystemExit** – AWS Account associated with the current credentials did not match the defined criteria.

classmethod run_list(*action: RunwayActionTypeDef*, *context: RunwayContext*, *deployments: List[RunwayDeploymentDefinition]*, *future: RunwayFutureDefinitionModel*, *variables: RunwayVariablesDefinition*) → None

Run a list of deployments.

Parameters

- **action** – Name of action to run.
- **context** – Runway context.
- **deployments** – List of deployments to run.
- **future** – Future definition.
- **variables** – Runway variables for lookup resolution.

__getitem__(*name: str*) → Any

Make the object subscriptable.

Parameters **name** – Attribute to get.

__new__(***kwargs*)

class runway.core.components.**Module**

Bases: object

Runway module.

__init__(*context: RunwayContext*, *definition: RunwayModuleDefinition*, *deployment: RunwayDeploymentDefinition = None*, *future: RunwayFutureDefinitionModel = None*, *variables: RunwayVariablesDefinition = None*) → None

Instantiate class.

Parameters

- **context** – Runway context object.
- **definition** – A single module definition.
- **deployment** – Deployment that this module is a part of.

- **future** – Future functionality configuration.
- **variables** – Runway variables.

property child_modules: List[*runway.core.components._module.Module*]

Return child modules.

property environment_matches_defined: Optional[bool]

Environment matches one of the defined environments.

Will return None if there is nothing defined for the current environment.

property environments: RunwayEnvironmentsType

Environments defined for the deployment and module.

property fqn

Fully qualified name.

property opts_from_file: Dict[str, Any]

Load module options from local file.

property path: *runway.core.components._module_path.ModulePath*

Return resolve module path.

property payload: Dict[str, Any]

Return payload to be passed to module class handler class.

property should_skip: bool

Whether the module should be skipped by Runway.

property type: *runway.core.components._module_type.RunwayModuleType*

Determine Runway module type.

property use_async: bool

Whether to use asynchronous method.

deploy() → None

Deploy the module.

High level method for running a module.

destroy() → None

Destroy the module.

High level method for running a module.

init() → None

Initialize/bootstrap module.

High level method for running a deployment.

plan() → None

Plan for the next deploy of the module.

High level method for running a module.

run(*action: RunwayActionTypeDef*) → None

Run a single module.

Low level API access to run a module object.

Parameters **action** – Name of action to run.

__new__(**kwargs)

classmethod run_list(*action: RunwayActionTypeDef*, *context:* RunwayContext, *modules: List[*RunwayModuleDefinition*]*, *variables:* RunwayVariablesDefinition, *deployment:* RunwayDeploymentDefinition = *None*, *future: Optional[*RunwayFutureDefinitionModel*]* = *None*) → None

Run a list of modules.

Parameters

- **action** – Name of action to run.
- **context** – Runway context.
- **modules** – List of modules to run.
- **variables** – Variable definition for resolving lookups in the module.
- **deployment** – Deployment the modules are a part of.
- **future** – Future functionality configuration.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

class runway.core.components.**ModulePath**

Bases: object

Handler for the path field of a Runway module.

__init__(*definition: Optional[Union[pathlib.Path, str]]* = *None*, *, *cache_dir: pathlib.Path*, *deploy_environment: Optional[*runway.core.components._deploy_environment.DeployEnvironment*]* = *None*) → None

Instantiate class.

Parameters

- **definition** – Path definition.
- **cache_dir** – Directory to use for caching if needed.
- **deploy_environment** – Current deploy environment object.

property arguments: Dict[str, str]

Remote source arguments.

property location: str

Location of the module.

property metadata: runway.core.components._module_path.ModulePathMetadataTypeDef

Information that describes the module path.

property module_root: pathlib.Path

Root directory of the module.

property source: str

Source of the module.

property uri: str

Remote source URI.

classmethod parse_obj(*obj: Optional[Union[pathlib.Path, runway.config.components.runway._module_def.RunwayModuleDefinition, runway.config.models.runway.RunwayModuleDefinitionModel, str]]*, ***, *cache_dir: pathlib.Path*, *deploy_environment: Optional[runway.core.components._deploy_environment.DeployEnvironment] = None*) → *runway.core.components._module_path.ModulePath*

Parse object.

Parameters

- **obj** – Object to parse.
- **cache_dir** – Directory to use for caching if needed.
- **deploy_environment** – Current deploy environment object.

Raises **TypeError** – Unsupported type provided.

__new__(***kwargs*)

class runway.core.components.**RunwayModuleType**

Bases: object

Runway configuration `type` settings object.

The `type` property of a Runway configuration can be used to explicitly specify what module type you are intending to deploy.

Runway determines the type of module you are trying to deploy in 3 different ways. First, it will check for the `type` property as described here, next it will look for a suffix as described in *Module Definition*, and finally it will attempt to autodetect your module type by scanning the files of the project. If none of those settings produces a valid result an error will occur. The following are valid explicit types:

Type	IaC Tool/Framework
`cdk`	AWS CDK
`cloudformation`	CloudFormation
`serverless`	Serverless Framework
`terraform`	Terraform
`kubernetes`	Kubernetes
`static`	*Static Site*

Even when specifying a module `type` the module structure needs to be conducive with that type of project. If the files contained within don't match the type then an error will occur.

__init__(*path: Path*, *class_path: Optional[str] = None*, *type_str: Optional[RunwayModuleTypeTypeDef] = None*) → None

Instantiate class.

Keyword Arguments

- **path** – The required path to the module
- **class_path** – A supplied class_path to override the autodetected one.
- **type_str** – An explicit type to assign to the RunwayModuleType

__new__(***kwargs*)

runway.core.providers package

Runway providers.

Subpackages

runway.core.providers.aws package

Runway AWS objects.

class runway.core.providers.aws.**AccountDetails**

Bases: object

AWS account details.

__init__(*context: Union[*CfnginContext, RunwayContext*]*) → None

Instantiate class.

Parameters **context** – Runway context object.

property aliases: List[str]

Get the aliases of the AWS account.

property id: str

Get the ID of the AWS account.

__new__(***kwargs*)

class runway.core.providers.aws.**AssumeRole**

Bases: ContextManager[*AssumeRole*]

Context manager for assuming an AWS role.

__init__(*context:* runway.context.RunwayContext, *role_arn: Optional[str] = None*, *duration_seconds: Optional[int] = None*, *revert_on_exit: bool = True*, *session_name: Optional[str] = None*)

Instantiate class.

Parameters

- **context** – Runway context object.
- **role_arn** – ARN of role to be assumed.
- **duration_seconds** – Seconds that the assumed role's credentials will be valid for. (default: 3600)
- **revert_on_exit** – Whether credentials in the environment will be reverted upon exiting the context manager.
- **session_name** – Name to use for the assumed role session. (default: runway)

__new__(***kwargs*)

assume() → None

Perform role assumption.

restore_existing_iam_env_vars() → None

Restore backed up IAM environment variables.

save_existing_iam_env_vars() → None

Backup IAM environment variables for later restoration.

__enter__() → *runway.core.providers.aws._assume_role.AssumeRole*

Enter the context manager.

__exit__(*exc_type: Optional[Type[BaseException]]*, *exc_value: Optional[BaseException]*, *traceback: Optional[TracebackType]*) → None

Exit the context manager.

class runway.core.providers.aws.**BaseResponse**

Bases: *runway.utils.BaseModel*

Analyse the response from AWS S3 HeadBucket API response.

Keyword Arguments

- **Error** – Information about a service or networking error.
- **ResponseMetadata** – Information about the request.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

error: *runway.core.providers.aws._response.ResponseError*

Information about a service or networking error.

metadata: *runway.core.providers.aws._response.ResponseMetadata*

Information about the request.

class runway.core.providers.aws.**ResponseError**

Bases: *runway.utils.BaseModel*

Analyse the response from AWS S3 HeadBucket API response.

Keyword Arguments

- **Code** – A unique short code representing the error that was emitted.
- **Message** – A longer human readable error message.

code: str

A unique short code representing the error that was emitted.

message: str

A longer human readable error message.

__bool__() → bool

Implement evaluation of instances as a bool.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.core.providers.aws.**ResponseMetadata**

Bases: *runway.utils.BaseModel*

Analyse the response from AWS S3 HeadBucket API response.

Keyword Arguments

- **HostId** – Host ID data.

- **HTTPHeaders** – A map of response header keys and their respective values.
- **HTTPStatusCode** – The HTTP status code of the response (e.g., 200, 404).
- **RequestId** – The unique request ID associated with the response. Log this value when debugging requests for AWS support.
- **RetryAttempts** – The number of retries that were attempted before the request was completed.

host_id: `str`

Host ID data.

https_headers: `Dict[str, Any]`

A map of response header keys and their respective values.

http_status_code: `int`

he HTTP status code of the response (e.g., 200, 404).

request_id: `str`

The unique request ID associated with the response. Log this value when debugging requests for AWS support.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

retry_attempts: int

The number of retries that were attempted before the request was completed.

property forbidden: bool

Whether the response returned 403 (forbidden).

property not_found: bool

Whether the response returned 404 (Not Found).

Subpackages

runway.core.providers.aws.s3 package

AWS S3 objects.

class runway.core.providers.aws.s3.**Bucket**

Bases: *runway.mixins.DelCachedPropMixin*

AWS S3 bucket.

__init__(*context: Union[*CfnginContext, RunwayContext*]*, *name: str*, *region: Optional[str] = None*) → None

Instantiate class.

Parameters

- **context** – Current context object.
- **name** – The name of the bucket.
- **region** – The bucket's region.

property client: S3Client

Create or reuse a boto3 client.

property exists: bool

Check whether the bucket exists and is accessible.

property forbidden: bool

Check whether access to the bucket is forbidden.

property head

Check if a bucket exists and you have permission to access it.

To use this operation, the user must have permissions to perform the `s3:ListBucket` action.

This is a low level action that returns the raw result of the request.

property not_found: bool

Check whether the bucket exists.

property session: boto3.Session

Create cached boto3 session.

create(***kwargs: Any*) → Optional[CreateBucketOutputTypeDef]

Create an S3 Bucket if it does not already exist.

Bucket creation will be skipped if it already exists or access is forbidden.

Keyword arguments are passed directly to the boto3 method.

Returns boto3 response.

`enable_versioning`() → None

Enable versioning on the bucket if not already enabled.

`format_bucket_path_uri`(*, *key: Optional[str] = None*, *prefix: Optional[str] = None*) → str

Format bucket path URI.

Parameters

- **key** – S3 object key.
- **prefix** – Directory tree to append to key.

Returns S3 bucket URI in `s3://{bucket-name}/{prefix}/{key}` format

`get_versioning`() → GetBucketVersioningOutputTypeDef

Get the versioning state of a bucket.

To retrieve the versioning state of a bucket, you must be the bucket owner.

Returns The current versioning state of the bucket containing `Status` and `MFADelete` (only if this has ever been configured).

`sync_from_local`(*src_directory: str*, *, *delete: bool = False*, *exclude: Optional[List[str]] = None*, *follow_symlinks: bool = False*, *include: Optional[List[str]] = None*, *prefix: Optional[str] = None*) → None

Sync local directory to the S3 Bucket.

Parameters

- **src_directory** – Local directory to sync to S3.
- **delete** – If true, files that exist in the destination but not in the source are deleted.
- **exclude** – List of patterns for files/objects to exclude.
- **follow_symlinks** – If symlinks should be followed.
- **include** – List of patterns for files/objects to explicitly include.
- **prefix** – Optional prefix to append to synced objects.

`sync_to_local`(*dest_directory: str*, *, *delete: bool = False*, *exclude: Optional[List[str]] = None*, *follow_symlinks: bool = False*, *include: Optional[List[str]] = None*, *prefix: Optional[str] = None*) → None

Sync S3 bucket to local directory.

Parameters

- **dest_directory** – Local directory to sync S3 objects to.
- **delete** – If true, files that exist in the destination but not in the source are deleted.
- **exclude** – List of patterns for files/objects to exclude.
- **follow_symlinks** – If symlinks should be followed.
- **include** – List of patterns for files/objects to explicitly include.
- **prefix** – Optional prefix to append to synced objects.

`__bool__`() → bool

Implement evaluation of instances as a bool.

`__new__`(**kwargs*)

Submodules

runway.core.providers.aws.s3.exceptions module

AWS S3 exceptions.

exception runway.core.providers.aws.s3.exceptions.**BucketAccessDeniedError**

Bases: *runway.exceptions.RunwayError*

Access denied to S3 Bucket.

__init__(*bucket:* Bucket) → None

Instantiate class.

Parameters **bucket** – AWS S3 Bucket object.

bucket_name: str

Name of the S3 Bucket.

message: str

Error message.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.core.providers.aws.s3.exceptions.**BucketNotFoundError**

Bases: *runway.exceptions.RunwayError*

S3 Bucket not found.

__init__(*bucket:* Bucket) → None

Instantiate class.

Parameters **bucket** – AWS S3 Bucket object.

bucket_name: str

Name of the S3 Bucket

message: str

Error message.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.core.providers.aws.s3.exceptions.**S3ObjectDoesNotExistError**

Bases: *runway.exceptions.RunwayError*

Required S3 object does not exist.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*bucket: str*, *key: str*) → None

Instantiate class.

Parameters

- **bucket** – Name of the S3 bucket.
- **key** – S3 object key.

bucket: str

Name of the S3 Bucket

key: str

S3 object key.

uri: str

S3 object URI.

message: str

Error message.

Submodules

runway.core.providers.aws.type_defs module

AWS type definitions.

class runway.core.providers.aws.type_defs.**TagTypeDef**

Bases: typing_extensions.TypedDict

AWS resource tags.

Submodules

runway.core.type_defs module

Type definitions.

runway.dependency_managers package

Classes for interacting with dependency managers using subprocesses.

class runway.dependency_managers.**Pip**

Bases: *runway.dependency_managers.base_classes.DependencyManager*

pip CLI interface.

CONFIG_FILES: Final[Tuple[Literal['requirements.txt']]] = ('requirements.txt',)

Configuration files used by pip.

EXECUTABLE: Final[Literal['pip']] = 'pip'

CLI executable.

property python_version: runway.utils._version.Version

Python version where pip is installed (<major>.<minor> only).

property version: runway.utils._version.Version

pip version.

classmethod dir_is_project(*directory: StrPath*, ***kwargs: Any*) → bool

Determine if the directory contains a project for this dependency manager.

Parameters **directory** – Directory to check.

classmethod generate_install_command(*, *cache_dir: Optional[StrPath] = None*, *no_cache_dir: bool = False*, *no_deps: bool = False*, *requirements: StrPath*, *target: StrPath*) → List[str]

Generate the command that when run will install dependencies.

This method is exposed to easily format the command to be run by with a subprocess or within a Docker container.

Parameters

- **cache_dir** – Store the cache data in the provided directory.
- **no_cache_dir** – Disable the cache.
- **no_deps** – Don't install package dependencies.
- **requirements** – Path to a `requirements.txt` file.
- **target** – Path to a directory where dependencies will be installed.

__init__(*context: Union[CfnginContext, RunwayContext]*, *cwd: StrPath*) → None

Instantiate class.

Parameters

- **context** – CFNgin or Runway context object.
- **cwd** – Working directory where commands will be run.

__new__(***kwargs*)

static convert_to_cli_arg(*arg_name: str*, *, *prefix: str = '--'*) → str

Convert string kwarg name into a CLI argument.

classmethod found_in_path() → bool

Determine if executable is found in $PATH.

install(*, *cache_dir: Optional[StrPath] = None*, *extend_args: Optional[List[str]] = None*, *no_cache_dir: bool = False*, *no_deps: bool = False*, *requirements: StrPath*, *target: StrPath*) → Path

Install dependencies to a target directory.

Parameters

- **cache_dir** – Store the cache data in the provided directory.
- **extend_args** – Optional list of extra arguments to pass to `pip install`. This value will not be parsed or sanitized in any way - it will be used as is. It is the user's responsibility to ensure that there are no overlapping arguments between this list and the arguments that are automatically generated.
- **no_cache_dir** – Disable the cache.
- **no_deps** – Don't install package dependencies.
- **requirements** – Path to a `requirements.txt` file.

- **target** – Path to a directory where dependencies will be installed.

Raises *PipInstallFailedError* – The subprocess used to run the commend exited with an error.

static list2cmdline(*split_command: Iterable[str]*) → str

Combine a list of strings into a string that can be run as a command.

Handles multi-platform differences.

ctx: Union[*CfnginContext*, *RunwayContext*]

CFNgin or Runway context object.

cwd: Path

Working directory where commands will be run.

classmethod generate_command(*command: Union[List[str], str], **kwargs: Optional[Union[bool, Iterable[str], str]]*) → List[str]

Generate command to be executed and log it.

Parameters

- **command** – Command to run.
- **args** – Additional args to pass to the command.

Returns The full command to be passed into a subprocess.

exception runway.dependency_managers.**PipInstallFailedError**

Bases: *runway.exceptions.RunwayError*

Pip install failed.

__init__(**args: Any, **kwargs: Any*) → None

Instantiate class. All args/kwargs are passed to parent method.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class runway.dependency_managers.**Pipenv**

Bases: *runway.dependency_managers.base_classes.DependencyManager*

Pipenv dependency manager.

CONFIG_FILES: Final[Tuple[Literal['Pipfile'], Literal['Pipfile.lock']]] = ('Pipfile', 'Pipfile.lock')

Configuration files used by pipenv.

EXECUTABLE: Final[Literal['pipenv']] = 'pipenv'

CLI executable.

property version: runway.utils._version.Version

pipenv version.

__init__(*context: Union[*CfnginContext, RunwayContext*], cwd: StrPath*) → None

Instantiate class.

Parameters

- **context** – CFNgin or Runway context object.

- **cwd** – Working directory where commands will be run.

__new__(***kwargs*)

static convert_to_cli_arg(*arg_name: str*, ***, *prefix: str = '--'*) → str

Convert string kwarg name into a CLI argument.

classmethod dir_is_project(*directory: StrPath*, ***_Pipenv__kwargs: Any*) → bool

Determine if the directory contains a project for this dependency manager.

Parameters **directory** – Directory to check.

classmethod found_in_path() → bool

Determine if executable is found in $PATH.

classmethod generate_command(*command: Union[List[str], str]*, ***kwargs: Optional[Union[bool, Iterable[str], str]]*) → List[str]

Generate command to be executed and log it.

Parameters

- **command** – Command to run.
- **args** – Additional args to pass to the command.

Returns The full command to be passed into a subprocess.

static list2cmdline(*split_command: Iterable[str]*) → str

Combine a list of strings into a string that can be run as a command.

Handles multi-platform differences.

ctx: Union[*CfnginContext*, *RunwayContext*]

CFNgin or Runway context object.

cwd: Path

Working directory where commands will be run.

export(***, *dev: bool = False*, *output: StrPath*) → Path

Export the lock file to other formats (requirements.txt only).

The underlying command being executed by this method is `pipenv lock --requirements`.

Parameters

- **dev** – Include development dependencies.
- **output** – Path to the output file.

exception runway.dependency_managers.**PipenvExportFailedError**

Bases: *runway.exceptions.RunwayError*

Pipenv export failed to produce a `requirements.txt` file.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class. All args/kwargs are passed to parent method.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.dependency_managers.**PipenvNotFoundError**

Bases: *runway.exceptions.RunwayError*

Pipenv not installed or found in $PATH.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class. All args/kwargs are passed to parent method.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class runway.dependency_managers.**Poetry**

Bases: *runway.dependency_managers.base_classes.DependencyManager*

Poetry dependency manager.

CONFIG_FILES: Final[Tuple[Literal['poetry.lock'], Literal['pyproject.toml']]] = ('poetry.lock', 'pyproject.toml')

Configuration files used by poetry.

EXECUTABLE: Final[Literal['poetry']] = 'poetry'

CLI executable.

property version: runway.utils._version.Version

poetry version.

__init__(*context: Union[*CfnginContext, RunwayContext*]*, *cwd: StrPath*) → None

Instantiate class.

Parameters

- **context** – CFNgin or Runway context object.
- **cwd** – Working directory where commands will be run.

__new__(***kwargs*)

static convert_to_cli_arg(*arg_name: str*, ***, *prefix: str = '--'*) → str

Convert string kwarg name into a CLI argument.

classmethod dir_is_project(*directory: StrPath*, ***_Poetry__kwargs: Any*) → bool

Determine if the directory contains a project for this dependency manager.

Parameters **directory** – Directory to check.

classmethod found_in_path() → bool

Determine if executable is found in $PATH.

classmethod generate_command(*command: Union[List[str], str]*, ***kwargs: Optional[Union[bool, Iterable[str], str]]*) → List[str]

Generate command to be executed and log it.

Parameters

- **command** – Command to run.
- **args** – Additional args to pass to the command.

Returns The full command to be passed into a subprocess.

static list2cmdline(*split_command: Iterable[str]*) → str

Combine a list of strings into a string that can be run as a command.

Handles multi-platform differences.

ctx: Union[*CfnginContext*, *RunwayContext*]

CFNgin or Runway context object.

cwd: Path

Working directory where commands will be run.

export(*, *dev: bool = False*, *extras: Optional[List[str]] = None*, *output: StrPath*, *output_format: str = 'requirements.txt'*, *with_credentials: bool = True*, *without_hashes: bool = True*) → Path

Export the lock file to other formats.

Parameters

- **dev** – Include development dependencies.
- **extras** – Extra sets of dependencies to include.
- **output** – Path to the output file.
- **output_format** – The format to export to.
- **with_credentials** – Include credentials for extra indices.
- **without_hashes** – Exclude hashes from the exported file.

Returns Path to the output file.

exception runway.dependency_managers.**PoetryExportFailedError**

Bases: *runway.exceptions.RunwayError*

Poetry export failed to produce a `requirements.txt` file.

__init__(*output: str*, **args: Any*, ***kwargs: Any*) → None

Instantiate class. All args/kwargs are passed to parent method.

Parameters **output** – The output from running `poetry export`.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.dependency_managers.**PoetryNotFoundError**

Bases: *runway.exceptions.RunwayError*

Poetry not installed or found in $PATH.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class. All args/kwargs are passed to parent method.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Submodules

runway.dependency_managers.base_classes module

Base classes for dependency managers.

class runway.dependency_managers.base_classes.**DependencyManager**

Bases: *runway.mixins.CliInterfaceMixin*

Dependency manager for the AWS Lambda runtime.

Dependency managers are interfaced with via subprocess to ensure that the correct version is being used. This is primarily target at Python dependency manager that we could import and use directly.

CONFIG_FILES: ClassVar[Tuple[str, ...]]

Configuration files used by the dependency manager.

__init__(*context: Union[*CfnginContext, RunwayContext*]*, *cwd: StrPath*) → None

Instantiate class.

Parameters

- **context** – CFNgin or Runway context object.
- **cwd** – Working directory where commands will be run.

ctx: Union[*CfnginContext*, *RunwayContext*]

CFNgin or Runway context object.

cwd: Path

Working directory where commands will be run.

property version: Version

Get executable version.

__new__(***kwargs*)

static convert_to_cli_arg(*arg_name: str*, *, *prefix: str = '--'*) → str

Convert string kwarg name into a CLI argument.

classmethod dir_is_project(*directory: StrPath*, ***_DependencyManager__kwargs: Any*) → bool

Determine if the directory contains a project for this dependency manager.

Parameters **directory** – Directory to check.

classmethod found_in_path() → bool

Determine if executable is found in $PATH.

classmethod generate_command(*command: Union[List[str], str]*, ***kwargs: Optional[Union[bool, Iterable[str], str]]*) → List[str]

Generate command to be executed and log it.

Parameters

- **command** – Command to run.
- **args** – Additional args to pass to the command.

Returns The full command to be passed into a subprocess.

static list2cmdline(*split_command: Iterable[str]*) → str

Combine a list of strings into a string that can be run as a command.

Handles multi-platform differences.

EXECUTABLE: ClassVar[str]

CLI executable.

runway.env_mgr package

Base module for environment managers.

runway.env_mgr.**handle_bin_download_error**(*exc: URLError*, *name: str*) → None

Give user info about their failed download.

Raises SystemExit – Always raised after logging reason.

class runway.env_mgr.**EnvManager**

Bases: *runway.mixins.DelCachedPropMixin*

Base environment manager class.

binPath to the binary of the current version.

current_version

The current binary version being used.

Type Optional[str]

env_dir_name

Name of the directory within the users home directory where binary versions will be stored.

Type str

path

The current working directory.

Type pathlib.Path

__init__(*bin_name: str*, *dir_name: str*, *path: Optional[pathlib.Path] = None*) → None

Initialize class.

Parameters

- **bin_name** – Name of the binary file (e.g. kubectl)
- **dir_name** – Name of the directory within the users home directory where binary versions will be stored.
- **path** – The current working directory.

property bin: pathlib.Path

Path to the version binary.

Returns Path

property command_suffix: str

Return command suffix based on platform.system.

property env_dir: pathlib.Path

Return the directory used to store version binaries.

property versions_dir: pathlib.Path

Return the directory used to store binary.

When first used, the existence of the directory is checked and it is created if needed.

__new__(***kwargs*)

property version_file: Optional[pathlib.Path]

Find and return a "<bin version file>" file if one is present.

Returns Path to the <bin> version file.

install(*version_requested: Optional[str] = None*) → str

Ensure <bin> is installed.

list_installed() → Generator[pathlib.Path, None, None]

List installed versions of <bin>.

uninstall(*version: Union[str, Version]*) → bool

Uninstall a version of the managed binary.

Parameters **version** – Version of binary to uninstall.

Returns Whether a version of the binary was uninstalled or not.

Submodules

runway.env_mgr.kbenv module

Kubectl version management.

runway.env_mgr.kbenv.**verify_kb_release**(*kb_url: str*, *download_dir: str*, *filename: str*) → None

Compare checksum and exit if it doesn't match.

Different releases provide varying checksum files. To account for this, start at SHA512 and work down to the first available checksum.

requests is used for downloading these small files because of difficulty in getting 404 status from urllib on py2. Once py2 support is dropped, downloads can be moved to urllib.

https://stackoverflow.com/questions/1308542/how-to-catch-404-error-in-urllib-urlretrieve

runway.env_mgr.kbenv.**download_kb_release**(*version: str*, *versions_dir: Path*, *kb_platform: Optional[str] = None*, *arch: Optional[str] = None*) → None

Download kubectl and return path to it.

class runway.env_mgr.kbenv.**KBEnvManager**

Bases: *runway.env_mgr.EnvManager*

kubectl version management.

Designed to be compatible with https://github.com/alexppg/kbenv.

__init__(*path: Optional[Path] = None*, ***, *overlay_path: Optional[Path] = None*) → None

Initialize class.

Parameters

- **path** – Module path.
- **overlay_path** – Path to Kustomize overlay.

property version: Optional[runway.utils._version.Version]

Terraform version.

property version_file: Optional[Path]

Find and return a ".kubectl-version" file if one is present.

Returns Path to the kubectl version file.

get_version_from_file(*file_path: Optional[Path] = None*) → Optional[str]

Get kubectl version from a file.

Parameters **file_path** – Path to file that will be read.

install(*version_requested: Optional[str] = None*) → str

Ensure kubectl is available.

list_installed() → Generator[Path, None, None]

List installed versions of kubectl.

Only lists versions of kubectl that have been installed by an instance if this class or by kbenv.

set_version(*version: str*) → None

Set current version.

Clears cached values as needed.

Parameters **version** – Version string. Must be in the format of `v<major>.<minor>.<patch>` with an optional `-<prerelease>`.

classmethod parse_version_string(*version: str*) → runway.utils._version.Version

Parse version string into a `Version`.

Parameters **version** – Version string to parse. Must be in the format of `<major>.<minor>.<patch>` with an optional `-<prerelease>`.

__new__(***kwargs*)

property bin: pathlib.Path

Path to the version binary.

Returns Path

property command_suffix: str

Return command suffix based on platform.system.

property env_dir: pathlib.Path

Return the directory used to store version binaries.

uninstall(*version: Union[str, Version]*) → bool

Uninstall a version of the managed binary.

Parameters **version** – Version of binary to uninstall.

Returns Whether a version of the binary was uninstalled or not.

property versions_dir: pathlib.Path

Return the directory used to store binary.

When first used, the existence of the directory is checked and it is created if needed.

runway.env_mgr.tfenv module

Terraform version management.

runway.env_mgr.tfenv.**download_tf_release**(*version: str*, *versions_dir: Path*, *command_suffix: str*, *tf_platform: Optional[str] = None*, *arch: Optional[str] = None*) → None

Download Terraform archive and return path to it.

runway.env_mgr.tfenv.**get_available_tf_versions**(*include_prerelease: bool = False*) → List[str]

Return available Terraform versions.

runway.env_mgr.tfenv.**get_latest_tf_version**(*include_prerelease: bool = False*) → str

Return latest Terraform version.

runway.env_mgr.tfenv.**load_terraform_module**(*parser: ModuleType*, *path: Path*) → Dict[str, Any]

Load all Terraform files in a module into one dict.

Parameters

- **parser** (*Union[hcl, hcl2]*) – Parser to use when loading files.
- **path** – Terraform module path. All Terraform files in the path will be loaded.

class runway.env_mgr.tfenv.**TFEnvManager**

Bases: *runway.env_mgr.EnvManager*

Terraform version management.

Designed to be compatible with https://github.com/tfutils/tfenv.

__init__(*path: Optional[Path] = None*) → None

Initialize class.

property backend: Dict[str, Any]

Backend config of the Terraform module.

property terraform_block: Dict[str, Any]

Collect Terraform configuration blocks from a Terraform module.

property version: Optional[runway.utils._version.Version]

Terraform version.

property version_file: Optional[Path]

Find and return a ".terraform-version" file if one is present.

Returns Path to the Terraform version file.

get_min_required() → str

Get the defined minimum required version of Terraform.

Returns The minimum required version as defined in the module.

get_version_from_file(*file_path: Optional[Path] = None*) → Optional[str]

Get Terraform version from a file.

Parameters **file_path** – Path to file that will be read.

install(*version_requested: Optional[str] = None*) → str

Ensure Terraform is available.

list_installed() → Generator[Path, None, None]

List installed versions of Terraform.

Only lists versions of Terraform that have been installed by an instance if this class or by tfenv.

set_version(*version: str*) → None

Set current version.

Clears cached values as needed.

Parameters **version** – Version string. Must be in the format of <major>.<minor>.<patch> with an optional -<prerelease>.

classmethod get_version_from_executable(*bin_path: Union[Path, str]*, *, *cwd: Optional[Union[Path, str]] = None*, *env: Optional[Dict[str, str]] = None*) → Optional[Version]

Get Terraform version from an executable.

Parameters

- **bin_path** – Path to the Terraform binary to retrieve the version from.
- **cwd** – Current working directory to use when calling the executable.
- **env** – Environment variable overrides.

classmethod parse_version_string(*version: str*) → runway.utils._version.Version

Parse version string into a Version.

Parameters **version** – Version string to parse. Must be in the format of <major>.<minor>.<patch> with an optional -<prerelease>.

__new__(***kwargs*)

property bin: pathlib.Path

Path to the version binary.

Returns Path

property command_suffix: str

Return command suffix based on platform.system.

property env_dir: pathlib.Path

Return the directory used to store version binaries.

uninstall(*version: Union[str, Version]*) → bool

Uninstall a version of the managed binary.

Parameters **version** – Version of binary to uninstall.

Returns Whether a version of the binary was uninstalled or not.

property versions_dir: pathlib.Path

Return the directory used to store binary.

When first used, the existence of the directory is checked and it is created if needed.

runway.lookups package

Empty init for python import traversal.

Subpackages

runway.lookups.handlers package

Runway lookup handlers.

Submodules

runway.lookups.handlers.base module

Base class for lookup handlers.

runway.lookups.handlers.base.**str2bool**(*v: str*)

Return boolean value of string.

class runway.lookups.handlers.base.**LookupHandler**

Bases: object

Base class for lookup handlers.

TYPE_NAME: ClassVar[str]

Name that the Lookup is registered as.

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises TypeError – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*_LookupHandler__value: str*, *context: Union[CfnginContext, RunwayContext]*, **_LookupHandler__args: Any*, *provider: Optional[Provider] = None*, ***_LookupHandler__kwargs: Any*) → Any

Perform the lookup.

Parameters

- **__value** – Parameter(s) given to the lookup.
- **context** – The current context object.
- **provider** – CFNgin AWS provider.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

__init__()

__new__(***kwargs*)

runway.lookups.handlers.cfn module

Retrieve a value from CloudFormation Stack Outputs.

The query syntax for this lookup is `<stack-name>.<output-name>`. When specifying the output name, be sure to use the *Logical ID* of the output; not the *Export.Name*.

class runway.lookups.handlers.cfn.**OutputQuery**

Bases: NamedTuple

Output query NamedTuple.

stack_name: *str*

Alias for field number 0

output_name: *str*

Alias for field number 1

__init__()

static __new__(*_cls*, *stack_name: str*, *output_name: str*)

Create new instance of OutputQuery(stack_name, output_name)

count(*value*, */*)

Return number of occurrences of value.

index(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

class runway.lookups.handlers.cfn.**CfnLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

CloudFormation Stack Output lookup.

TYPE_NAME: Final[typing_extensions.Literal[cfn]] = 'cfn'

Name that the Lookup is registered as.

static should_use_provider(*args: Dict[str, str]*, *provider: Optional[*Provider*]*) → bool

Determine if the provider should be used for the lookup.

This will open happen when the lookup is used with CFNgin.

Parameters

- **args** – Parsed arguments provided to the lookup.
- **provider** – CFNgin provider.

static get_stack_output(*client: CloudFormationClient*, *query:* OutputQuery) → str

Get CloudFormation Stack output.

Parameters

- **client** – Boto3 CloudFormation client.
- **query** – What to get.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises **TypeError** – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, ***, *provider: Optional[Provider] = None*, ***_: Any*) → Any

Retrieve a value from CloudFormation Stack outputs.

Parameters

- **value** – The value passed to the Lookup.
- **context** – The current context object.
- **provider** – AWS provider.

Returns Result of the query.

Raises ***OutputDoesNotExist*** – Output does not exist on the Stack provided and default was not provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, *, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.lookups.handlers.ecr module

Retrieve a value from AWS Elastic Container Registry (ECR).

class runway.lookups.handlers.ecr.**EcrLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

ECR Lookup.

TYPE_NAME: Final[typing_extensions.Literal[ecr]] = 'ecr'

Name that the Lookup is registered as.

static get_login_password(*client: ECRClient*) → str

Get a password to login to ECR registry.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Union[*CfnginContext, RunwayContext*]*, **_EcrLookup__args: Any*, ***_EcrLookup__kwargs: Any*) → Any

Retrieve a value from AWS Elastic Container Registry (ECR).

Parameters

- **value** – The value passed to the Lookup.
- **context** – The current context object.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.lookups.handlers.env module

Retrieve a value from an environment variable.

class runway.lookups.handlers.env.**EnvLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Environment variable Lookup.

TYPE_NAME: Final[typing_extensions.Literal[env]] = 'env'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **_EnvLookup__args: Any*, ***_EnvLookup__kwargs: Any*) → Any

Retrieve an environment variable.

The value is retrieved from a copy of the current environment variables that is saved to the context object. These environment variables are manipulated at runtime by Runway to fill in additional values such as `DEPLOY_ENVIRONMENT` and `AWS_REGION` to match the current execution.

Parameters

- **value** – The value passed to the Lookup.
- **context** – The current context object.

Raises `ValueError` – Unable to find a value for the provided query and a default value was not provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.lookups.handlers.random_string module

Generate a random string.

class runway.lookups.handlers.random_string.**ArgsDataModel**

Bases: *runway.utils.BaseModel*

Arguments data model.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.lookups.handlers.random_string.**RandomStringLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Random string lookup.

TYPE_NAME: Final[typing_extensions.Literal[random.string]] = 'random.string'

Name that the Lookup is registered as.

static calculate_char_set(*args:* runway.lookups.handlers.random_string.ArgsDataModel) → str

Calculate character set from the provided arguments.

static generate_random_string(*char_set: Sequence[str]*, *length: int*) → str

Generate a random string of a set length from a set of characters.

static has_digit(*value: str*) → bool

Check if value contains a digit.

static has_lowercase(*value: str*) → bool

Check if value contains lowercase.

static has_punctuation(*value: str*) → bool

Check if value contains uppercase.

static has_uppercase(*value: str*) → bool

Check if value contains uppercase.

classmethod ensure_has_one_of(*args:* runway.lookups.handlers.random_string.ArgsDataModel, *value: str*) → bool

Ensure value has at least one of each required character.

Parameters

- **args** – Hook args.
- **value** – Value to check.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises **TypeError** – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **_RandomStringLookup__args: Any*, ***_RandomStringLookup__kwargs: Any*) → Any

Generate a random string.

Parameters

- **value** – The value passed to the Lookup.
- **context** – The current context object.

Raises **ValueError** – Unable to find a value for the provided query and a default value was not provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.lookups.handlers.ssm module

Retrieve a value from SSM Parameter Store.

class runway.lookups.handlers.ssm.**SsmLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

SSM Parameter Store Lookup.

TYPE_NAME: Final[typing_extensions.Literal[ssm]] = 'ssm'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query: VariableValue*) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.

3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, *context: Union[CfnginContext, RunwayContext]*, **_SsmLookup__args: Any*, ***_SsmLookup__kwargs: Any*) → Any

Retrieve a value from SSM Parameter Store.

Parameters

- **value** – The value passed to the Lookup.
- **context** – The current context object.

Raises **ParameterNotFound** – Parameter not found in SSM and a default value was not provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

runway.lookups.handlers.var module

Retrieve a variable from the variables file or definition.

class runway.lookups.handlers.var.**VarLookup**

Bases: *runway.lookups.handlers.base.LookupHandler*

Variable definition Lookup.

TYPE_NAME: Final[typing_extensions.Literal[var]] = 'var'

Name that the Lookup is registered as.

__init__()

__new__(***kwargs*)

classmethod dependencies(*_LookupHandler__lookup_query:* VariableValue) → Set[str]

Calculate any dependencies required to perform this lookup.

Note that lookup_query may not be (completely) resolved at this time.

classmethod format_results(*value: Any*, *get: Optional[str] = None*, *load: Optional[str] = None*, *transform: Optional[typing_extensions.Literal[bool, str]] = None*, ***kwargs: Any*) → Any

Format results to be returned by a lookup.

Parameters

- **value** – Data collected by the Lookup.
- **get** – Nested value to get from a dictionary like object.
- **load** – Parser to use to parse a formatted string before the `get` and `transform` method.
- **transform** – Convert the final value to a different data type before returning it.

Raises `TypeError` – If `get` is provided but the value value is not a dictionary like object.

Runs the following actions in order:

1. *load()* if `load` is provided.
2. `runway.util.MutableMap.find()` or `dict.get()` depending on the data type if `get` is provided.
3. Convert null value string to `NoneType` object. This includes string values of "None" and "null". This conversion is case insensitive.
4. *transform()* if `transform` is provided.

classmethod handle(*value: str*, **_VarLookup__args: Any*, *variables:* MutableMap, ***_VarLookup__kwargs: Any*) → Any

Retrieve a variable from the variable definition.

The value is retrieved from the variables passed to Runway using either a variables file or the `variables` directive of the config file.

Parameters

- **value** – The value passed to the Lookup.
- **variables** – The resolved variables pass to Runway.

Raises `ValueError` – Unable to find a value for the provided query and a default value was not provided.

classmethod load(*value: Any*, *parser: Optional[str] = None*, ***kwargs: Any*) → Any

Load a formatted string or object into a python data type.

First action taken in *format_results()*. If a lookup needs to handling loading data to process it before it enters *format_results()*, is should use `args.pop('load')` to prevent the data from being loaded twice.

Parameters

- **value** – What is being loaded.
- **parser** – Name of the parser to use.

Returns The loaded value.

classmethod parse(*value: str*) → Tuple[str, Dict[str, str]]

Parse the value passed to a lookup in a standardized way.

Parameters **value** – The raw value passed to a lookup.

Returns The lookup query and a dict of arguments

classmethod transform(*value: Any*, ***, *to_type: Optional[typing_extensions.Literal[bool, str]] = 'str'*, ***kwargs: Any*) → Any

Transform the result of a lookup into another datatype.

Last action taken in *format_results()*. If a lookup needs to handling transforming the data in a way that the base class can't support it should overwrite this method of the base class to register different transform methods.

Parameters

- **value** – What is to be transformed.
- **to_type** – The type the value will be transformed into.

Returns The transformed value.

Submodules

runway.lookups.registry module

Register test handlers.

runway.lookups.registry.**register_lookup_handler**(*lookup_type: str*, *handler_or_path: Union[str, Type[runway.lookups.handlers.base.LookupHandler]]*) → None

Register a lookup handler.

Parameters

- **lookup_type** – Name to register the handler under
- **handler_or_path** – a function or a path to a handler

runway.lookups.registry.**unregister_lookup_handler**(*lookup_type: str*) → None

Unregister the specified test type.

This is useful when testing various lookup types if you want to unregister the lookup type after the test runs.

Parameters **lookup_type** – Name of the lookup type to unregister

runway.module package

Classes and utilities for working with each Runway module type.

Subpackages

runway.module.staticsite package

Runway Static Site Module.

class runway.module.staticsite.**StaticSite**

Bases: *runway.module.base.RunwayModule*

Static website Runway Module.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.staticsite.handler (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property sanitized_name: str

Sanitized name safe to use in a CloudFormation Stack name.

Errors are usually caused here by a `.` in the name. This unintelligently replaces `.` with `-`.

If issues are still encountered, we can check against the regex of `(?=^.{1,128}$)^[a-zA-Z][-a-zA-Z0-9_]+$`.

deploy() → None

Create website CFN module and run CFNgin.deploy.

destroy() → None

Create website CFN module and run CFNgin.destroy.

init() → None

Run init.

plan() → None

Create website CFN module and run CFNgin.diff.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

Subpackages

runway.module.staticsite.options package

Runway Static Site Module options.

class runway.module.staticsite.options.**RunwayStaticSiteExtraFileDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module extra_files option item.

content_type

An explicit content type for the file. If not provided, will attempt to determine based on the name provided.

Type Optional[str]

content

Inline content that will be used as the file content. This or `file` must be provided.

Type Any

file

Path to an existing file. The content of this file will be uploaded to the static site S3 bucket using the name as the object key. This or `content` must be provided.

Type Optional[pathlib.Path]

name

The destination name of the file to create.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model

- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.**RunwayStaticSiteModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module options.

build_output

Directory where build output is placed. Defaults to current working directory.

Type str

build_steps

List of commands to run to build the static site.

Type List[str]

extra_files

List of files that should be uploaded to S3 after the build. Used to dynamically create or select file.

Type List[*runway.module.staticsite.options.models.RunwayStaticSiteExtraFileDataModel*]

pre_build_steps

Commands to be run prior to the build process.

Type List[*runway.module.staticsite.options.models.RunwayStaticSitePreBuildStepDataModel*]

source_hashing

Overrides for source hash calculation and tracking.

Type *runway.module.staticsite.options.models.RunwayStaticSiteSourceHashingDataModel*

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.

- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.**RunwayStaticSitePreBuildStepDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module pre_build_steps option item.

command

The command to run.

Type str

cwd

The working directory for the subprocess running the command. If not provided, the current working directory is used.

Type pathlib.Path

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.**RunwayStaticSiteSourceHashingDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module source_hashing option.

directories

Explicitly provide the directories to use when calculating the hash. If not provided, will default to the root of the module.

Type List[*runway.module.staticsite.options.models.RunwayStaticSiteSourceHashingDirectoryDataModel*]

enabled

Enable source hashing. If not enabled, build and upload will occur on every deploy.

Type bool

parameter

SSM parameter where the hash of each build is stored.

Type Optional[str]

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.**RunwayStaticSiteSourceHashingDirectoryDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module source_hashing.directory option item.

exclusions

List of gitignore formatted globs to ignore when calculating the hash.

Type List[str]

path

Path to files to include in the hash.

Type pathlib.Path

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.**StaticSiteOptions**

Bases: *runway.module.base.ModuleOptions*

Static site options.

build_output

Directory where build output is placed. Defaults to current working directory.

build_steps

List of commands to run to build the static site.

data

Options parsed into a data model.

extra_files

List of files that should be uploaded to S3 after the build. Used to dynamically create or select file.

pre_build_steps

Commands to be run prior to the build process.

source_hashing

Overrides for source hash calculation and tracking.

__init__(*data:* runway.module.staticsite.options.models.RunwayStaticSiteModuleOptionsDataModel) → None

Instantiate class.

classmethod parse_obj(*obj: object*) → *runway.module.staticsite.options.components.StaticSiteOptions*

Parse options definition and return an options object.

Parameters **obj** – Object to parse.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

Submodules

runway.module.staticsite.options.components module

Runway Static Site Module options component classes.

class runway.module.staticsite.options.components.**StaticSiteOptions**

Bases: *runway.module.base.ModuleOptions*

Static site options.

build_output

Directory where build output is placed. Defaults to current working directory.

build_steps

List of commands to run to build the static site.

data

Options parsed into a data model.

extra_files

List of files that should be uploaded to S3 after the build. Used to dynamically create or select file.

pre_build_steps

Commands to be run prior to the build process.

source_hashing

Overrides for source hash calculation and tracking.

__init__(*data:* runway.module.staticsite.options.models.RunwayStaticSiteModuleOptionsDataModel) → None

Instantiate class.

classmethod parse_obj(*obj: object*) → *runway.module.staticsite.options.components.StaticSiteOptions*

Parse options definition and return an options object.

Parameters **obj** – Object to parse.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

runway.module.staticsite.options.models module

Runway static site Module options.

class runway.module.staticsite.options.models.**RunwayStaticSiteExtraFileDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module extra_files option item.

content_type

An explicit content type for the file. If not provided, will attempt to determine based on the name provided.

Type Optional[str]

content

Inline content that will be used as the file content. This or `file` must be provided.

Type Any

file

Path to an existing file. The content of this file will be uploaded to the static site S3 bucket using the name as the object key. This or `content` must be provided.

Type Optional[pathlib.Path]

name

The destination name of the file to create.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.models.**RunwayStaticSitePreBuildStepDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module pre_build_steps option item.

command

The command to run.

Type str

cwd

The working directory for the subprocess running the command. If not provided, the current working directory is used.

Type pathlib.Path

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.models.**RunwayStaticSiteSourceHashingDirectoryDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module source_hashing.directory option item.

exclusions

List of gitignore formatted globs to ignore when calculating the hash.

Type List[str]

path

Path to files to include in the hash.

Type pathlib.Path

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

`__init__`(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

`__iter__`() → TupleGenerator

so *dict(model)* works

`__new__`(***kwargs*)

`__pretty__`(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

`__repr_name__`() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.models.**RunwayStaticSiteSourceHashingDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module source_hashing option.

directories

Explicitly provide the directories to use when calculating the hash. If not provided, will default to the root of the module.

Type List[*runway.module.staticsite.options.models.RunwayStaticSiteSourceHashingDirectoryDataModel*]

enabled

Enable source hashing. If not enabled, build and upload will occur on every deploy.

Type bool

parameter

SSM parameter where the hash of each build is stored.

Type Optional[str]

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

object_pairs_hook is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

parse_float, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

parse_int, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

parse_constant, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom JSONDecoder subclass, specify it with the cls kwarg; otherwise JSONDecoder is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model

- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.options.models.**RunwayStaticSiteModuleOptionsDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module options.

build_output

Directory where build output is placed. Defaults to current working directory.

Type str

build_steps

List of commands to run to build the static site.

Type List[str]

extra_files

List of files that should be uploaded to S3 after the build. Used to dynamically create or select file.

Type List[*runway.module.staticsite.options.models.RunwayStaticSiteExtraFileDataModel*]

pre_build_steps

Commands to be run prior to the build process.

Type List[*runway.module.staticsite.options.models.RunwayStaticSitePreBuildStepDataModel*]

source_hashing

Overrides for source hash calculation and tracking.

Type *runway.module.staticsite.options.models.RunwayStaticSiteSourceHashingDataModel*

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.

- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

runway.module.staticsite.parameters package

Runway Static Site Module parameters.

class runway.module.staticsite.parameters.**RunwayStaticSiteCustomErrorResponseDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway stat site Module staticsite_custom_error_responses parameter item.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-cloudfront-distribution-customerrorrespons
html

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

`__rich_repr__`() → RichReprResult

Get fields for Rich library

`__setitem__`(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

`classmethod __try_update_forward_refs__`(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

`classmethod construct`(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

`copy`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

`dict`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

`get`(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

`json`(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(**localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class
runway.module.staticsite.parameters.**RunwayStaticSiteLambdaFunctionAssociationDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway stat site Module staticsite_lambda_function_associations parameter item.

arn

Lambda function ARN.

Type str

type

Association type.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(**kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of pydantic.utils.GetterDict

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize obj to a JSON formatted str.

If skipkeys is true then dict keys that are not basic types (str, int, float, bool, None) will be skipped instead of raising a TypeError.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.parameters.**RunwayStaticSiteModuleParametersDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module parameters.

acmcert_arn

The certificate arn used for any alias domains supplied. This is a requirement when supplying any custom domain.

Type Optional[str]

additional_redirect_domains

Additional domains (beyond the `aliases` domains or the CloudFront URL if no aliases are provided) that will be authorized by the Auth@Edge UserPool AppClient.

Type List[str]

aliases

Any custom domains that should be added to the CloudFront Distribution.

Type List[str]

auth_at_edge

Auth@Edge make the static site private by placing it behind an authorization wall.

Type bool

cf_disable

Wether deployment of the CloudFront Distribution should be disabled.

Type bool

compress

Whether the CloudFront default cache behavior will automatically compress certain files.

Type bool

cookie_settings

The default cookie settings for retrieved tokens and generated nonce's.

Type Dict[str, str]

create_user_pool

Wether to create a User Pool for the Auth@Edge configuration.

Type bool

custom_error_responses

Define custom error responses.

Type List[*runway.module.staticsite.parameters.models.RunwayStaticSiteCustomErrorResponseDataModel*]

enable_cf_logging

Enable CloudFront logging.

Type bool

http_headers

Headers that should be sent with each origin response.

Type Dict[str, str]

lambda_function_associations

This allows the user to deploy custom Lambda@Edge associations with their pre-build function versions.

Type List[*runway.module.staticsite.parameters.models.RunwayStaticSiteLambdaFunctionAssociationDataModel*]

namespace

The unique namespace for the deployment.

Type str

non_spa

Wether this site is a single page application (SPA).

Type bool

oauth_scopes

Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account.

Type List[str]

redirect_path_auth_refresh

The path that a user is redirected to when their authorization tokens have expired (1 hour).

Type str

redirect_path_sign_in

The path that a user is redirected to after sign-in.

Type str

redirect_path_sign_out

The path that a user is redirected to after sign-out.

Type str

required_group

Name of Cognito User Pool group of which users must be a member to be granted access to the site. If `None`, allows all UserPool users to have access.

Type Optional[str]

rewrite_directory_index

Deploy a Lambda@Edge function designed to rewrite directory indexes.

Type Optional[str]

role_boundary_arn

Defines an IAM Managed Policy that will be set as the permissions boundary for any IAM Roles created to support the site.

Type Optional[str]

service_role

IAM role that CloudFormation will use.

Type Optional[str]

sign_out_url

The path a user should access to sign themselves out of the application.

Type str

supported_identity_providers

A comma delimited list of the User Pool client identity providers.

Type List[str]

user_pool_arn

The ARN of a pre-existing Cognito User Pool to use with Auth@Edge.

Type Optional[str]

web_acl

The ARN of a web access control list (web ACL) to associate with the CloudFront Distribution.

Type Optional[str]

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises `AttributeError` – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(**kwargs)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

Submodules

runway.module.staticsite.parameters.models module

Runway static site Module parameters.

class
runway.module.staticsite.parameters.models.**RunwayStaticSiteCustomErrorResponseDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway stat site Module staticsite_custom_error_responses parameter item.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-cloudfront-distribution-customerrorrespons
html

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises AttributeError – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.module.staticsite.parameters.models.**RunwayStaticSiteLambdaFunctionAssociationDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway stat site Module staticsite_lambda_function_associations parameter item.

arn

Lambda function ARN.

Type str

type

Association type.

Type str

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If ensure_ascii is false, then the return value can contain non-ASCII characters if they appear in strings contained in obj. Otherwise, all such characters are escaped in JSON strings.

If check_circular is false, then the circular reference check for container types will be skipped and a circular reference will result in an RecursionError (or worse).

If allow_nan is false, then it will be a ValueError to serialize out of range float values (nan, inf, -inf) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (NaN, Infinity, -Infinity).

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

default(obj) is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: False), then the output of dictionaries will be sorted by key.

To use a custom JSONEncoder subclass (e.g. one that overrides the .default() method to serialize additional types), specify it with the cls kwarg; otherwise JSONEncoder is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize s (a str, bytes or bytearray instance containing a JSON document) to a Python object.

object_hook is an optional function that will be called with the result of any object literal decode (a dict). The return value of object_hook will be used instead of the dict. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

object_pairs_hook is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of object_pairs_hook will be used instead of the dict. This feature can be used to implement custom decoders. If object_hook is also defined, the object_pairs_hook takes priority.

parse_float, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

parse_int, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

parse_constant, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom JSONDecoder subclass, specify it with the cls kwarg; otherwise JSONDecoder is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class
runway.module.staticsite.parameters.models.**RunwayStaticSiteModuleParametersDataModel**

Bases: *runway.config.models.base.ConfigProperty*

Model for Runway static site Module parameters.

acmcert_arn

The certificate arn used for any alias domains supplied. This is a requirement when supplying any custom domain.

Type Optional[str]

additional_redirect_domains

Additional domains (beyond the `aliases` domains or the CloudFront URL if no aliases are provided) that will be authorized by the Auth@Edge UserPool AppClient.

Type List[str]

aliases

Any custom domains that should be added to the CloudFront Distribution.

Type List[str]

auth_at_edge

Auth@Edge make the static site private by placing it behind an authorization wall.

Type bool

cf_disable

Wether deployment of the CloudFront Distribution should be disabled.

Type bool

compress

Whether the CloudFront default cache behavior will automatically compress certain files.

Type bool

cookie_settings

The default cookie settings for retrieved tokens and generated nonce's.

Type Dict[str, str]

create_user_pool

Wether to create a User Pool for the Auth@Edge configuration.

Type bool

custom_error_responses

Define custom error responses.

Type List[*runway.module.staticsite.parameters.models.RunwayStaticSiteCustomErrorResponseDataModel*]

enable_cf_logging

Enable CloudFront logging.

Type bool

http_headers

Headers that should be sent with each origin response.

Type Dict[str, str]

lambda_function_associations

This allows the user to deploy custom Lambda@Edge associations with their pre-build function versions.

Type List[*runway.module.staticsite.parameters.models.RunwayStaticSiteLambdaFunctionAssociationDataModel*]

namespace

The unique namespace for the deployment.

Type str

non_spa

Wether this site is a single page application (SPA).

Type bool

oauth_scopes

Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account.

Type List[str]

redirect_path_auth_refresh

The path that a user is redirected to when their authorization tokens have expired (1 hour).

Type str

redirect_path_sign_in

The path that a user is redirected to after sign-in.

Type str

redirect_path_sign_out

The path that a user is redirected to after sign-out.

Type str

required_group

Name of Cognito User Pool group of which users must be a member to be granted access to the site. If `None`, allows all UserPool users to have access.

Type Optional[str]

rewrite_directory_index

Deploy a Lambda@Edge function designed to rewrite directory indexes.

Type Optional[str]

role_boundary_arn

Defines an IAM Managed Policy that will be set as the permissions boundary for any IAM Roles created to support the site.

Type Optional[str]

service_role

IAM role that CloudFormation will use.

Type Optional[str]

sign_out_url

The path a user should access to sign themselves out of the application.

Type str

supported_identity_providers

A comma delimited list of the User Pool client identity providers.

Type List[str]

user_pool_arn

The ARN of a pre-existing Cognito User Pool to use with Auth@Edge.

Type Optional[str]

web_acl

The ARN of a web access control list (web ACL) to associate with the CloudFront Distribution.

Type Optional[str]

class Config

Bases: *runway.config.models.base.ConfigProperty.Config*

Model configuration.

__init__()

__new__(***kwargs*)

classmethod get_field_info(*name: unicode*) → Dict[str, Any]

Get properties of FieldInfo from the *fields* property of the config class.

getter_dict

alias of `pydantic.utils.GetterDict`

json_dumps(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort_keys=False*, ***kw*)

Serialize `obj` to a JSON formatted `str`.

If `skipkeys` is true then `dict` keys that are not basic types (`str`, `int`, `float`, `bool`, `None`) will be skipped instead of raising a `TypeError`.

If `ensure_ascii` is false, then the return value can contain non-ASCII characters if they appear in strings contained in `obj`. Otherwise, all such characters are escaped in JSON strings.

If `check_circular` is false, then the circular reference check for container types will be skipped and a circular reference will result in an `RecursionError` (or worse).

If `allow_nan` is false, then it will be a `ValueError` to serialize out of range `float` values (`nan`, `inf`, `-inf`) in strict compliance of the JSON specification, instead of using the JavaScript equivalents (`NaN`, `Infinity`, `-Infinity`).

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an `(item_separator, key_separator)` tuple. The default is `(', ', ': ')` if *indent* is `None` and `(',', ': ')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

`default(obj)` is a function that should return a serializable version of obj or raise TypeError. The default simply raises TypeError.

If *sort_keys* is true (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

json_loads(*, *cls=None*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *object_pairs_hook=None*, ***kw*)

Deserialize `s` (a `str`, `bytes` or `bytearray` instance containing a JSON document) to a Python object.

`object_hook` is an optional function that will be called with the result of any object literal decode (a `dict`). The return value of `object_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders (e.g. JSON-RPC class hinting).

`object_pairs_hook` is an optional function that will be called with the result of any object literal decoded with an ordered list of pairs. The return value of `object_pairs_hook` will be used instead of the `dict`. This feature can be used to implement custom decoders. If `object_hook` is also defined, the `object_pairs_hook` takes priority.

`parse_float`, if specified, will be called with the string of every JSON float to be decoded. By default this is equivalent to float(num_str). This can be used to use another datatype or parser for JSON floats (e.g. decimal.Decimal).

`parse_int`, if specified, will be called with the string of every JSON int to be decoded. By default this is equivalent to int(num_str). This can be used to use another datatype or parser for JSON integers (e.g. float).

`parse_constant`, if specified, will be called with one of the following strings: -Infinity, Infinity, NaN. This can be used to raise an exception if invalid JSON numbers are encountered.

To use a custom `JSONDecoder` subclass, specify it with the `cls` kwarg; otherwise `JSONDecoder` is used.

classmethod prepare_field(*field: ModelField*) → None

Optional hook to check or modify fields during model creation.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model

- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

Submodules

runway.module.staticsite.handler module

Static website Module.

class runway.module.staticsite.handler.**StaticSite**

Bases: *runway.module.base.RunwayModule*

Static website Runway Module.

__init__(*context: RunwayContext*, ***, *explicitly_enabled: Optional[bool] = False*, *logger: RunwayLogger = <RunwayLogger runway.module.staticsite.handler (WARNING)>*, *module_root: Path*, *name: Optional[str] = None*, *options: Optional[Union[Dict[str, Any], ModuleOptions]] = None*, *parameters: Optional[Dict[str, Any]] = None*, ***_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.

- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property sanitized_name: str

Sanitized name safe to use in a CloudFormation Stack name.

Errors are usually caused here by a `.` in the name. This unintelligently replaces `.` with `-`.

If issues are still encountered, we can check against the regex of `(?=^.{1,128}$)^[a-zA-Z][-a-zA-Z0-9_]+$`.

deploy() → None

Create website CFN module and run CFNgin.deploy.

destroy() → None

Create website CFN module and run CFNgin.destroy.

init() → None

Run init.

plan() → None

Create website CFN module and run CFNgin.diff.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

runway.module.staticsite.utils module

Static site utilities.

runway.module.staticsite.utils.**add_url_scheme**(*url: str*) → str

Add the scheme to an existing url.

Parameters **url** (*str*) – The current url.

Submodules

runway.module.base module

Base classes for runway modules.

class runway.module.base.**RunwayModule**

Bases: object

Base class for Runway modules.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.base (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

deploy() → None

Abstract method called when running deploy.

destroy() → None

Abstract method called when running destroy.

init() → None

Abstract method called when running init.

plan() → None

Abstract method called when running plan.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

class runway.module.base.**RunwayModuleNpm**

Bases: *runway.module.base.RunwayModule*

Base class for Runway modules that use npm.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.base (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

log_npm_command(*command: List[str]*) → None

Log an npm command that is going to be run.

Parameters **command** – List that will be passed into a subprocess.

npm_install() → None

Run `npm install`.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

deploy() → None

Abstract method called when running deploy.

destroy() → None

Abstract method called when running destroy.

init() → None

Abstract method called when running init.

package_json_missing() → bool

Check for the existence for a package.json file in the module.

Returns True if the file was not found.

Return type bool

plan() → None

Abstract method called when running plan.

static check_for_npm(*, *logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Ensure npm is installed and in the current path.

Parameters **logger** – Optionally provide a custom logger to use.

static warn_on_boto_env_vars(*env_vars: Dict[str, str], *, logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Inform user if boto-specific environment variables are in use.

Parameters

- **env_vars** – Environment variables to check.
- **logger** – Optionally provide a custom logger to use.

class runway.module.base.**ModuleOptions**

Bases: object

Base class for Runway module options.

__init__()

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

__eq__(*other: Any*) → bool

Assess equality.

runway.module.cdk module

CDK module.

class runway.module.cdk.**CloudDevelopmentKit**

Bases: *runway.module.base.RunwayModuleNpm*

CDK Runway Module.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.cdk (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.

- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property cli_args: List[str]

Generate CLI args from self used in all CDK commands.

property cli_args_context: List[str]

Generate CLI args from self passed to CDK commands as `--context`.

property skip: bool

Determine if the module should be skipped.

cdk_bootstrap() → None

Execute `cdk bootstrap` command.

cdk_deploy() → None

Execute `cdk deploy` command.

cdk_destroy() → None

Execute `cdk destroy` command.

cdk_diff(*stack_name: Optional[str] = None*) → None

Execute `cdk diff` command.

cdk_list() → List[str]

Execute `cdk list` command.

deploy() → None

Run cdk deploy.

destroy() → None

Run cdk destroy.

gen_cmd(*command: typing_extensions.Literal[bootstrap, context, deploy, destroy, diff, docs, doctor, init, list, metadata, synthesize]*, *args_list: Optional[List[str]] = None*, ***, *include_context: bool = False*) → List[str]

Generate and log a CDK command.

This does not execute the command, only prepares it for use.

Parameters

- **command** – The CDK command to be executed.
- **args_list** – Additional arguments to include in the generated command.
- **include_context** – Optionally, pass context to the CLI. Context is not valid for all commands.

Returns The full command to be passed into a subprocess.

init() → None

Run cdk bootstrap.

plan() → None

Run cdk diff.

run_build_steps() → None

Run build steps.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

static check_for_npm(**, logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Ensure npm is installed and in the current path.

Parameters **logger** – Optionally provide a custom logger to use.

log_npm_command(*command: List[str]*) → None

Log an npm command that is going to be run.

Parameters **command** – List that will be passed into a subprocess.

npm_install() → None

Run `npm install`.

package_json_missing() → bool

Check for the existence for a package.json file in the module.

Returns True if the file was not found.

Return type bool

static warn_on_boto_env_vars(*env_vars: Dict[str, str], *, logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Inform user if boto-specific environment variables are in use.

Parameters

- **env_vars** – Environment variables to check.
- **logger** – Optionally provide a custom logger to use.

class runway.module.cdk.**CloudDevelopmentKitOptions**

Bases: *runway.module.base.ModuleOptions*

Module options for AWS Cloud Development Kit.

build_steps

A list of commands to be executed before each action (e.g. diff, deploy, destroy).

data

Options parsed into a data model.

skip_npm_ci

Skip running `npm ci` in the module directory prior to processing the module.

__init__(*data:* runway.config.models.runway.options.cdk.RunwayCdkModuleOptionsDataModel) → None

Instantiate class.

Parameters **data** – Options parsed into a data model.

classmethod parse_obj(*obj: object*) → *runway.module.cdk.CloudDevelopmentKitOptions*

Parse options definition and return an options object.

Parameters **obj** – Object to parse.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

runway.module.cloudformation module

Cloudformation module.

class runway.module.cloudformation.**CloudFormation**

Bases: *runway.module.base.RunwayModule*

CloudFormation (CFNgin) Runway Module.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.cloudformation (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

deploy() → None

Run deploy.

destroy() → None

Run destroy.

init() → None

Run init.

plan() → None

Run diff.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

runway.module.k8s module

K8s (kustomize) module.

class runway.module.k8s.**K8s**

Bases: *runway.module.base.RunwayModule*

Kubectl Runway Module.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.k8s (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property kbenv: *runway.env_mgr.kbenv.KBEnvManager*

Kubectl environment manager.

property kubectl_bin: str

Path to kubectl binary.

property skip: bool

Determine if the module should be skipped.

deploy() → None

Run kubectl apply.

destroy() → None

Run kubectl delete.

gen_cmd(*command: typing_extensions.Literal[annotation, apply, auth, autoscale, cp, create, delete, describe, diff, edit, exec, expose, get, kustomize, label, logs, patch, port - forward, proxy]*, *args_list: Optional[List[str]] = None*) → List[str]

Generate and log a kubectl command.

This does not execute the command, only prepares it for use.

Parameters

- **command** – The CDK command to be executed.
- **args_list** – Additional arguments to include in the generated command.

Returns The full command to be passed into a subprocess.

init() → None

Run init.

kubectl_apply() → None

Execute `kubectl apply` command.

https://kubectl.docs.kubernetes.io/references/kubectl/apply/

kubectl_delete() → None

Execute `kubectl delete` command.

https://kubectl.docs.kubernetes.io/references/kubectl/delete/

kubectl_kustomize() → str

Execute `kubectl kustomize` command.

https://kubectl.docs.kubernetes.io/references/kubectl/kustomize/

plan() → None

Run kustomize build and display generated plan.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

class runway.module.k8s.**K8sOptions**

Bases: *runway.module.base.ModuleOptions*

Module options for Kubernetes.

data

Options parsed into a data model.

Type *runway.config.models.runway.options.k8s.RunwayK8sModuleOptionsDataModel*

deploy_environment

Runway deploy environment object.

Type *runway.core.components._deploy_environment.DeployEnvironment*

kubectl_version

Version of kubectl to use.

Type Optional[str]

path

Module path.

Type pathlib.Path

__init__(*data:* runway.config.models.runway.options.k8s.RunwayK8sModuleOptionsDataModel, *deploy_environment:* runway.core.components._deploy_environment.DeployEnvironment, *path:* *pathlib.Path*) → None

Instantiate class.

Parameters

- **data** – Options parsed into a data model.
- **deploy_environment** – Current deploy environment.
- **path** – Module path.

property kustomize_config: pathlib.Path

Kustomize configuration file.

property overlay_path: pathlib.Path

Directory containing the kustomize overlay to use.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

static gen_overlay_dirs(*environment: str*, *region: str*) → List[str]

Generate possible overlay directories.

Prefers more explicit directory name but falls back to environment name only.

Parameters

- **environment** – Current deploy environment.
- **region** – Current AWS region.

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

classmethod get_overlay_dir(*path: pathlib.Path*, *environment: str*, *region: str*) → pathlib.Path

Determine the overlay directory to use.

classmethod parse_obj(*deploy_environment:* runway.core.components._deploy_environment.DeployEnvironment, *obj: object*, *path: Optional[pathlib.Path] = None*) → *runway.module.k8s.K8sOptions*

Parse options definition and return an options object.

Parameters

- **deploy_environment** – Current deploy environment.
- **obj** – Object to parse.
- **path** – Module path.

runway.module.serverless module

Serverless module.

runway.module.serverless.**gen_sls_config_files**(*stage: str*, *region: str*) → List[str]

Generate possible SLS config files names.

class runway.module.serverless.**Serverless**

Bases: *runway.module.base.RunwayModuleNpm*

Serverless Runway Module.

__init__(*context: RunwayContext*, ***, *explicitly_enabled: Optional[bool] = False*, *logger: RunwayLogger = <RunwayLogger runway.module.serverless (WARNING)>*, *module_root: Path*, *name: Optional[str] = None*, *options: Optional[Union[Dict[str, Any], ModuleOptions]] = None*, *parameters: Optional[Dict[str, Any]] = None*, ***_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property cli_args: List[str]

Generate CLI args from self used in all Serverless commands.

property env_file: Optional[pathlib.Path]

Find the environment file for the module.

property skip: bool

Determine if the module should be skipped.

extend_serverless_yml(*func: Callable[[...], None]*) → None

Extend the Serverless config file with additional YAML from options.

Parameters **func** – Callable to use after handling the Serverless config file.

gen_cmd(*command: str*, *args_list: Optional[List[str]] = None*) → List[str]

Generate and log a Serverless command.

This does not execute the command, only prepares it for use.

Parameters

- **command** – The Serverless command to be executed.
- **args_list** – Additional arguments to include in the generated command.

Returns The full command to be passed into a subprocess.

`sls_deploy`(*, *package: Optional[AnyPath] = None*, *skip_install: bool = False*) → None

Execute `sls deploy` command.

Parameters

- **package** – Path to Serverless package to deploy.
- **skip_install** – Skip `npm ci|install` before running the Serverless command.

`sls_package`(*, *output_path: Optional[AnyPathConstrained] = None*, *skip_install: bool = False*) → Optional[AnyPathConstrained]

Execute `sls package` command.

Parameters

- **output_path** – Path where the package should be output.
- **skip_install** – Skip `npm ci|install` before running the Serverless command.

`sls_print`(*, *item_path: Optional[str] = None*, *skip_install: bool = False*) → Dict[str, Any]

Execute `sls print` command.

Keyword Arguments

- **item_path** – Period-separated path to print a sub-value (eg: "provider.name").
- **skip_install** – Skip `npm ci|install` before running the Serverless command.

Returns Resolved Serverless config file.

Raises **SystemExit** – If a runway-tmp.serverless.yml file already exists.

`sls_remove`(*, *skip_install: bool = False*) → None

Execute `sls remove` command.

Parameters **skip_install** – Skip `npm ci|install` before running the Serverless command.

`deploy`() → None

Entrypoint for Runway's deploy action.

`destroy`() → None

Entrypoint for Runway's destroy action.

`init`() → None

Run init.

`plan`() → None

Entrypoint for Runway's plan action.

`__getitem__`(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

`__new__`(***kwargs*)

`static check_for_npm`(*, *logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Ensure npm is installed and in the current path.

Parameters **logger** – Optionally provide a custom logger to use.

log_npm_command(*command: List[str]*) → None

Log an npm command that is going to be run.

Parameters **command** – List that will be passed into a subprocess.

npm_install() → None

Run `npm install`.

package_json_missing() → bool

Check for the existence for a package.json file in the module.

Returns True if the file was not found.

Return type bool

static warn_on_boto_env_vars(*env_vars: Dict[str, str], *, logger: Union[logging.Logger, PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.base (WARNING)>*) → None

Inform user if boto-specific environment variables are in use.

Parameters

- **env_vars** – Environment variables to check.
- **logger** – Optionally provide a custom logger to use.

class runway.module.serverless.**ServerlessArtifact**

Bases: `object`

Object for interacting with a Serverless artifact directory.

__init__(*context: RunwayContext, config: Dict[str, Any], *, logger: Union[PrefixAdaptor, RunwayLogger] = <RunwayLogger runway.module.serverless (WARNING)>, package_path: AnyPath, path: AnyPath*) → None

Instantiate class.

Parameters

- **context** – Runway context object.
- **config** – Rendered Serverless config file.
- **logger** – Logger this object will log to. If not provided, the logger in the local module will be used.
- **package_path** – Local path to the artifact directory.
- **path** – Root directory of the Serverless project.

property source_hash: Dict[str, str]

File hash(es) of each service's source code.

sync_with_s3(*bucket_name: str*) → None

Sync local archive files with S3 bucket.

Parameters **bucket_name** – Name of S3 bucket to upload files to.

__new__(***kwargs*)

class runway.module.serverless.**ServerlessOptions**

Bases: *runway.module.base.ModuleOptions*

Module options for Serverless Framework.

data

Options parsed into a data model.

extend_serverless_yml

If provided, the value of this option will be recursively merged into the module's Serverless config file.

promotezip

If provided, promote Serverless Framework generated zip files between environments from a build AWS account.

skip_npm_ci

Skip running `npm ci` in the module directory prior to processing the module.

__init__(*data:* runway.config.models.runway.options.serverless.RunwayServerlessModuleOptionsDataModel) → None

Instantiate class.

Parameters **data** – Options parsed into a data model.

property args: List[str]

List of CLI arguments/options to pass to the Serverless Framework CLI.

update_args(*key: str*, *value: str*) → None

Update a known CLI argument.

Parameters

- **key** – Dict key to be updated.
- **value** – New value

Raises **KeyError** – The key provided for update is not a known arg.

classmethod parse_obj(*obj: object*) → *runway.module.serverless.ServerlessOptions*

Parse options definition and return an options object.

Parameters **obj** – Object to parse.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

runway.module.terraform module

Terraform module.

runway.module.terraform.**gen_workspace_tfvars_files**(*environment: str*, *region: str*) → List[str]

Generate possible Terraform workspace tfvars filenames.

runway.module.terraform.**update_env_vars_with_tf_var_values**(*os_env_vars: Dict[str, str]*, *tf_vars: Dict[str, Union[Dict[str, Any], List[Any], str]]*) → Dict[str, str]

Return os_env_vars with TF_VAR values for each tf_var.

class runway.module.terraform.**Terraform**

Bases: *runway.module.base.RunwayModule*, *runway.mixins.DelCachedPropMixin*

Terraform Runway Module.

__init__(*context: RunwayContext, *, explicitly_enabled: Optional[bool] = False, logger: RunwayLogger = <RunwayLogger runway.module.terraform (WARNING)>, module_root: Path, name: Optional[str] = None, options: Optional[Union[Dict[str, Any], ModuleOptions]] = None, parameters: Optional[Dict[str, Any]] = None, **_: Any*) → None

Instantiate class.

Parameters

- **context** – Runway context object for the current session.
- **explicitly_enabled** – Whether or not the module is explicitly enabled. This is can be set in the event that the current environment being deployed to matches the defined environments of the module/deployment.
- **logger** – Used to write logs.
- **module_root** – Root path of the module.
- **name** – Name of the module.
- **options** – Options passed to the module class from the config as `options` or `module_options` if coming from the deployment level.
- **parameters** – Values to pass to the underlying infrastructure as code tool that will alter the resulting infrastructure being deployed. Used to templatize IaC.

property **auto_tfvars**: pathlib.Path

Return auto.tfvars file if one is being used.

property **current_workspace**: str

Wrap "terraform_workspace_show" to cache the value.

Returns The currently active Terraform workspace.

property **env_file**: List[str]

Find the environment file for the module.

property **skip**: bool

Determine if the module should be skipped.

property **tfenv**: *runway.env_mgr.tfenv.TFEnvManager*

Terraform environment manager.

property **tf_bin**: str

Path to Terraform binary.

property **version**: runway.utils._version.Version

Version of Terraform being used.

cleanup_dot_terraform() → None

Remove .terraform excluding the plugins directly.

This step is crucial for allowing Runway to deploy to multiple regions or deploy environments without prompting the user for input.

The plugins directory is retained to improve performance when they are used by subsequent runs.

deploy() → None

Run Terraform apply.

destroy() → None

Run Terraform destroy.

gen_command(*command: Union[List[str], str, Tuple[str, ...]]*, *args_list: Optional[List[str]] = None*) → List[str]

Generate Terraform command.

handle_backend() → None

Handle backend configuration.

This needs to be run before "skip" is assessed or env_file/auto_tfvars is used in case their behavior needs to be altered.

handle_parameters() → None

Handle parameters.

Either updating environment variables or writing to a file.

init() → None

Run init.

plan() → None

Run Terraform plan.

terraform_apply() → None

Execute `terraform apply` command.

https://www.terraform.io/docs/cli/commands/apply.html

terraform_destroy() → None

Execute `terraform destroy` command.

https://www.terraform.io/docs/cli/commands/destroy.html

terraform_get() → None

Execute `terraform get` command.

https://www.terraform.io/docs/cli/commands/get.html

terraform_init() → None

Execute `terraform init` command.

https://www.terraform.io/docs/cli/commands/init.html

terraform_plan() → None

Execute `terraform plan` command.

https://www.terraform.io/docs/cli/commands/plan.html

terraform_workspace_list() → str

Execute `terraform workspace list` command.

https://www.terraform.io/docs/cli/commands/workspace/list.html

Returns The available Terraform workspaces.

Return type str

terraform_workspace_new(*workspace: str*) → None

Execute `terraform workspace new` command.

permanently to https://www.terraform.io/docs/cli/commands/workspace/new.html

Parameters **workspace** – Terraform workspace to create.

terraform_workspace_select(*workspace: str*) → None

Execute `terraform workspace select` command.

https://www.terraform.io/docs/cli/commands/workspace/select.html

Parameters **workspace** – Terraform workspace to select.

terraform_workspace_show() → str

Execute `terraform workspace show` command.

https://www.terraform.io/docs/cli/commands/workspace/show.html

Returns The current Terraform workspace.

run(*action: typing_extensions.Literal[apply, destroy, get, init, plan, workspace_list, workspace_new, workspace_select, workspace_show]*) → None

Run module.

__getitem__(*key: str*) → Any

Make the object subscriptable.

Parameters **key** – Attribute to get.

__new__(***kwargs*)

class runway.module.terraform.**TerraformOptions**

Bases: *runway.module.base.ModuleOptions*

Module options for Terraform.

args

CLI arguments/options to pass to Terraform.

data

Options parsed into a data model.

env

Current deploy environment.

path

Module path.

version

String containing a Terraform version.

write_auto_tfvars

Optionally write parameters to a tfvars file instead of updating variables.

__init__(*data:* RunwayTerraformModuleOptionsDataModel, *deploy_environment:* DeployEnvironment, *path: Optional[Path] = None*) → None

Instantiate class.

Parameters

- **deploy_environment** – Current deploy environment.

- **data** – Options parsed into a data model.
- **path** – Module path.

property backend_config: *runway.module.terraform.TerraformBackendConfig*

Backend configuration options.

classmethod parse_obj(*deploy_environment:* DeployEnvironment, *obj: object*, *path: Optional[Path] = None*) → *TerraformOptions*

Parse options definition and return an options object.

Parameters

- **deploy_environment** – Current deploy environment.
- **obj** – Object to parse.
- **path** – Module path.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

class runway.module.terraform.**TerraformBackendConfig**

Bases: *runway.module.base.ModuleOptions*

Terraform backend configuration module options.

__init__(*data:* RunwayTerraformBackendConfigDataModel, *deploy_environment:* DeployEnvironment, *path: Path*) → None

Instantiate class.

Parameters

- **data** – Options parsed into a data model.
- **deploy_environment** – Current deploy environment.
- **path** – Module path.

property config_file: Optional[pathlib.Path]

Backend configuration file.

property init_args: List[str]

Return command line arguments for init.

get_full_configuration() → Dict[str, str]

Get full backend configuration.

classmethod get_backend_file(*path: pathlib.Path*, *environment: str*, *region: str*) → Optional[pathlib.Path]

Determine Terraform backend file.

Parameters

- **path** – Path to the module.
- **environment** – Current deploy environment.
- **region** – Current AWS region.

static gen_backend_filenames(*environment: str*, *region: str*) → List[str]

Generate possible Terraform backend filenames.

Parameters

- **environment** – Current deploy environment.
- **region** – Current AWS region.

classmethod parse_obj(*deploy_environment:* DeployEnvironment, *obj: object*, *path: Optional[Path] = None*) → *TerraformBackendConfig*

Parse options definition and return an options object.

Parameters

- **deploy_environment** – Current deploy environment.
- **obj** – Object to parse.
- **path** – Module path.

__eq__(*other: Any*) → bool

Assess equality.

__new__(***kwargs*)

get(*name: str*, *default: Optional[Any] = None*) → Any

Get a value or return the default.

runway.module.utils module

Runway module utilities.

runway.module.utils.**format_npm_command_for_logging**(*command: List[str]*) → str

Convert npm command list to string for display to user.

runway.module.utils.**generate_node_command**(*command: str*, *command_opts: List[str]*, *path: Path*, ***, *logger: Union[logging.Logger, 'logging.LoggerAdapter[Any]'] = <RunwayLogger runway.module.utils (WARNING)>*, *package: Optional[str] = None*) → List[str]

Return node bin command list for subprocess execution.

Parameters

- **command** – Command to execute from a local `node_modules/.bin`.
- **command_opts** – Options to include with the command.
- **path** – Current working directory. Used to construct a "fall-back" command when `npx` is not available/included as part of npm.
- **logger** – A specific logger to use when logging the constructed command.
- **package** – Name of the npm package containing the binary to execute. This is recommended when the name of the binary does not match the name of the npm package.

runway.module.utils.**run_module_command**(*cmd_list: List[str]*, *env_vars: Dict[str, str]*, *exit_on_error: bool = True*, *logger: Union[logging.Logger, 'logging.LoggerAdapter[Any]'] = <RunwayLogger runway.module.utils (WARNING)>*) → None

Shell out to provisioner command.

Parameters

- **cmd_list** – Command to run.
- **env_vars** – Environment variables.
- **exit_on_error** – If true, `subprocess.CalledProcessError` will be caught and the resulting exit code will be passed to `sys.exit()`. If false, the error will not be caught within this function. logger: Optionally, supply a logger to use.
- **logger** – A specific logger to use when logging the constructed command.

`runway.module.utils.use_npm_ci`(*path: pathlib.Path*) → bool

Return true if npm ci should be used in lieu of npm install.

runway.sources package

Empty init for python import traversal.

Submodules

runway.sources.git module

'Git type Path Source.

`class runway.sources.git.Git`

Bases: *runway.sources.source.Source*

Git Path Source.

The Git path source can be tasked with cloning a remote repository and pointing to a specific module folder (or the root).

`__init__`(*, *arguments: Optional[Dict[str, str]] = None*, *location: str = ''*, *uri: str = ''*, ***kwargs: Any*) → None

Git Path Source.

Parameters

- **arguments** – A reference can be passed along via the arguments so that a specific version of the repository is cloned. **commit**, **tag**, **branch** are all valid keys with respective output
- **location** – The relative location to the root of the repository where the module resides. Leaving this as an empty string, `/`, or `./` will have runway look in the root folder.
- **uri** – The uniform resource identifier that targets the remote git repository

`fetch`() → pathlib.Path

Retrieve the git repository from it's remote location.

`classmethod sanitize_git_path`(*path: str*) → str

Sanitize the git path for folder/file assignment.

Keyword Arguments `path` – The path string to be sanitized

__new__(***kwargs*)

static sanitize_directory_path(*uri: str*) → str

Sanitize a Source directory path string.

Parameters **uri** – The uniform resource identifier when targeting a remote resource.

runway.sources.source module

Abstract parent class for a 'Source' type object.

Allows us to specify specific remote sourced resources for out application (Git, S3, ect.)

class runway.sources.source.**Source**

Bases: object

Abstract parent class for a 'Source' type object.

The Source parent class allows us to specify remote resources for our application via services such as Git or S3. A cache directory, as part of object's configuration, is automatically created by default: `./.runway/cache`. This folder can be overridden by specifying the `cache_dir` property in the configuration passed.

Every Source type object is expected to have a `fetch` method which will return the folder path at where the module requested resides.

__init__(*, *cache_dir: Union[pathlib.Path, str]*, **_: *Any*)

Source.

Parameters **cache_dir** – The directory where the given remote resource should be cached.

fetch() → pathlib.Path

Retrieve remote source. To be implemented in each subclass.

__new__(***kwargs*)

static sanitize_directory_path(*uri: str*) → str

Sanitize a Source directory path string.

Parameters **uri** – The uniform resource identifier when targeting a remote resource.

runway.tests package

Empty init for python import traversal.

Subpackages

runway.tests.handlers package

Import classes.

Submodules

runway.tests.handlers.base module

Base module for test handlers.

class runway.tests.handlers.base.**TestHandler**

Bases: object

Base class for test handlers.

classmethod handle(*name: str*, *args: Union[*ConfigProperty, *Dict[str, Any]]*) → None

Redefine in subclass.

static get_dirs(*provided_path: str*) → List[str]

Return list of directories.

__init__()

__new__(***kwargs*)

runway.tests.handlers.cfn_lint module

cfn-list test runner.

class runway.tests.handlers.cfn_lint.**CfnLintHandler**

Bases: *runway.tests.handlers.base.TestHandler*

Lints CFN.

classmethod handle(*name: str*, *args: Union[*ConfigProperty, *Dict[str, Any]]*) → None

Perform the actual test.

Relies on .cfnlintrc file to be located beside the Runway config file.

__init__()

__new__(***kwargs*)

static get_dirs(*provided_path: str*) → List[str]

Return list of directories.

runway.tests.handlers.script module

Script test runner.

class runway.tests.handlers.script.**ScriptHandler**

Bases: *runway.tests.handlers.base.TestHandler*

Handle script tests.

Parameters commands (*List[str]*) – A list of commands to be executed in order. Each command is run in its own subprocess. The working directory will be the same as where the 'runway test' command was executed.

Example

classmethod handle(*name: str*, *args: Union[*ConfigProperty, *Dict[str, Any]]*) → None
Perform the actual test.

__init__()

__new__(***kwargs*)

static get_dirs(*provided_path: str*) → List[str]
Return list of directories.

runway.tests.handlers.yaml_lint module

yamllint test runner.

class runway.tests.handlers.yaml_lint.**YamllintHandler**
Bases: *runway.tests.handlers.base.TestHandler*

Lints yaml.

static get_yaml_files_at_path(*provided_path: str*) → List[str]
Return list of yaml files.

classmethod get_yamllint_options(*path: str*) → List[str]
Return yamllint option list.

classmethod handle(*name: str*, *args: Union[*ConfigProperty, *Dict[str, Any]]*) → None
Perform the actual test.

__init__()

__new__(***kwargs*)

static get_dirs(*provided_path: str*) → List[str]
Return list of directories.

Submodules

runway.tests.registry module

Register test handlers.

runway.tests.registry.**register_test_handler**(*test_type: str*, *handler: Type[*TestHandler*]*) → None
Register a test handler.

Parameters

- **test_type** – Name to register the handler under.
- **handler** – Test handler class.

runway.tests.registry.**unregister_test_handler**(*test_type: str*) → None
Unregister the specified test type.

This is useful when testing various lookup types if you want to unregister the lookup type after the test runs.

Parameters **test_type** (*str*) – Name of the lookup type to unregister.

runway.utils package

Utility functions.

class runway.utils.**BaseModel**

Bases: pydantic.main.BaseModel

Base class for Runway models.

get(*name: str*, *default: Optional[Any] = None*) → Any

Safely get the value of an attribute.

Parameters

- **name** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

__contains__(*name: object*) → bool

Implement evaluation of 'in' conditional.

Parameters **name** – The name to check for existence in the model.

__getitem__(*name: str*) → Any

Implement evaluation of self[name].

Parameters **name** – Attribute name to return the value for.

Returns The value associated with the provided name/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

__setitem__(*name: str*, *value: Any*) → None

Implement item assignment (e.g. `self[name] = value`).

Parameters

- **name** – Attribute name to set.
- **value** – Value to assign to the attribute.

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises ValidationError if the input data cannot be parsed to form a valid model.

__iter__() → TupleGenerator

so *dict(model)* works

__new__(***kwargs*)

__pretty__(*fmt: Callable[[Any], Any]*, ***kwargs: Any*) → Generator[Any, None, None]

Used by devtools (https://python-devtools.helpmanual.io/) to provide a human readable representations of objects

__repr_name__() → unicode

Name of the instance's class, used in __repr__.

__rich_repr__() → RichReprResult

Get fields for Rich library

classmethod __try_update_forward_refs__(***localns: Any*) → None

Same as update_forward_refs but will not raise exception when forward references are not defined.

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *update: Optional[DictStrAny] = None*, *deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

json(***, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → unicode

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

class runway.utils.**JsonEncoder**

Bases: `json.encoder.JSONEncoder`

Encode Python objects to JSON data.

This class can be used with `json.dumps()` to handle most data types that can occur in responses from AWS.

Usage:

```
>>> json.dumps(data, cls=JsonEncoder)
```

default(*o: Any*) → Any

Encode types not supported by the default JSONEncoder.

Parameters **o** – Object to encode.

Returns JSON serializable data type.

Raises `TypeError` – Object type could not be encoded.

__init__(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

__new__(**kwargs*)

encode(*o*)

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

iterencode(*o*, *_one_shot=False*)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

class runway.utils.**MutableMap**

Bases: `MutableMapping`[`str`, `Any`]

Base class for mutable map objects.

__init__(**kwargs: Any*) → None

Initialize class.

Provided `kwargs` are added to the object as attributes.

Example

property data: Dict[str, Any]

Sanitized output of __dict__.

Removes anything that starts with _.

clear_found_cache() → None

Clear _found_cache.

find(*query: str*, *default: Optional[Any] = None*, *ignore_cache: bool = False*) → Any

Find a value in the map.

Previously found queries are cached to increase search speed. The cached value should only be used if values are not expected to change.

Parameters

- **query** – A period delimited string that is split to search for nested values
- **default** – The value to return if the query was unsuccessful.
- **ignore_cache** – Ignore cached value.

get(*key: str*, *default: Optional[Any] = None*) → Any

Implement evaluation of self.get.

Parameters

- **key** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

__bool__() → bool

Implement evaluation of instances as a bool.

__contains__(*value: Any*) → bool

Implement evaluation of 'in' conditional.

__getitem__(*key: str*) → Any

Implement evaluation of self[key].

Parameters **key** – Attribute name to return the value for.

Returns The value associated with the provided key/attribute name.

Raises **AttributeError** – If attribute does not exist on this object.

Example

__setitem__(*key: str*, *value: Any*) → None

Implement assignment to self[key].

Parameters

- **key** – Attribute name to associate with a value.
- **value** – Value of a key/attribute.

Example

__delitem__(*key: str*) → None

Implement deletion of self[key].

Parameters **key** – Attribute name to remove from the object.

Example

__len__() → int

Implement the built-in function len().

Example

__iter__() → Iterator[Any]

Return iterator object that can iterate over all attributes.

Example

__str__() → str

Return string representation of the object.

__new__(***kwargs*)

clear() → None. Remove all items from D.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class runway.utils.**SafeHaven**

Bases: ContextManager[*SafeHaven*]

Context manager that caches and resets important values on exit.

Caches and resets os.environ, sys.argv, sys.modules, and sys.path.

__init__(*argv: Optional[Iterable[str]] = None*, *environ: Optional[Dict[str, str]] = None*, *sys_modules_exclude: Optional[Iterable[str]] = None*, *sys_path: Optional[Iterable[str]] = None*) → None

Instantiate class.

Parameters

- **argv** – Override the value of sys.argv.
- **environ** – Update os.environ.
- **sys_modules_exclude** – A list of modules to exclude when reverting sys.modules to its previous state. These are checked as `module.startswith(name)`.
- **sys_path** – Override the value of sys.path.

reset_all() → None

Reset all values cached by this context manager.

reset_os_environ() → None

Reset the value of os.environ.

reset_sys_argv() → None

Reset the value of sys.argv.

reset_sys_modules() → None

Reset the value of sys.modules.

reset_sys_path() → None

Reset the value of sys.path.

__enter__() → *runway.utils.SafeHaven*

Enter the context manager.

Returns Instance of the context manager.

Return type *SafeHaven*

__exit__(*exc_type: Optional[Type[BaseException]]*, *exc_value: Optional[BaseException]*, *traceback: Optional[types.TracebackType]*) → None

Exit the context manager.

__new__(***kwargs*)

class runway.utils.**YamlDumper**

Bases: `yaml.dumper.Dumper`

Custom YAML Dumper.

This Dumper allows for YAML to be output to follow YAML spec 1.2, example 2.3 of collections (2.1). This provides an output that is more humanreadable and complies with yamllint.

Example

```
>>> print(yaml.dump({'key': ['val1', 'val2']}, Dumper=YamlDumper))
```

Note: YAML 1.2 Specification: https://yaml.org/spec/1.2/spec.html used for reference.

increase_indent(*flow: bool = False*, *indentless: bool = False*) → None

Override parent method.

__init__(*stream*, *default_style=None*, *default_flow_style=False*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*, *sort_keys=True*)

__new__(***kwargs*)

runway.utils.**argv**(**args: str*) → Iterator[None]

Context manager for temporarily changing sys.argv.

runway.utils.**change_dir**(*newdir: Union[pathlib.Path, str]*) → Iterator[None]

Change directory.

Adapted from http://stackoverflow.com/a/24176022

Parameters **newdir** – Path to change directory into.

runway.utils.**ensure_file_is_executable**(*path: str*) → None

Exit if file is not executable.

Parameters **path** – Path to file.

Raises **SystemExit** – file is not executable.

runway.utils.**ensure_string**(*value: Any*) → str

Ensure value is a string.

runway.utils.**environ**(*env: Optional[Dict[str, str]] = None*, ***kwargs: str*) → Iterator[None]

Context manager for temporarily changing os.environ.

The original value of os.environ is restored upon exit.

Parameters **env** – Dictionary to use when updating os.environ.

runway.utils.**json_serial**(*obj: Any*) → Any

JSON serializer for objects not serializable by default json code.

runway.utils.**load_object_from_string**(*fqcn: str*, *try_reload: bool = False*) → Union[type, Callable[[...], Any]]

Convert "." delimited strings to a python object.

Parameters

- **fqcn** – A "." delimited string representing the full path to an object (function, class, variable) inside a module
- **try_reload** – Try to reload the module so any global variables set within the file during import are reloaded. This only applies to modules that are already imported and are not builtin.

Returns Object being imported from the provided path.

Example:

```
load_object_from_string("os.path.basename")
load_object_from_string("logging.Logger")
load_object_from_string("LocalClassName")
```

runway.utils.**merge_dicts**(*dict1: Dict[Any, Any]*, *dict2: Dict[Any, Any]*, *deep_merge: bool = True*) → Dict[str, Any]

runway.utils.**merge_dicts**(*dict1: List[Any]*, *dict2: List[Any]*, *deep_merge: bool = True*) → List[Any]

Merge dict2 into dict1.

runway.utils.**snake_case_to_kebab_case**(*value: str*) → str

Convert snake_case to kebab-case.

Parameters **value** – The string value to convert.

runway.utils.**extract_boto_args_from_env**(*env_vars: Dict[str, str]*) → Dict[str, str]

Return boto3 client args dict with environment creds.

runway.utils.**flatten_path_lists**(*env_dict: Dict[str, Any]*, *env_root: Optional[str] = None*) → Dict[str, Any]

Join paths in environment dict down to strings.

runway.utils.**merge_nested_environment_dicts**(*env_dicts: Dict[str, Any]*, *env_name: Optional[str] = None*, *env_root: Optional[str] = None*) → Dict[str, Any]

Return single-level dictionary from dictionary of dictionaries.

runway.utils.**find_cfn_output**(*key: str*, *outputs: List[OutputTypeDef]*) → Optional[str]

Return CFN output value.

Parameters

- **key** – Name of the output.
- **outputs** – List of Stack outputs.

runway.utils.**get_embedded_lib_path**() → str

Return path of embedded libraries.

runway.utils.**get_hash_for_filename**(*filename: str*, *hashfile_path: str*) → str

Return hash for filename in the hashfile.

runway.utils.**ignore_exit_code_0**() → Iterator[None]

Capture exit calls and ignore those with exit code 0.

runway.utils.**fix_windows_command_list**(*commands: List[str]*) → List[str]

Return command list with working Windows commands.

npm on windows is npm.cmd, which will blow up subprocess.check_call([‘npm’, ‘...’])

Similar issues arise when calling python apps that will have a windows-only suffix applied to them.

runway.utils.**run_commands**(*commands: List[Union[str, List[str], Dict[str, Union[str, List[str]]]]]*, *directory: Union[pathlib.Path, str]*, *env: Optional[Dict[str, str]] = None*) → None

Run list of commands.

runway.utils.**get_file_hash**(*filename: str*, *algorithm: typing_extensions.Literal[blake2b, md5, sha1, sha224, sha256, sha3_224, sha3_256, sha3_384, sha3_512, sha384, sha512, shake_128, shake_256]*) → str

Return cryptographic hash of file.

Deprecated since version 2.4.0: Use `runway.utils.FileHash` instead.

`runway.utils.`**`md5sum`**(*filename: str*) → str

Return MD5 hash of file.

Deprecated since version 2.4.0: Use `runway.utils.FileHash` instead.

`runway.utils.`**`sha256sum`**(*filename: str*) → str

Return SHA256 hash of file.

Deprecated since version 2.4.0: Use `runway.utils.FileHash` instead.

`runway.utils.`**`use_embedded_pkgs`**(*embedded_lib_path: Optional[str] = None*) → Iterator[None]

Temporarily prepend embedded packages to sys.path.

`runway.utils.`**`which`**(*program: str*) → Optional[str]

Mimic 'which' command behavior.

Submodules

runway.compat module

Python dependency compatibility handling.

exception `runway.compat.`**`PackageNotFoundError`**

Bases: `ModuleNotFoundError`

The package was not found.

`__init__`(**args*, ***kwargs*)

`__new__`(***kwargs*)

`msg`

exception message

`name`

module name

`path`

module path

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`runway.compat.`**`shlex_join`**(*split_command*)

Return a shell-escaped string from *split_command*.

`runway.compat.`**`version`**(*distribution_name*)

Get the version string for the named package.

Parameters **`distribution_name`** – The name of the distribution package to query.

Returns The version string for the package as defined in the package's "Version" metadata key.

runway.constants module

Runway constants.

runway.constants.**BOTO3_CREDENTIAL_CACHE: Dict[str, Any] = {}**

A global credential cache that can be shared among boto3 sessions. This is inherently threadsafe thanks to the GIL. (https://docs.python.org/3/glossary.html#term-global-interpreter-lock)

runway.exceptions module

Runway exceptions.

exception runway.exceptions.**RunwayError**

Bases: Exception

Base class for custom exceptions raised by Runway.

message: str

Error message.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**ConfigNotFound**

Bases: *runway.exceptions.RunwayError*

Configuration file could not be found.

__init__(*, *looking_for: Optional[List[str]] = None*, *path: pathlib.Path*) → None

Instantiate class.

Parameters

- **path** – Path where the config file was expected to be found.
- **looking_for** – List of file names that were being looked for.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**DockerConnectionRefusedError**

Bases: *runway.exceptions.RunwayError*

Docker connection refused.

This can be caused by a number of reasons:

- Docker is not installed.
- Docker service is not running.
- The current user does not have adequate permissions (e.g. not a member of the `docker` group).

__init__() → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**DockerExecFailedError**

Bases: *runway.exceptions.RunwayError*

Docker failed when trying to execute a command.

This can be used for `docker container run`, `docker container start` and `docker exec` equivalents.

__init__(*response: Dict[str, Any]*) → None

Instantiate class.

Parameters **response** – The return value of `docker.models.containers.Container.wait()`, Docker API's response as a Python dictionary. This can contain important log information pertinent to troubleshooting that may not streamed.

exit_code: int

The `StatusCode` returned by Docker.

message: str

Error message.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**FailedLookup**

Bases: *runway.exceptions.RunwayError*

Intermediary Exception to be converted to FailedVariableLookup.

Should be caught by error handling and `runway.cfngin.exceptions.FailedVariableLookup` raised instead to construct a proper error message.

__init__(*lookup:* VariableValueLookup, *cause: Exception*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **lookup** – The variable value lookup that was attempted and resulted in an exception being raised.
- **cause** – The exception that was raised.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**FailedVariableLookup**

Bases: *runway.exceptions.RunwayError*

Lookup could not be resolved.

Raised when an exception is raised when trying to resolve a lookup.

__init__(*variable:* Variable, *lookup_error:* FailedLookup, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters

- **variable** – The variable containing the failed lookup.
- **lookup_error** – The exception that was raised directly before this one.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**HclParserError**

Bases: *runway.exceptions.RunwayError*

HCL/HCL2 parser error.

__init__(*exc: Exception*, *file_path: Union[Path, str]*, *parser: Optional[ModuleType] = None*) → None

Instantiate class.

Parameters

- **exc** – Exception that was raised.
- **file_path** – File that resulted in the error.
- **parser** – The parser what was used to try to parse the file (hcl|hcl2).

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**InvalidLookupConcatenation**

Bases: *runway.exceptions.RunwayError*

Invalid return value for a concatenated (chained) lookup.

The return value must be a string when lookups are concatenated.

__init__(*invalid_lookup:* VariableValue, *concat_lookups:* VariableValueConcatenation*[Any]*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**KubectlVersionNotSpecified**

Bases: *runway.exceptions.RunwayError*

kubectl version is required but was not specified.

Version can be specified by using a file or option.

__init__(**args: Any*, ***kwargs: Any*) → None

Instantiate class.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**NpmNotFound**
Bases: *runway.exceptions.RunwayError*

Raised when npm could not be executed or was not found in path.

__init__(**args: Any*, ***kwargs: Any*) → None
Instantiate class.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception runway.exceptions.**OutputDoesNotExist**
Bases: *runway.exceptions.RunwayError*

Raised when a specific stack output does not exist.

__init__(*stack_name: str*, *output: str*, **args: Any*, ***kwargs: Any*) → None
Instantiate class.

Parameters

- **stack_name** – Name of the stack.
- **output** – The output that does not exist.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

stack_name: *str*
Name of a CloudFormation Stack.

output: *str*
Name of the CloudFormation Stack's Output that does not exist.

message: *str*
Error message.

exception runway.exceptions.**RequiredTagNotFoundError**
Bases: *runway.exceptions.RunwayError*

Required tag not found on resource.

__new__(***kwargs*)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*resource: str*, *tag_key: str*) → None
Instantiate class.

Parameters

- **resource** – An ID or name to identify a resource.
- **tag_key** – Key of the tag that could not be found.

resource: str

An ID or name to identify a resource.

tag_key: str

Key of the tag that could not be found.

message: str

Error message.

exception runway.exceptions.**UnknownLookupType**

Bases: *runway.exceptions.RunwayError*

Lookup type provided does not match a registered lookup.

Example

If a lookup of `${<lookup_type> query}` is used and `<lookup_type>` is not a registered lookup, this exception will be raised.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*lookup:* VariableValueLookup, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **lookup** – Variable value lookup that could not find a handler.

exception runway.exceptions.**UnresolvedVariable**

Bases: *runway.exceptions.RunwayError*

Raised when trying to use a variable before it has been resolved.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*variable:* Variable, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **variable** – The unresolved variable.

exception runway.exceptions.**UnresolvedVariableValue**

Bases: *runway.exceptions.RunwayError*

Intermediary Exception to be converted to UnresolvedVariable.

Should be caught by error handling and `runway.cfngin.exceptions.UnresolvedVariable` raised instead to construct a proper error message.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*lookup: VariableValueLookup*, **args: Any*, ***kwargs: Any*) → None

Instantiate class.

Parameters **lookup** – The variable value lookup that is not resolved.

exception runway.exceptions.**VariablesFileNotFound**

Bases: *runway.exceptions.RunwayError*

Defined variables file could not be found.

__new__(***kwargs*)

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

__init__(*file_path: pathlib.Path*) → None

Instantiate class.

Parameters **file_path** – Path where the file was expected to be found.

runway.mixins module

Class mixins.

class runway.mixins.**CliInterfaceMixin**

Bases: object

Mixin for adding CLI interface methods.

EXECUTABLE: ClassVar[str]

CLI executable.

ctx: Union[*CfnginContext*, *RunwayContext*]

CFNgin or Runway context object.

cwd: Path

Working directory where commands will be run.

static convert_to_cli_arg(*arg_name: str*, ***, *prefix: str = '--'*) → str

Convert string kwarg name into a CLI argument.

classmethod found_in_path() → bool

Determine if executable is found in $PATH.

classmethod generate_command(*command: Union[List[str], str]*, ***kwargs: Optional[Union[bool, Iterable[str], str]]*) → List[str]

Generate command to be executed and log it.

Parameters

- **command** – Command to run.
- **args** – Additional args to pass to the command.

Returns The full command to be passed into a subprocess.

static list2cmdline(*split_command: Iterable[str]*) → str

Combine a list of strings into a string that can be run as a command.

Handles multi-platform differences.

__init__()

__new__(***kwargs*)

class runway.mixins.**DelCachedPropMixin**

Bases: object

Mixin to handle safely clearing the value of functools.cached_property().

__init__()

__new__(***kwargs*)

runway.s3_utils module

Utility functions for S3.

runway.s3_utils.**purge_and_delete_bucket**(*bucket_name: str*, *region: str = 'us-east-1'*, *session: Optional[boto3.session.Session] = None*) → None

Delete all objects and versions in bucket, then delete bucket.

runway.s3_utils.**purge_bucket**(*bucket_name: str*, *region: str = 'us-east-1'*, *session: Optional[boto3.session.Session] = None*) → None

Delete all objects and versions in bucket.

runway.s3_utils.**delete_bucket**(*bucket_name: str*, *region: str = 'us-east-1'*, *session: Optional[boto3.session.Session] = None*) → None

Delete bucket.

runway.s3_utils.**does_bucket_exist**(*bucket_name: str*, *region: str = 'us-east-1'*, *session: Optional[boto3.session.Session] = None*) → bool

Check if bucket exists in S3.

runway.s3_utils.**ensure_bucket_exists**(*bucket_name: str*, *region: str = 'us-east-1'*, *session: Optional[boto3.session.Session] = None*) → None

Ensure S3 bucket exists.

runway.s3_utils.**does_s3_object_exist**(*bucket: str*, *key: str*, *session: Optional[boto3.session.Session] = None*, *region: str = 'us-east-1'*) → bool

Determine if object exists on s3.

runway.s3_utils.**upload**(*bucket: str*, *key: str*, *filename: str*, *session: Optional[boto3.session.Session] = None*) → None

Upload file to S3 bucket.

runway.s3_utils.**download**(*bucket: str*, *key: str*, *file_path: str*, *session: Optional[boto3.session.Session] = None*) → str

Download a file from S3 to the given path.

runway.s3_utils.**download_and_extract_to_mkdtemp**(*bucket: str*, *key: str*, *session: Optional[boto3.session.Session] = None*) → str

Download zip archive and extract it to temporary directory.

runway.s3_utils.**get_matching_s3_objects**(*bucket: str*, *prefix: Sequence[str] = ''*, *suffix: str = ''*, *session: Optional[boto3.Session] = None*) → Iterator[ObjectTypeDef]

Generate objects in an S3 bucket.

Parameters

- **bucket** – Name of the S3 bucket.
- **prefix** – Only fetch objects whose key starts with this prefix (optional).
- **suffix** – Only fetch objects whose keys end with this suffix (optional).
- **session** – Boto3/botocore session.

runway.s3_utils.**get_matching_s3_keys**(*bucket: str*, *prefix: str = ''*, *suffix: str = ''*, *session: Optional[boto3.session.Session] = None*) → Iterator[str]

Generate the keys in an S3 bucket.

Parameters

- **bucket** – Name of the S3 bucket.
- **prefix** – Only fetch keys that start with this prefix (optional).
- **suffix** – Only fetch keys that end with this suffix (optional).
- **session** – Boto3/botocore session.

runway.type_defs module

Type definitions.

class runway.type_defs.**Boto3CredentialsTypeDef**

Bases: typing_extensions.TypedDict

Boto3 credentials.

class runway.type_defs.**EnvVarsAwsCredentialsTypeDef**

Bases: typing_extensions.TypedDict

AWS credentials from/for environment variables.

runway.variables module

Runway variables.

class runway.variables.**Variable**

Bases: object

Represents a variable provided to a Runway directive.

__init__(*name: str*, *value: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Initialize class.

Parameters

- **name** – Name of the variable (directive/key).
- **value** – The variable itself.
- **variable_type** – Type of variable (cfngin|runway).

property dependencies: Set[str]

Stack names that this variable depends on.

Returns Stack names that this variable depends on.

Return type Set[str]

property resolved: bool

Boolean for whether the Variable has been resolved.

Variables only need to be resolved if they contain lookups.

property value: Any

Return the current value of the Variable.

Raises ***UnresolvedVariable*** – Value accessed before it have been resolved.

resolve(*context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*, *variables: Optional[RunwayVariablesDefinition] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

Raises ***FailedVariableLookup*** –

get(*key: str*, *default: Optional[Any] = None*) → Any

Implement evaluation of self.get.

Parameters

- **key** – Attribute name to return the value for.
- **default** – Value to return if attribute is not found.

__repr__() → str

Return object representation.

__new__(***kwargs*)

runway.variables.**resolve_variables**(*variables: List[Variable]*, *context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*) → None

Given a list of variables, resolve all of them.

Parameters

- **variables** – List of variables.
- **context** – CFNgin context.
- **provider** – Subclass of the base provider.

class runway.variables.**VariableValue**

Bases: object

Syntax tree base class to parse variable values.

property dependencies: Set[Any]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

Raises NotImplementedError – Should be defined in a subclass.

property simplified: Any

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

Should be implimented in subclasses where applicable.

property value: Any

Value of the variable. Can be resolved or unresolved.

Raises NotImplementedError – Should be defined in a subclass.

resolve(*context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*, *variables: Optional[RunwayVariablesDefinition] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

classmethod parse_obj(*obj: runway.variables._PydanticModelTypeVar*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValuePydanticModel*[runway.variables._PydanticModelTypeVar]

classmethod parse_obj(*obj: Dict[str, Any]*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

classmethod parse_obj(*obj: List[Any]*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValueList*

classmethod parse_obj(*obj: int*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValueLiteral*[int]

classmethod parse_obj(*obj: str*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValueConcatenation*[Union[*runway.variables.VariableValueLiteral*[str], *runway.variables.VariableValueLookup*]]

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.
- **variable_type** – Type of variable (cfngin|runway).

__iter__() → Iterator[Any]

How the object is iterated.

Raises NotImplementedError – Should be defined in a subclass.

__repr__() → str

Return object representation.

Raises NotImplementedError – Should be defined in a subclass.

__init__()

__new__(**kwargs)

class runway.variables.VariableValueDict

Bases: runway.variables.VariableValue, MutableMapping[str, runway.variables.VariableValue]

A dict variable value.

__init__(data: Dict[str, Any], variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin') → None

Instantiate class.

Parameters

- **data** – Data to be stored in the object.
- **variable_type** – Type of variable (cfngin|runway).

property dependencies: Set[str]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

property simplified: Dict[str, Any]

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

property value: Dict[str, Any]

Value of the variable. Can be resolved or unresolved.

resolve(context: Union[CfnginContext, RunwayContext], provider: Optional[Provider] = None, variables: Optional[RunwayVariablesDefinition] = None, **kwargs: Any) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

__delitem__(_VariableValueDict__key: str) → None

Delete item by index.

__getitem__(_VariableValueDict__key: str) → runway.variables.VariableValue

Get item by index.

__iter__() → Iterator[str]

How the object is iterated.

__len__() → int

Length of the object.

__repr__() → str

Return object representation.

__setitem__(*_VariableValueDict__key: str*, *_VariableValueDict__value:* runway.variables.VariableValue) → None

Set item by index.

__new__(***kwargs*)

clear() → None. Remove all items from D.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.
- **variable_type** – Type of variable (cfngin|runway).

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class runway.variables.**VariableValueList**

Bases: *runway.variables.VariableValue*, MutableSequence[*runway.variables.VariableValue*]

List variable value.

__init__(*iterable: Iterable[Any]*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Instantiate class.

Parameters

- **iterable** – Data to store in the iterable.
- **variable_type** – Type of variable (cfngin|runway).

property dependencies: Set[str]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

property simplified: List[*runway.variables.VariableValue*]

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

property value: List[Any]

Value of the variable. Can be resolved or unresolved.

insert(*index: int*, *value:* runway.variables.VariableValue) → None

Insert a value at a specific index.

resolve(*context: Union[*CfnginContext, RunwayContext*]*, *provider: Optional[*Provider*] = None*, *variables: Optional[*RunwayVariablesDefinition*] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

__delitem__(*_VariableValueList__index: int*) → None

Delete item by index.

__getitem__(*__index: int*) → *runway.variables.VariableValue*

__getitem__(*__index: slice*) → List[*runway.variables.VariableValue*]

Get item by index.

__setitem__(*__index: int*, *__value:* runway.variables.VariableValue) → None

__setitem__(*__index: slice*, *__value: List[*runway.variables.VariableValue*]*) → None

Set item by index.

__iter__() → Iterator[*runway.variables.VariableValue*]

Object iteration.

__len__() → int

Length of the object.

__repr__() → str

Object string representation.

__new__(***kwargs*)

append(*value*)

S.append(value) – append value to the end of the sequence

clear() → None -- remove all items from S

count(*value*) → integer -- return number of occurrences of value

extend(*values*)

S.extend(iterable) – extend sequence by appending elements from the iterable

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.
- **variable_type** – Type of variable (cfngin|runway).

pop([*index*]) → item -- remove and return item at index (default last).

Raise IndexError if list is empty or index is out of range.

remove(*value*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

reverse()

S.reverse() – reverse *IN PLACE*

class runway.variables.**VariableValueLiteral**

Bases: Generic[runway.variables._LiteralValue], *runway.variables.VariableValue*

The literal value of a variable as provided.

__init__(*value: runway.variables._LiteralValue*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Instantiate class.

Parameters

- **value** – Data to store in the object.
- **variable_type** – Type of variable (cfngin|runway).

property resolved: bool

Use to check if the variable value has been resolved.

The ValueLiteral will always appear as resolved because it does not "resolve" since it is the literal definition of the value.

property value: runway.variables._LiteralValue

Value of the variable.

__iter__() → Iterator[Any]

How the object is iterated.

__repr__() → str

Return object representation.

__new__(***kwargs*)

property dependencies: Set[Any]

Stack names that this variable depends on.

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.

- **variable_type** – Type of variable (cfngin|runway).

resolve(*context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*, *variables: Optional[RunwayVariablesDefinition] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

property simplified: Any

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

Should be implimented in subclasses where applicable.

class runway.variables.**VariableValueConcatenation**

Bases: Generic[runway.variables._VariableValue], *runway.variables.VariableValue*

A concatenated variable values.

__init__(*iterable: Iterable[runway.variables._VariableValue]*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Instantiate class.

Parameters

- **iterable** – Data to store in the iterable.
- **variable_type** – Type of variable (cfngin|runway).

property dependencies: Set[str]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

property simplified: *runway.variables.VariableValue*

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

property value: Any

Value of the variable. Can be resolved or unresolved.

Raises ***InvalidLookupConcatenation*** –

resolve(*context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*, *variables: Optional[RunwayVariablesDefinition] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

__delitem__(*_VariableValueConcatenation__index: int*) → None

Delete item by index.

__getitem__(*__index: int*) → runway.variables._VariableValue

__getitem__(*__index: slice*) → List[runway.variables._VariableValue]

Get item by index.

__setitem__(*__index: int*, *__value: runway.variables._VariableValue*) → None

__setitem__(*__index: slice*, *__value: List[runway.variables._VariableValue]*) → None

Set item by index.

__iter__() → Iterator[runway.variables._VariableValue]

Object iteration.

__len__() → int

Length of the object.

__repr__() → str

Return object representation.

__new__(***kwargs*)

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.
- **variable_type** – Type of variable (cfngin|runway).

class runway.variables.**VariableValueLookup**

Bases: *runway.variables.VariableValue*

A lookup variable value.

__init__(*lookup_name:* runway.variables.VariableValueLiteral*[str]*, *lookup_query: Union[str,* runway.variables.VariableValue*]*, *handler: Optional[Type[*runway.lookups.handlers.base.LookupHandler*]] = None*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Initialize class.

Parameters

- **lookup_name** – Name of the invoked lookup.
- **lookup_query** – Data portion of the lookup.
- **handler** – Lookup handler that will be use to resolve the value.
- **variable_type** – Type of variable (cfngin|runway).

Raises

- ***UnknownLookupType*** – Invalid lookup type.
- **ValueError** – Invalid value for variable_type.

property dependencies: Set[str]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

property simplified: *runway.variables.VariableValueLookup*

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

property value: Any

Value of the variable. Can be resolved or unresolved.

Raises *UnresolvedVariableValue* – Value accessed before it has been resolved.

resolve(*context: Union[*CfnginContext, RunwayContext*]*, *provider: Optional[*Provider*] = None*, *variables: Optional[*RunwayVariablesDefinition*] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

Raises *FailedLookup* – A lookup failed for any reason.

__iter__() → Iterator[*runway.variables.VariableValueLookup*]

How the object is iterated.

__new__(***kwargs*)

__repr__() → str

Return object representation.

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.
- **variable_type** – Type of variable (cfngin|runway).

__str__() → str

Object displayed as a string.

class runway.variables.**VariableValuePydanticModel**

Bases: Generic[runway.variables._PydanticModelTypeVar], *runway.variables.VariableValue*

A pydantic model variable value.

__new__(***kwargs*)

classmethod parse_obj(*obj: Any*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → *runway.variables.VariableValue*

Parse complex variable structures using type appropriate subclasses.

Parameters

- **obj** – The objected defined as the value of a variable.

- **variable_type** – Type of variable (cfngin|runway).

__init__(*data: runway.variables._PydanticModelTypeVar*, *variable_type: typing_extensions.Literal[cfngin, runway] = 'cfngin'*) → None

Instantiate class.

Parameters

- **data** – Data to be stored in the object.
- **variable_type** – Type of variable (cfngin|runway).

property dependencies: Set[str]

Stack names that this variable depends on.

property resolved: bool

Use to check if the variable value has been resolved.

property simplified: Dict[str, Any]

Return a simplified version of the value.

This can be used to concatenate two literals into one literal or flatten nested concatenations.

property value: runway.variables._PydanticModelTypeVar

Value of the variable. Can be resolved or unresolved.

Uses the original pydantic model class to parse the resolved data back into a pydantic model.

resolve(*context: Union[CfnginContext, RunwayContext]*, *provider: Optional[Provider] = None*, *variables: Optional[RunwayVariablesDefinition] = None*, ***kwargs: Any*) → None

Resolve the variable value.

Parameters

- **context** – The current context object.
- **provider** – Subclass of the base provider.
- **variables** – Object containing variables passed to Runway.

__delitem__(*_VariableValuePydanticModel__key: str*) → None

Delete item by index.

__getitem__(*_VariableValuePydanticModel__key: str*) → *runway.variables.VariableValue*

Get item by index.

__iter__() → Iterator[str]

How the object is iterated.

__len__() → int

Length of the object.

__repr__() → str

Return object representation.

__setitem__(*_VariableValuePydanticModel__key: str*, *_VariableValuePydanticModel__value: runway.variables.VariableValue*) → None

Set item by index.

10.6.5 Advanced Configuration

This section has been placed in the **Developer Guide** because it details advanced configuration that should only be used by those with intimate knowledge of how Runway works.

Environment Variables

Environment variables can be used to alter the functionality of Runway.

CI: Any

If not *undefined*, Runway will operate in non-iterative mode.

DEBUG: int = 0

Used to select the debug logs to display

- `0` or not defined with show no debug logs
- `1` will show Runway's debug logs
- `2` will show Runway's debug logs and some dependency debug logs (e.g. botocore)

New in version 1.10.0.

DEPLOY_ENVIRONMENT: str

Explicitly define the deploy environment.

New in version 1.3.4.

CFNGIN_STACK_POLL_TIME: int = 30

Number of seconds between CloudFormation API calls. Adjusting this will impact API throttling.

RUNWAY_COLORIZE: str

Explicitly enable/disable colorized output for *CDK*, *Serverless*, and *Terraform* modules. Having this set to a truthy value will prevent `-no-color`/`--no-color` from being added to any commands even if stdout is not a TTY. Having this set to a falsy value will include `-no-color`/`--no-color` in commands even if stdout is a TTY. If the IaC tool has other mechanisms for disabling color output, using a truthy value will not circumvent them.

Truthy values are `y`, `yes`, `t`, `true`, `on` and `1`. Falsy values are `n`, `no`, `f`, `false`, `off` and `0`. Raises `ValueError` if anything else is used.

New in version 1.8.1.

RUNWAY_MAX_CONCURRENT_MODULES: int

Max number of modules that can be deployed to concurrently. (*default:* `min(61, os.cpu_count())`)

On Windows, this must be equal to or lower than `61`.

IMPORTANT: When using `parallel_regions` and `child_modules` together, please consider the nature of their relationship when manually setting this value. (`parallel_regions * child_modules`)

New in version 1.4.3.

RUNWAY_MAX_CONCURRENT_REGIONS: int

Max number of regions that can be deployed to concurrently. (*default:* `min(61, os.cpu_count())`)

On Windows, this must be equal to or lower than `61`.

IMPORTANT: When using `parallel_regions` and `child_modules` together, please consider the nature of their relationship when manually setting this value. (`parallel_regions * child_modules`)

New in version 1.4.3.

RUNWAY_LOG_FIELD_STYLES: str

Can be provided to customize the styling (color, bold, etc) used for LogRecord attributes (except for message). By default, Runway does not apply style to fields. For information on how to format the value, see the documentation provided by coloredlogs.

New in version 1.10.0.

RUNWAY_LOG_FORMAT: str

Can be provided to use a custom log message format. The value should be a format string using %-formatting. In addition to being able to use LogRecord attributes in the string, Runway provides the additional fields of `%(hostname)s` and `%(programname)s`.

If not provided, `[%(programname)s] %(message)s` is used unless using debug, verbose or no color. In that case, `%(levelname)s:%(name)s:%(message)s` is used.

New in version 1.10.0.

RUNWAY_LOG_LEVEL_STYLES: str

Can be provided to customize the styling (color, bold, etc) used for log messages sent to each log level. If provided, the parsed value will be merged with Runway's default styling. For information on how to format the value, see the documentation provided by coloredlogs.

New in version 1.10.0.

RUNWAY_NO_COLOR: Any

Disable Runway's colorized logs. Providing this will also change the log format to `%(levelname)s:%(name)s:%(message)s`.

New in version 1.8.1.

VERBOSE: Any

If not *undefined*, Runway will display verbose logs and change the logging format to `%(levelname)s:%(name)s:%(message)s`.

New in version 1.10.0.

10.6.6 Contribution Requirements

Please review our *Getting Started Guide for Developers* before working on your contribution. It contains information on setting up a development environment to make following some of these requirements easier.

Branch Requirements

Branches must start with one of the following prefixes (e.g. `<prefix>/<your-branch-name>`). This is due to how labels are applied to PRs. If the branch does not meet the requirement, any PRs from it will be blocked from being merged.

bugfix | fix | hotfix The branch contains a fix for a big.

feature | feat The branch contains a new feature or enhancement to an existing feature.

docs | documentation The branch only contains updates to documentation.

maintain | maint | maintenance The branch does not contain changes to the project itself to is aimed at maintaining the repo, CI/CD, or testing infrastructure. (e.g. README, GitHub action, integration test infrastructure)

release Reserved for maintainers to prepare for the release of a new version.

Documentation Requirements

Docstrings

In general, we loosely follow the *Google Python Style Guide for Comments and Docstrings*.

We use the `napoleon` extension for Sphinx to parse docstrings. Napoleon provides an Example Google Style Python Docstring that can be referenced.

reStructuredText Formatting

In general, we loosely follow the Python Style Guide for documentation.

Note: Not all documentation pages follow this yet. If the page you are updating deviates from this format, adapt to the format of the page.

Headings

Section headers are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text.

1. # with overline, for **h1** (generally only one per file, at the top of the file)
2. * with overline, for **h2**
3. =, for **h3**
4. -, for **h4**
5. ^, for **h5**
6. ", for **h6**

h1 and **h2** should have two blank lines separating them from sections with headings of the same level or higher.

A ''rubric" directive can be used to create a non-indexed heading.

Example

```
#########
Heading 1
#########


*********
Heading 2
*********

Heading 3
=========

Heading 4
---------
```

(continues on next page)

(continued from previous page)

```
Heading 5
^^^^^^^^^

Heading 6
"""""""""""

.. rubric:: Non-indexed Heading


*********
Heading 2
*********

Heading 3
=========
```

PR Requirements

In order for a PR to be merged it must be passing all checks and be approved by at least one maintainer. Some of the checks can be run locally using `make lint` and `make test`.

To be considered for approval, the PR must meet the following requirements.

- PR title must be a brief explanation of what was done in the PR (think commit message).
- PR body must comply with the formatting and prompts provided in the template (automatically applied when creating a PR on GitHub). At a minimum the following should be provided in the body of the PR:
  - A summary of was done.
  - Explain why this change is needed.
  - Detail the changes that were made (think CHANGELOG).
- Include tests for any new features or changes to existing features. (unit tests and integration tests depending on the nature of the change)
- Documentation was updated for any new feature or changes to existing features.

10.6.7 Custom Plugin Support

Need to expand Runway to wrap other tools? Yes - you can do that with custom plugin support.

Overview

Runway can import Python modules that can perform custom deployments with your own set of Runway modules. Let's say for example you want to have Runway execute an Ansible playbook to create an EC2 security group as one of the steps in the middle of your Runway deployment list - this is possible with your own plugin. The custom plugin support allows you to mix-and-match natively supported modules (e.g. CloudFormation, Terraform) with plugins you write providing additional support for non-native modules. Although written in Python, these plugins can natively execute non-Python binaries.

RunwayModule Class

Runway provides *RunwayModule* to use as the base class of all module handler classes. This base class will give you the ability to write your own module handler class that can be added to your runway.yml deployment list (More info on runway.yml below). There are four methods that need to be defined for the class:

deploy This method is called when `runway deploy` is run.

destroy This method is called when `runway destroy` is run.

init This method is called when `runway init` is run.

plan This method is called when `runway plan` is run.

Context Object

`self.ctx` includes many helpful resources for use in your Python module.

Some notable examples are:

- `self.ctx.env.name` - name of the environment
- `self.ctx.env.aws_region` - region in which the module is being executed
- `self.ctx.env.vars` - OS environment variables provided to the module
- `self.path` - path to your Runway module folder

runway.yml Example

After you have written your plugin, you need to add the module `class_path` to your module's configuration. Below is an example `runway.yml` containing a single module that looks for an Ansible playbook in a folder at the root of your Runway environment (i.e. repo) named "security_group.ansible".

Setting `class_path` tells Runway to import the DeployToAWS Python class, from a file named Ansible.py in a folder named "local_runway_extensions" (Standard Python import conventions apply). Runway will execute the `deploy` function in your class when you perform a `runway deploy` (AKA takeoff).

```
deployments:
  - modules:
      - path: security_group.ansible
        class_path: local_runway_extensions.Ansible.DeployToAWS
    regions:
      - us-east-1
```

Below is the `Ansible.py` module referenced above that wraps the `ansible-playbook` command. It will be responsible for deploying an EC2 Security Group from the playbook with a naming convention of `<env>-<region>.yaml` within a fictional `security_group.ansible` Runway module folder. In this example, the `ansible-playbook` binary would already have been installed prior to a Runway deploy, but this example does check to see if it is installed before execution and logs an error if not. The Runway plugin will only execute the ansible-playbook against a `yaml` file associated with the environment and set for the Runway execution and region defined in the `runway.yml`.

Using the above `runway.yml` and the plugin/playbook below saved to the Runway module folder you will only have a deployment occur in the `dev` environment in `us-east-1`. If you decide to perform a runway deployment in the `prod` environment, or in a different region, the ansible-playbook deployment will be skipped. This matches the behavior of the Runway's native modules.

```
"""Ansible Plugin example for Runway."""
from __future__ import annotations

import logging
import subprocess
import sys
from typing import TYPE_CHECKING, Dict

from runway.module.base import RunwayModule
from runway.utils import which

if TYPE_CHECKING:
    from pathlib import Path

LOGGER = logging.getLogger("runway")


def check_for_playbook(playbook_path: Path) -> Dict[str, bool]:
    """Determine if environment/region playbook exists."""
    if playbook_path.is_file():
        LOGGER.info("Processing playbook: %s", playbook_path)
        return {"skipped_configs": False}
    LOGGER.error(
        "No playbook for this environment/region found -- looking for %s",
        playbook_path,
    )
    return {"skipped_configs": True}


class DeployToAWS(RunwayModule):
    """Ansible Runway Module."""

    def deploy(self) -> None:
        """Run ansible-playbook."""
        if not which("ansible-playbook"):
            LOGGER.error(
                '"ansible-playbook" not found in path or is not '
                "executable; please ensure it is installed"
                "correctly."
            )
            sys.exit(1)
        playbook_path = self.path / f"{self.ctx.env.name}-{self.ctx.env.aws_region}"
        response = check_for_playbook(playbook_path)
        if response["skipped_configs"]:
            return
        subprocess.check_output(["ansible-playbook", str(playbook_path)])

    def destroy(self) -> None:
        """Skip destroy."""
        LOGGER.info("destroy not currently supported for Ansible")

    def init(self) -> None:
        """Skip init."""
```

(continues on next page)

(continued from previous page)

```
        LOGGER.info("init not currently supported for Ansible")

    def plan(self) -> None:
        """Skip plan."""
        LOGGER.info("plan not currently supported for Ansible")
```

And below is the example Ansible playbook itself, saved as `dev-us-east-1.yaml` in the security_group.ansible folder:

```
- hosts: localhost
  connection: local
  gather_facts: false
  tasks:
      - name: create a security group in us-east-1
        ec2_group:
          name: dmz
          description: Dev example ec2 group
          region: us-east-1
          rules:
            - proto: tcp
              from_port: 80
              to_port: 80
              cidr_ip: 0.0.0.0/0
        register: security_group
```

The above would be deployed if `runway deploy` was executed in the `dev` environment to `us-east-1`.

10.6.8 Getting Started

Before getting started, fork this repo and clone your fork.

Development Environment

This project includes an optional VSCode Dev Container. This is an Ubuntu 22.04 image that will launch with operating system pre-requisites already installed and VSCode configured for Python debugging. It's not required to use this for development work, but does provide an easy and consistent way to get started.

This project uses poetry to create Python virtual environment. This must be installed on your system before setting up your dev environment.

With poetry installed, run `make setup` to setup your development environment. This will create all the required virtual environments to work on Runway, build docs locally, and run integration tests locally. The virtual environments all have Runway installed as editable meaning as you make changes to the code of your local clone, it will be reflected in all the virtual environments.

pre-commit

pre-commit is configured for this project to help developers follow the coding style. If you used `make setup` to setup your environment, it is already setup for you. If not, you can run `make setup-pre-commit` to to install the pre-commit hooks.

You can also run `make run-pre-commit` at any time to manually trigger these hooks.

pyright Type Checking

This project uses pyright to perform type checking. To run type checking locally, install pyright (`make npm-ci`) then run `make lint` or `make lint-pyright`.

10.6.9 Building Pyinstaller Packages Locally

We use Pyinstaller to build executables that do not require Python to be installed on a system. These are built by Travis CI for distribution to ensure a consistent environment but they can also be build locally for testing.

Prerequisites

These need to be installed globally.

- poetry

Process

1. Export `OS_NAME` environment variable for your system (`ubuntu-22.04`, `macos-12`, or `windows-latest`).
2. Execute `make build-pyinstaller-file` or `make build-pyinstaller-folder` from the root of the repo.

The output of these commands can be found in `./artifacts`

10.6.10 Infrastructure

The Runway repository uses some external infrastructure to run tests and server public content. The code that deploys this infrastructure is located in the `infrastructure/` directory. Each subdirectory is a logical separation between AWS accounts.

Deploying Infrastructure

Infrastructure can be deployed from the root of the `infrastructure/` directory for any environment. We make use of `make` to simplify the process.

To execute Runway for an environment, use the following command syntax.

```
$ make <runway-subcommand> <environment>
```

Example

```
$ make deploy public
```

public

AWS account for public facing assets.

Onica SSO Name

onica-public-prod

Resources

- public S3 bucket that houses build artifacts
  - binary executables
- IAM user used by GitHub Actions & it's policies
  - able to sync with the artifact bucket
  - add entries to a DynamoDB table for the `oni.ca` URL shortener app
    - path to download the binary executables from S3

test

AWS account for running Runway functional tests.

Onica SSO Name

onica-platform-runway-testing-lab

Resources

TBA

test-alt

AWS account for running Runway functional tests that require cross-account access to complete.

Onica SSO Name

onica-platform-runway-testing-alt-lab

Resources

TBA

10.6.11 Release Process

Steps that should be taken when preparing for and executing a release.

Contents

Preparation

1. Merge all PRs that should be included in the release.
2. Ensure that all checks have completed and passed on the *master* branch.

Execution

1. Navigate to the Releases section of the repository on GitHub. There should be a *Draft* already started that was automatically generated from PRs that were merged since the last release.
2. Enter the *Edit* screen of the *Draft*.
3. The *Title* and *Tag* fields should already be filled in with the correct values (e.g. `v<major>.<minor>.<patch>`). Ensure these values match what is expected. The *Tag* should also be targeting the *master* branch.
4. Edit the description of the release as needed but, there should be little to no changes required if all PRs were properly labeled.
5. Mark the release as a *pre-release* if applicable (alpha, beta, release candidate, etc).
6. Publish the release.

At this point, GitHub Actions will begin building the deployment packages & automatically publishing them to npm, PyPi, and AWS S3. The **Publish Release** workflow can be monitored for progress. It can take around 20 minutes for the process to complete. At which time, the logs and package repositories should be checked to verify that the release was published successfully.

10.6.12 Secrets

Information about the secrets used by this project.

GitHub

GitHub secrets are set in the settings of the repository for use by GitHub Actions.

Environment Secrets

Secrets specific to the repository, specific to a single environment.

DEPLOY_AWS_ACCESS_KEY_ID

AWS Access Key for the IAM user used to deploy infrastructure to accounts. This is **not** the user used when running tests but for deploying infrastructure used by tests (including the IAM user running the tests).

DEPLOY_AWS_SECRET_ACCESS_KEY

AWS Secret Access Key for the IAM user used to deploy infrastructure to accounts. This is **not** the user used when running tests but for deploying infrastructure used by tests (including the IAM user running the tests).

Repository Secrets

Secrets specific to the repository, available to all environments.

AWS_ACCESS_KEY

AWS Access Key for the IAM user used to publish artifacts to S3. This IAM user exists in the **public** AWS account.

AWS_SECRET_KEY

AWS Secret Access Key for the IAM user used to publish artifacts to S3. This IAM user exists in the **public** AWS account.

NPM_API_TOKEN

API access token used to publish Runway to NPM.

PYPI_PASSWORD

API token used to publish Runway to PyPi. This should be scoped to only the Runway project.

TEST_PYPI_PASSWORD

Similar to *PYPI_PASSWORD* but for https://test.pypi.org/.

TEST_RUNNER_AWS_ACCESS_KEY_ID

AWS Access Key for the IAM user used to run tests.

TEST_RUNNER_AWS_SECRET_ACCESS_KEY

AWS Secret Access Key for the IAM user used to run tests.

ReadTheDocs

Secrets are set as environment variables for ReadTheDocs to use when building documentation.

`SPHINX_GITHUB_CHANGELOG_TOKEN`

Used by sphinx-github-changelog to generate a changelog for GitHub Releases. The GitHub personal access token scope only needs to include `repo.public_repo`.

10.6.13 CHANGELOG

10.6.14 Apache License

Version 2.0

Date January 2004

URL http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means *(i)* the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or *(ii)* ownership of fifty percent (50%) or more of the outstanding shares, or *(iii)* beneficial ownership of such entity.

"You" (or **"Your"**) shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- You must give any other recipients of the Work or Derivative Works a copy of this License; and
- You must cause any modified files to carry prominent notices stating that You changed the files; and
- You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- If the Work includes a `"NOTICE"` text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such `NOTICE` file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a `NOTICE` text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the `NOTICE` file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the `NOTICE` text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the `NOTICE` file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an **"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND**, either express or implied, including, without limitation, any warranties or conditions of **TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE**. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

10.6.15 Terminology

Runway

Deploy Environment

Deploy environments are used for selecting the options/variables/parameters to be used with each *Module*. The deploy environment is derived from the current directory (if its not a git repo), active git branch, or environment variable (`DEPLOY_ENVIRONMENT`). Standard deploy environments would be something like prod, dev, and test.

When using a git branch, Runway expects the branch to be prefixed with **ENV-**. If this is found, Runway knows that it should always use the value that follows the prefix. If it's the **master** branch, Runway will use the deploy environment name of *common*. If the branch name does not follow either of these schemas and Runway is being run interactively from the CLI, it will prompt of confirmation of the deploy environment that should be used.

When using a directory, Runway expects the directory's name to be prefixed with **ENV-**. If this is found, Runway knows that it should always use the value that follows the prefix.

Deployment

A *deployment* contains a list of *modules* and options for all the *modules* in the *deployment*. A *Runway Config File* can contain multiple *deployments* and a *deployment* can contain multiple *modules*.

Lookup (Runway)

A method for expanding values in the *Runway Config File* file when processing a deployment/module. These are only supported in select areas of the *Runway Config File* (see the config docs for more details).

Module

A *module* is a directory containing a single infrastructure-as-code tool configuration of an application, a component, or some infrastructure (eg. a set of CloudFormation templates). It is defined in a *deployment* by path. Modules can also contain granular options that only pertain to it based on its *module.type*.

Parameters

A mapping of `key: value` that is passed to a module. Through the use of a *Lookup (Runway)*, the value can be changed per region or deploy environment. The `value` can be any data type but, support for complex data types depends on the *module.type*.

Runway's CFngin

Blueprint

A python class that is responsible for creating a CloudFormation template. Usually this is built using troposphere.

context

Context is responsible for translating the values passed in via the command line and specified in the *config* to *stacks*.

graph

A mapping of **object name** to **set/list of dependencies**.

A graph is constructed for each execution of CFNgin from the contents of the *config* file.

Example

```
{
    "stack1": [],
    "stack2": [
        "stack1"
    ]
}
```

- **stack1** depends on nothing.
- **stack2** depends on **stack1**

lookup

A method for expanding values in the *config* at runtime. By default lookups are used to reference Output values from other *stacks* within the same *namespace*.

output

A CloudFormation Template concept. *Stacks* can output values, allowing easy access to those values. Often used to export the unique ID's of resources that templates create.

CFNgin makes it simple to pull outputs from one *stack* and then use them in the *variables* of another *stack*.

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z