drover

Release 0.7.2rc2.dev2+gf238b39

Jeffrey Wilges

Feb 20, 2021

INTRODUCTION

CHAPTER

ONE

BACKGROUND

This utility aims to provide a simple, repeatable, and efficient process for deploying a Python package as a Lambda.

To encourage separating infrequently changing Python dependencies in a distinct "requirements" layer, by default Drover requires a list of regular expressions to define which files to include in the Lambda function; all other files are placed in a requirements layer that is then attached to the Lambda function.

Next, Drover generates and stores hashes for both the Lambda function and the requirements layer. This allows Drover to avoid redundantly updating the Lambda function and/or requirements layer if no package contents have changed.

As much as possible, Drover avoids altering existing infrastructure. Infrastructure utilities such as Terraform may be used to create a Lambda and manage its surrounding resources and Dover may be used to update the Lambda function as well as its layers.

1.1 Examples

1.1.1 A basic Lambda

This example outlines how to deploy a basic Python package with no external dependencies.

Settings

The following `drover.yml` settings file demonstrates how to configure a `staging` stage that may be used to deploy a Python package to a Lambda named `basic-lambda` in the `us-east-1` region:

```
stages:
  staging:
    region_name: us-east-1
    function_name: basic-lambda
    compatible_runtime: python3.8
    function_file_patterns:
    - '^basic_lambda.*'
    function_extra_paths:
    - instance
    upload_bucket:
    region_name: us-east-1
    bucket_name: drover-examples
```

The `compatible_runtime` value will be used to define the compatible runtime for both the requirements layer (if present) and the Lambda function.

While processing files from the install path (see: `--install-path` below), any files matching regular expressions defined in the `function_file_patterns` list will be included in the function; any remaining files will be included in the requirements layer.

The `function_extra_paths` list may contain additional paths to include in the function layer archive; non-absolute paths will be relative to the current working directory.

The `upload_bucket` map may provide a S3 Bucket name and its associated region for use when uploading Lambda function and layer archive files.

Command line interface

Assuming a Python package exists in the `basic_lambda` directory, the following commands demonstrate a simple Lambda deploy with drover:

```
pip install --target install basic_lambda
drover --install-path install staging
```

Assuming the Lambda is not already up to date, drover will attempt to upload the latest source and update the Lambda function:

```
Requirements digest: None
Function digest: 0b37cf78f6ad4c137fb1f77751c0c0e759dd2d6c515937d33fae435b9e091f72
Skipping requirements upload
Uploading function archive...
Failed to upload function archive to bucket; falling back to direct file upload.
Updating function resource...
Updated function "basic-lambda" resource; size: 1.78 KiB; ARN: arn:aws:lambda:us-east-
↪1:977874552542:function:basic-lambda
```

1.2 drover

1.2.1 drover package

Submodules

drover.cli module

Command-line interface functionality for the Drover interface

drover.cli.**main**()
The main command-line entry point for the Drover interface

drover.io module

Generic functionality related to files and I/O

class drover.io.**ArchiveMapping**(*source_file_name: pathlib.Path*, *archive_file_name: pathlib.Path*)

Bases: object

A mapping between an archive file name and its corresponding source filesystem path

archive_file_name: Path = None

source_file_name: Path = None

class drover.io.**FunctionLayerMappings**(*function_mappings: Sequence[drover.io.ArchiveMapping] = <class 'list'>*, *function_digest: str = None*, *requirements_mappings: Sequence[drover.io.ArchiveMapping] = <class 'list'>*, *requirements_digest: str = None*)

Bases: object

A function and requirements layer mapping and digest container

function_digest: str = None

function_mappings

alias of builtins.list

requirements_digest: str = None

requirements_mappings

alias of builtins.list

drover.io.**format_file_size**(*size_in_bytes: float*) → str

Return a string representation of the specified size as its largest 2^10 representation

Examples

```
>>> format_file_size(2048)
'2.00 KiB'
>>> format_file_size(16252928.0)
'15.50 MiB'
```

Parameters **size_in_bytes** – a size in bytes

Returns: a string representation of the specified size as its largest 2^10 representation

drover.io.**get_digest**(*source_file_names: Sequence[pathlib.Path]*, *block_size: int = 8192*) → Optional[str]

Return a SHA256 hash composed from the content of all source files.

Parameters **source_file_names** – A sequence of source file paths

Returns: A SHA256 hash composed from the content of all source files.

drover.io.**get_relative_file_names**(*source_path: pathlib.Path*, *exclude_patterns: Sequence[Pattern] = None*) → Iterable[pathlib.Path]

Return an unsorted iterable of files recursively beneath the source path

Parameters

- **source_path** – a filesystem path from which to recursively iterate all files

- **exclude_patterns** – an optional sequence of regular expressions which will be used to exclude files

Returns: an unsorted iterable of files recursively beneath the source path

drover.io.**write_archive**(*archive_file_name: pathlib.Path*, *archive_mappings: Iterable[drover.io.ArchiveMapping]*) → None

Write a zip file archive composed of the specified archive file mappings

Parameters

- **archive_file_name** – a writable file
- **archive_mappings** – an iterable of mappings of filesystem file names to archive file names

drover.models module

Models for settings and Amazon Web Services interactions

class drover.models.**S3BucketFileVersion**

Bases: pydantic.main.BaseModel

bucket_name: str = None

key: str = None

version_id: Optional[str] = None

class drover.models.**S3BucketPath**

Bases: pydantic.main.BaseModel

bucket_name: str = None

prefix: str = None

region_name: str = None

class drover.models.**Settings**

Bases: pydantic.main.BaseModel

stages: Mapping[str, Stage] = None

class drover.models.**Stage**(***kwargs*)

Bases: pydantic.main.BaseModel

compatible_runtime: str = None

function_extra_paths: Sequence[Path] = None

function_file_patterns: Sequence[Pattern] = None

function_name: str = None

package_exclude_patterns: Sequence[Pattern] = None

region_name: str = None

requirements_layer_name: Optional[str] = None

supplemental_layer_arns: Sequence[str] = None

upload_bucket: Optional[S3BucketPath] = None

Module contents

drover: a command-line utility to deploy Python packages to Lambda functions

class drover.**Drover**(*settings:* drover.models.Settings, *stage: str*, *interactive: bool* = *False*)

Bases: object

An interface to efficiently publish and update a Lambda function and requirements layer representation of a Python package directory

update(*install_path: pathlib.Path*) → None

Publish and/or update a Lambda function and/or requirements layer representation of a Python package directory

Parameters **install_path** – a Python package directory (e.g. via `pip install -t`)

exception drover.**SettingsError**

Bases: RuntimeError

Base settings error

exception drover.**UpdateError**

Bases: RuntimeError

Base update error

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

U

V

W