tropostack

Mar 03, 2020

Contents

CHAPTER 1

Quickstart

1.1 About

Tropostack is a CLI/workflow library that simplifies the creation and management of CloudFormation stacks, based on the excellent Troposphere Project.

Tropostack features:

- Single stack template = single executable Python file = single CLI
- Support for different configuration and CLI plugins
- A collection of generic commands available to each stack (e.g. *create*)
- Support for user-defined CLI commands (e.g. *upscale*)
- Helper routines (e.g. locate the newest matching AMI)

1.2 Docs

Full docs are at https://tropostack.readthedocs.io/en/latest/

1.3 Installation

```
$ pip install tropostack
```

Or, you can use setup.py to install from a cloned repository:

```
$ python setup.py install
```

1.4 First stack

You use *tropostack* as a library to:

- Consisteny define CloudFormation templates in Python code
- Have a CLI around each stack definition, enabling it to live as a standalone executable

Here is a minimalistic example of a stack that creates an S3 bucket, and exports the ARN as an Output:

```
#!/usr/bin/env python3

from troposphere import s3
from troposphere import Output, Export, Sub, GetAtt

from tropostack.base import InlineConfStack
from tropostack.cli import InlineConfCLI


class MyS3BucketStack(InlineConfStack):
    # Name of the stack
    BASE_NAME = 'my-s3-bucket-stack'

    # Define configuration values for the stack
    CONF = {
        # Region is always explicitly required
        'region': 'eu-west-1',
        # Prefix the bucket name with the account ID
        'bucket_name': Sub('${AWS::AccountId}-my-first-tropostack-bucket')
    }

    # Stack Resources are defined as class properties prefixed with 'r_'
    @property
    def r_bucket(self):
        return s3.Bucket(
            'MyBucketResource',
            BucketName=self.conf['bucket_name']
        )

    # Stack Outputs are defined as class properties prefixed with 'o_'
    @property
    def o_bucket_arn(self):
        _id = 'BucketArn'
        return Output(
            _id,
            Description='The ARN of the S3 bucket',
            Value=GetAtt(self.r_bucket, 'Arn'),
            # We're exporting the output as <StackName>-<OutputId>
            # Other stacks can read the output relying on the same convention
            Export=Export(Sub("${AWS::StackName}-%s" % _id))
        )


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    # Wrap the stack in a CLI and run it
    cli = InlineConfCLI(MyS3BucketStack)
    cli.run()
```

The above already gives you a usable CLI around your stack definition.

Assuming you put it inside an executable file called *s3_minimal.py*, you'd be able to call it already:

```
$ ./s3_minimal.py -h
usage: s3_minimal.py [-h]
                     {apply,create,delete,outputs,print,update,validate}

positional arguments:
  {apply,create,delete,outputs,print,update,validate}

optional arguments:
  -h, --help            show this help message and exit
```

You can now inspect the "raw" CloudFormation code generated by the stack:

```
$ ./s3_minimal.py print
Outputs:
  BucketArn:
    Description: The ARN of the S3 bucket
    Export:
      Name: !Sub '${AWS::StackName}-BucketArn'
    Value: !GetAtt 'MyBucketResource.Arn'
Resources:
  MyBucketResource:
    Properties:
      BucketName: !Sub '${AWS::AccountId}-my-first-tropostack-bucket'
    Type: AWS::S3::Bucket
```

Assuming AWS credentials are present in the environment, we can now fire up stack that would create our S3 bucket:

```
$ ./s3_minimal.py create
Stack creation initiated for: arn:aws:cloudformation:eu-west-1:472799024263:stack/my-
↪s3-bucket-stack/dd5e93c0-225c-11ea-93d8-0641c159a77a
TIMESTAMP (UTC)          RESOURCE TYPE                                RESOURCE ID
↪           STATUS                                  REASON
2019-12-19 12:41:23      AWS::CloudFormation::Stack                   my-s3-bucket-
↪stack           CREATE_IN_PROGRESS                      User Initiated
2019-12-19 12:41:26      AWS::S3::Bucket                              MyBucketResource
↪           CREATE_IN_PROGRESS
2019-12-19 12:41:27      AWS::S3::Bucket                              MyBucketResource
↪           CREATE_IN_PROGRESS                      Resource creation Initiated
2019-12-19 12:41:48      AWS::S3::Bucket                              MyBucketResource
↪           CREATE_COMPLETE
2019-12-19 12:41:50      AWS::CloudFormation::Stack                   my-s3-bucket-
↪stack           CREATE_COMPLETE
```

We can also inspect the stack Outputs - in this case, the ARN of the bucket:

```
$ ./s3_minimal.py outputs
Stack is in status: CREATE_COMPLETE
OutputKey    OutputValue                                              Description
↪       ExportName
```

(continues on next page)

(continued from previous page)

```
-----------  --------------------------------------------------  -------------------
↪-----  ----------------------------
BucketArn    arn:aws:s3:::472799024263-my-first-tropostack-bucket  The ARN of the S3␣
↪bucket  my-s3-bucket-stack-BucketArn
```

Finally, we can clean up and have our stack deleted:

```
$ ./s3_minimal.py delete
Destroy initiated for stack: my-s3-bucket-stack
TIMESTAMP (UTC)          RESOURCE TYPE                                RESOURCE ID
↪           STATUS                                  REASON                                ␣
2019-12-19 12:44:59      AWS::CloudFormation::Stack                   my-s3-bucket-
↪stack           DELETE_IN_PROGRESS                        User Initiated
Stack is gone: my-s3-bucket-stack (An error occurred (ValidationError) when calling␣
↪the DescribeStackEvents operation: Stack [my-s3-bucket-stack] does not exist)
```

1.5 Stock commands

While the CLI can be expanded/customized for each individual tropostack, there are several subcommands that come out of the box:

- *print* - prints the resulting CloudFormation YAML to the screen
- *validate* - Sends the CloudFormation template to the AWS API for validation, and reports back result
- *create* - Initiates the stack creation (should only be used if the stack does not exist yet)
- *update* - Updates an existing stack (should only be used if the stack exists)
- *apply* - Idempotently updates or creates a stack, based on whether it exists or not
- *outputs* - Shows the outputs of an existing stack
- *delete* - Deletes an existing stack

CHAPTER 2

Examples

2.1 S3

2.1.1 s3_minimal

class examples.s3_bucket.s3_minimal.**MyS3BucketStack**(*conf*)

Minimal S3 bucket creation class. Single stack per region - no environment/release variation.

Parameters **bucket_name** (*str*) – The name of the S3 bucket to be created. Can contain AWS variables such as `${AWS::AccountId}`

Outputs: BucketArn (str): The ARN of the created S3 bucket

2.1.2 s3_policy

class examples.s3_bucket.s3_policy.**AugmentedCLI**(*stack_cls*)

Extend the default set of CLI commands to add a custom action.

cmd_purge()

Delete all objects inside the S3 bucket, along with the bucket itself.

class examples.s3_bucket.s3_policy.**S3BucketStack**(*conf*)

Tropostack defining an S3 bucket with optional IP-based access restriction

Parameters

- **allowed_cidr** (*str*) – IP CIDR range to allow access from. Use `0.0.0.0/0` to allow access from anywhere.
- **bucket_name** (*str*) – The name of the S3 bucket to be created. Can contain AWS variables such as `${AWS::AccountId}`

Outputs: BucketArn (str): The ARN of the created S3 bucket

2.1.3 s3_user

class examples.s3_bucket.s3_user.**S3UserStack**(*conf*)
Tropostack defining an S3 bucket, together with an IAM user account that is allowed to access the bucket

Parameters

- **region** (*str*) – Explicit region specification for the stack
- **bucket_name** (*str*) – The name of the S3 bucket to be created. Can contain AWS variables such as `${AWS::AccountId}`
- **path** (*str*) – Templated IAM user path. Must start and finish with a `/`
- **username** (*str*) – Templated username, e.g. `${AWS::StackName}-bot`
- **allowed_actions** (*list of str*) – S3 API actions to be enabled for the user

Outputs: BucketArn (str): The ARN of the created S3 bucket UserName (str): The ARN of the created S3 bucket

2.2 EC2

2.2.1 ec2_static_ip

class examples.ec2.ec2_static_ip.**EC2Stack**(*conf*)
Single-instance EC2 stack, which assigns a static IP address to the instance. Also features a security group, dedicated to the instance/stack. Uses a human-friendly AMI path specification rather than AMI ID.

Parameters

- **region** (*str*) – Region where the stack/instance would be deployed
- **instance_type** (*str*) – EC2 instance type
- **ami_location** (*str*) – Qualified path to the AMI (i.e. Source in the UI). Example: `amazon/amzn2-ami-hvm-2.0.20191116.0-x86_64-ebs`
- **vpc_id** (*str*) – VPC that the instance would be a part of
- **subnet_id** (*str*) – ID of the subnet where the instance would be deployed
- **ssh_key_name** (*str*) – SSH Keypair name to be associated with the instance
- **private_ip** (*str*) – Static IP address of the instance. Must be available under the respective Subnet
- **access** (*list of 3-tuples*) – List of 3 tuples to allow Ingress from, formatted as (Protocol, Port, Network Range). Sample value: `[('tcp', 22, '0.0.0.0/0'), ]`

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index