Moto Documentation

Release 0.4.10

Steve Pulec

Oct 08, 2017

Contents

A library that allows you to easily mock out tests based on AWS infrastructure.

CHAPTER 1

Getting Started

If you've never used `moto` before, you should read the *Getting Started with Moto* guide to get familiar with `moto` and its usage.

CHAPTER 2

Currently implemented Services:

Service Name	Decorator	Development Status
API Gateway	@mock_apigateway	core endpoints done
Autoscaling	@mock_autoscaling	core endpoints done
Cloudformation	@mock_cloudformation	core endpoints done
Cloudwatch	@mock_cloudwatch	basic endpoints done
Data Pipeline	@mock_datapipeline	basic endpoints done
• DynamoDB • DynamoDB2	• @mock_dynamodb • @mock_dynamodb2	• core endpoints done • core endpoints + partial indexes
EC2 • AMI • EBS • Instances • Security Groups • Tags	@mock_ec2	core endpoints done core endpoints done core endpoints done all endpoints done core endpoints done all endpoints done
ECS	@mock_ecs	basic endpoints done
ELB	@mock_elb @mock_elbv2	core endpoints done core endpoints done
EMR	@mock_emr	core endpoints done
Glacier	@mock_glacier	core endpoints done
IAM	@mock_iam	core endpoints done
Lambda	@mock_lambda	basic endpoints done
Kinesis	@mock_kinesis	core endpoints done
KMS	@mock_kms	basic endpoints done
RDS	@mock_rds	core endpoints done
RDS2	@mock_rds2	core endpoints done
Redshift	@mock_redshift	core endpoints done
Route53	@mock_route53	core endpoints done
S3	@mock_s3	core endpoints done
SES	@mock_ses	core endpoints done
SNS	@mock_sns	core endpoints done
SQS	@mock_sqs	core endpoints done
STS	@mock_sts	core endpoints done
SWF	@mock_swf	basic endpoints done

CHAPTER 3

Additional Resources

- Moto Source Repository
- Moto Issue Tracker

Getting Started with Moto

Installing Moto

You can use `pip` to install the latest released version of `moto`:

```
pip install moto
```

If you want to install `moto` from source:

```
git clone git://github.com/spulec/moto.git
cd moto
python setup.py install
```

Moto usage

For example we have the following code we want to test:

```
import boto
from boto.s3.key import Key

class MyModel(object):
    def __init__(self, name, value):
        self.name = name
        self.value = value

    def save(self):
```

```
        conn = boto.connect_s3()
        bucket = conn.get_bucket('mybucket')
        k = Key(bucket)
        k.key = self.name
        k.set_contents_from_string(self.value)
```

There are several method to do this, just keep in mind Moto creates a full blank environment.

Decorator

With a decorator wrapping all the calls to S3 are automatically mocked out.

```
import boto
from moto import mock_s3
from mymodule import MyModel

@mock_s3
def test_my_model_save():
    conn = boto.connect_s3()
    # We need to create the bucket since this is all in Moto's 'virtual' AWS account
    conn.create_bucket('mybucket')

    model_instance = MyModel('steve', 'is awesome')
    model_instance.save()

    assert conn.get_bucket('mybucket').get_key('steve').get_contents_as_string() ==
↪'is awesome'
```

Context manager

Same as decorator, every call inside `with` statement are mocked out.

```
def test_my_model_save():
    with mock_s3():
        conn = boto.connect_s3()
        conn.create_bucket('mybucket')

        model_instance = MyModel('steve', 'is awesome')
        model_instance.save()

        assert conn.get_bucket('mybucket').get_key('steve').get_contents_as_string()␣
↪== 'is awesome'
```

Raw

You can also start and stop manually the mocking.

```
def test_my_model_save():
    mock = mock_s3()
    mock.start()

    conn = boto.connect_s3()
    conn.create_bucket('mybucket')
```

```
    model_instance = MyModel('steve', 'is awesome')
    model_instance.save()

    assert conn.get_bucket('mybucket').get_key('steve').get_contents_as_string() ==
↪'is awesome'

    mock.stop()
```

Stand-alone server mode

Moto comes with a stand-alone server allowing you to mock out an AWS HTTP endpoint. It is very useful to test even if you don't use Python.

```
$ moto_server ec2 -p3000
 * Running on http://127.0.0.1:3000/
```

This method isn't encouraged if you're using `boto`, best is to use decorator method.

Server mode

Moto has a stand-alone server mode. This allows you to utilize the backend structure of Moto even if you don't use Python.

It uses flask, which isn't a default dependency. You can install the server 'extra' package with:

```
pip install moto[server]
```

You can then start it running a service:

```
$ moto_server ec2
```

You can also pass the port:

```
$ moto_server ec2 -p3000
 * Running on http://127.0.0.1:3000/
```

If you want to be able to use the server externally you can pass an IP address to bind to as a hostname or allow any of your external interfaces with 0.0.0.0:

```
$ moto_server ec2 -H 0.0.0.0
 * Running on http://0.0.0.0:5000/
```

Please be aware this might allow other network users to access your server.

Then go to localhost to see a list of running instances (it will be empty since you haven't added any yet).

If you want to use boto3 with this, you can pass an *endpoint_url* to the resource

```
boto3.resource(
    service_name='s3',
    region_name='us-west-1',
    endpoint_url='http://localhost:5000',
)
```

Other languages

You don't need to use Python to use Moto; it can be used with any language. Here are some examples to run it with other languages:

- Java
- Ruby
- Javascript

Moto APIs

Moto provides some internal APIs to view and change the state of the backends.

Reset API

This API resets the state of all of the backends. Send an HTTP POST to reset:

```
requests.post("http://motoapi.amazonaws.com/moto-api/reset")
```

Dashboard

Moto comes with a dashboard to view the current state of the system:

```
http://localhost:5000/moto-api/
```

Use Moto as EC2 backend

This tutorial explains `moto.ec2`'s features and how to use it. This tutorial assumes that you have already downloaded and installed boto and moto. Before all code examples the following snippet is launched:

```
>>> import boto.ec2, moto
>>> mock_ec2 = moto.mock_ec2()
>>> mock_ec2.start()
>>> conn = boto.ec2.connect_to_region("eu-west-1")
```

Launching instances

After mock is started, the behavior is the same than previously:

```
>>> reservation = conn.run_instances('ami-f00ba4')
>>> reservation.instances[0]
Instance:i-91dd2f32
```

Moto set static or generate random object's attributes:

```
>>> vars(reservation.instances[0])
{'_in_monitoring_element': False,
 '_placement': None,
 '_previous_state': None,
 '_state': pending(0),
 'ami_launch_index': u'0',
 'architecture': u'x86_64',
 'block_device_mapping': None,
 'client_token': '',
 'connection': EC2Connection:ec2.eu-west-1.amazonaws.com,
 'dns_name': u'ec2-54.214.135.84.compute-1.amazonaws.com',
 'ebs_optimized': False,
 'eventsSet': None,
 'group_name': None,
 'groups': [],
 'hypervisor': u'xen',
 'id': u'i-91dd2f32',
 'image_id': u'f00ba4',
 'instance_profile': None,
 'instance_type': u'm1.small',
 'interfaces': [NetworkInterface:eni-ed65f870],
 'ip_address': u'54.214.135.84',
 'item': u'\n        ',
 'kernel': u'None',
 'key_name': u'None',
 'launch_time': u'2015-07-27T05:59:57Z',
 'monitored': True,
 'monitoring': u'\n          ',
 'monitoring_state': u'enabled',
 'persistent': False,
 'platform': None,
 'private_dns_name': u'ip-10.136.187.180.ec2.internal',
 'private_ip_address': u'10.136.187.180',
 'product_codes': [],
 'public_dns_name': u'ec2-54.214.135.84.compute-1.amazonaws.com',
 'ramdisk': None,
 'reason': '',
 'region': RegionInfo:eu-west-1,
 'requester_id': None,
 'root_device_name': None,
 'root_device_type': None,
 'sourceDestCheck': u'true',
 'spot_instance_request_id': None,
 'state_reason': None,
 'subnet_id': None,
 'tags': {},
 'virtualization_type': u'paravirtual',
 'vpc_id': None}
```