

---

# **Moto Documentation**

*Release 0.4.10*

**Steve Pulec**

**Sep 12, 2019**



---

# Contents

---

<b>1</b>	<b>Getting Started</b>	<b>3</b>
<b>2</b>	<b>Currently implemented Services:</b>	<b>5</b>
<b>3</b>	<b>Additional Resources</b>	<b>7</b>
3.1	Getting Started with Moto . . . . .	7
3.2	Server mode . . . . .	9
3.3	Moto APIs . . . . .	10
3.4	Use Moto as EC2 backend . . . . .	10



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.





---

## Currently implemented Services:

---

Service Name	Decorator	Development Status
ACM	@mock_acm	all endpoints done
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
CloudwatchEvents	@mock_events	all endpoints done
Cognito Identity	@mock_cognitoidentity	all endpoints done
Cognito Identity Provider	@mock_cognitoidp	all endpoints done
Config	@mock_config	basic endpoints done
Data Pipeline	@mock_datapipeline	basic endpoints done
DynamoDB DynamoDB2	<ul style="list-style-type: none"> <li>• @mock_dynamodb</li> <li>• @mock_dynamodb2</li> </ul>	<ul style="list-style-type: none"> <li>• core endpoints done</li> <li>• core endpoints + partial indexes</li> </ul>
<b>EC2</b> <ul style="list-style-type: none"> <li>• AMI</li> <li>• EBS</li> <li>• Instances</li> <li>• Security Groups</li> <li>• Tags</li> </ul>	@mock_ec2	<b>core endpoints done</b> <ul style="list-style-type: none"> <li>• core endpoints done</li> <li>• core endpoints done</li> <li>• all endpoints done</li> <li>• core endpoints done</li> <li>• all endpoints done</li> </ul>
ECR	@mock_ecr	basic endpoints done
ECS	@mock_ecs	basic endpoints done
ELB	@mock_elb	core endpoints done
ELBv2	@mock_elbv2	all endpoints done
EMR	@mock_emr	core endpoints done
Glacier	@mock_glacier	core endpoints done

Continued on next page

Table 1 – continued from previous page

Service Name	Decorator	Development Status
IAM	@mock_iam	core endpoints done
IoT	@mock_iot @mock_iotdata	core endpoints done core endpoints done
Kinesis	@mock_kinesis	core endpoints done
KMS	@mock_kms	basic endpoints done
Lambda	@mock_lambda	basic endpoints done, requires docker
Logs	@mock_logs	basic endpoints done
Organizations	@mock_organizations	some core endpoints done
Polly	@mock_polly	all 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
SecretsManager	@mock_secretsmanager	basic endpoints done
SES	@mock_ses	all endpoints done
SNS	@mock_sns	all endpoints done
SQS	@mock_sqs	core endpoints done
SSM	@mock_ssm	core endpoints done
STS	@mock_sts	core endpoints done
SWF	@mock_swf	basic endpoints done
X-Ray	@mock_xray	all endpoints done

---

## Additional Resources

---

- [Moto Source Repository](#)
- [Moto Issue Tracker](#)

### 3.1 Getting Started with Moto

#### 3.1.1 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
```

#### 3.1.2 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
```

(continues on next page)

(continued from previous page)

```
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 ways to do this, but you should keep in mind that 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 the Decorator, every call inside the `with` statement is 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 the mocking manually.

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

(continues on next page)

(continued from previous page)

```

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 also comes with a stand-alone server allowing you to mock out an AWS HTTP endpoint. For testing purposes, it's extremely useful even if you don't use Python.

```

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

```

However, this method isn't encouraged if you're using `boto`, the best solution would be to use a decorator method.

## 3.2 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',
)
```

### 3.2.1 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

## 3.3 Moto APIs

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

### 3.3.1 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")
```

### 3.3.2 Dashboard

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

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

## 3.4 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")
```

### 3.4.1 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}
```