# Contents

1 Getting Started  
2 Currently implemented Services:  
3 Additional Resources  
   3.1 Getting Started with Moto  
   3.2 Server mode  
   3.3 Moto APIs  
   3.4 Use Moto as EC2 backend  

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started</td>
<td>3</td>
</tr>
<tr>
<td>Currently implemented Services:</td>
<td>5</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Getting Started with Moto</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Server mode</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Moto APIs</td>
<td>10</td>
</tr>
<tr>
<td>3.4 Use Moto as EC2 backend</td>
<td>11</td>
</tr>
</tbody>
</table>
A library that allows you to easily mock out tests based on AWS infrastructure.
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:

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Decorator</th>
<th>Development Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACM</td>
<td>@mock_acm</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>API Gateway</td>
<td>@mock_apigateway</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>Autoscaling</td>
<td>@mock_autoscaling</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>Cloudformation</td>
<td>@mock_cloudformation</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>Cloudwatch</td>
<td>@mock_cloudwatch</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>CloudwatchEvents</td>
<td>@mock_events</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>Cognito Identity</td>
<td>@mock_cognitoidentity</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>Cognito Identity Provider</td>
<td>@mock_cognitoidp</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>Config</td>
<td>@mock_config</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>Data Pipeline</td>
<td>@mock_datapipeline</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>DMS</td>
<td>@mock_dms</td>
<td>basic endpoints done</td>
</tr>
</tbody>
</table>
| DynamoDB DynamoDB2            | @mock_dynamodb @mock_dynamodb2 | core endpoints done
                                    |                           | core endpoints + partial indexes |
| EC2                           | @mock_ec2                |                                      |
|                               |                          | core endpoints done                 |
|                               |                          | core endpoints done                 |
|                               |                          | all endpoints done                  |
|                               |                          | core endpoints done                 |
|                               |                          | all endpoints done                  |
| 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                 |

Continued on next page
<table>
<thead>
<tr>
<th>Service Name</th>
<th>Decorator</th>
<th>Development Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forecast</td>
<td>@mock_forecast</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>Glacier</td>
<td>@mock_glacier</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>IAM</td>
<td>@mock_iam</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>IoT</td>
<td>@mock_iot @mock_iotdata</td>
<td>core endpoints done core endpoints done</td>
</tr>
<tr>
<td>Kinesis</td>
<td>@mock_kinesis</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>KMS</td>
<td>@mock_kms</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>Lambda</td>
<td>@mock_lambda</td>
<td>basic endpoints done, requires docker</td>
</tr>
<tr>
<td>Logs</td>
<td>@mock_logs</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>Organizations</td>
<td>@mock_organizations</td>
<td>some core endpoints done</td>
</tr>
<tr>
<td>Polly</td>
<td>@mock_polly</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>RDS</td>
<td>@mock_rds</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>RDS2</td>
<td>@mock_rds2</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>Redshift</td>
<td>@mock_redshift</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>Route53</td>
<td>@mock_route53</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>S3</td>
<td>@mock_s3</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>SecretsManager</td>
<td>@mock_secretsmanager</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>SES</td>
<td>@mock_ses</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>SFN</td>
<td>@mock_stepfunctions</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>SNS</td>
<td>@mock_sns</td>
<td>all endpoints done</td>
</tr>
<tr>
<td>SQS</td>
<td>@mock_sqs</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>SSM</td>
<td>@mock_ssm</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>STS</td>
<td>@mock_sts</td>
<td>core endpoints done</td>
</tr>
<tr>
<td>SWF</td>
<td>@mock_swf</td>
<td>basic endpoints done</td>
</tr>
<tr>
<td>X-Ray</td>
<td>@mock_xray</td>
<td>all endpoints done</td>
</tr>
</tbody>
</table>


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[ec2,s3,..]
```

This will install Moto, and the dependencies required for that specific service.

If you don’t care about the number of dependencies, or if you want to mock many AWS services:

```
pip install moto[all]
```

Not all services might be covered, in which case you might see a warning: `moto 1.3.16 does not provide the extra 'service'`.

You can ignore the warning, or simply install moto as is:

```
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:

```python
import boto3

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

    def save(self):
        s3 = boto3.client('s3', region_name='us-east-1')
        s3.put_object(Bucket='mybucket', Key=self.name, Body=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.

```python
import boto3
from moto import mock_s3
from mymodule import MyModel

@mock_s3
def test_my_model_save():
    conn = boto3.resource('s3', region_name='us-east-1')
    # We need to create the bucket since this is all in Moto's 'virtual' AWS account
    conn.create_bucket(Bucket='mybucket')

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

    body = conn.Object('mybucket', 'steve').get()['Body'].read().decode("utf-8")

    assert body == 'is awesome'
```

**Context manager**

Same as the Decorator, every call inside the with statement is mocked out.

```python
def test_my_model_save():
    with mock_s3():
        conn = boto3.resource('s3', region_name='us-east-1')
        conn.create_bucket(Bucket='mybucket')

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

        body = conn.Object('mybucket', 'steve').get()['Body'].read().decode("utf-8")

        assert body == 'is awesome'
```
Raw

You can also start and stop the mocking manually.

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

    conn = boto3.resource('s3', region_name='us-east-1')
    conn.create_bucket(Bucket='mybucket')

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

    body = conn.Object('mybucket', 'steve').get()['Body'].read().decode('utf-8')

    assert body == '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: 

```python
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

### 3.2.2 Install with Homebrew

Moto is also available to install using Homebrew, which makes it much easier to manage if you’re not using Python as your primary development language.

Once Homebrew is installed, you can install Moto by running:

```bash
brew install moto
```

To make the Moto server start up automatically when you log into your computer, you can run:

```bash
brew services start moto
```

### 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:

```bash
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:

```bash
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:

```python
>>> 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:

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

Moto set static or generate random object’s attributes:

```python
>>> 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'
', 
'_kernel': u'None', 
'_key_name': u'None', 
'_launch_time': u'2015-07-27T05:59:57Z', 
'_monitored': True, 
'_monitoring': u'
', 
'_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',
```

(continues on next page)