**Final Project for Python class:**

**Use BOTO3 SDK**

**DRAFT (Due 6/15)**

**Mitsuyo Yasuhara**

**1. Project Overview**

**Get familiarize AWS Python SDK, “BOTO3” to instantiate resources.**

**2. Environment**

* Windows 10
* Canopy
* Python 2.7.14
* AWS Python SDK, “BOTO3”

**3. What is BOTO3?**

Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python, which allows Python developers to write software that makes use of services like Amazon S3 and Amazon EC2.

**4. Quick Start**

**(1) How to import boto3**

Install the library and set a default region:

$ pip install boto3

Install AWS CLI

Linux: pip install awscli

Windows: <https://s3.amazonaws.com/aws-cli/AWSCLI64.msi>

**(2) Configure AWS environment**

AWSCLI is a set of command-line tools for accessing AWS. Really, the main reason why we use it is for easy configuration, as it comes with a subcommand named configure.

After **awscli** has been installed, you should be able to run the configuration command from your command-line:

$ aws configure

This will take you to an interactive prompt which will ask you for 4 things:

* Your AWS Access Key
* Your AWS Secret Access Key
* A “Default region name” – enter us-east-1
* A “Default output format” – enter json

$ aws configure

AWS Access Key ID **[**None**]**: AKIAIOSFODNN7EXAMPLE

AWS Secret Access Key **[**None**]**: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

Default region name **[**None**]**: us-east-1

Default output format **[**None**]**: json

#### (3) [What aws configure does](http://2017.compciv.org/guide/topics/aws/intro-to-aws-boto3.html#id6)

Just to make it obvious that there’s no magic here, what the configure command does is set up a text file that the awscli and boto3`` libraries are configured to look at, by default, for your credentials. If you are on OS X/Linux, the configure command creates a new text file at this path:

~/.aws/credentials

(If you’re on Windows, the location is the same place, except with Windows-style paths)

You can view it using the cat program (or open):

$ cat ~/.aws/credentials

And the file should look like this:

[default]

aws\_access\_key\_id = AKIAIOSFODNN7EXAMPLE

aws\_secret\_access\_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

All that the aws configure tool does is make it so that we don’t have to store these credentials (i.e. our password) to AWS in our actual scripts. In other words, you should never, ever have to do this in a Python script:

MY\_AWS\_ACCESS\_KEY **=** 'AKIAIOSFODNN7EXAMPLE'

MY\_AWS\_SECRET\_KEY **=** 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'

Some examples of bad developers who put their credentials into actual code files and then saved them online:

**(4) How to import BOTO3 in your environment**

From a Python interpreter:

>>> import boto3

>>> s3 = boto3.resource('s3')

>>> for bucket in s3.buckets.all():

print(bucket.name)

Canopy is not able to identify the path of BOTO3 python. However, Python IDLE does not only need to specify “import BOTO3”.

To import BOTO3 on Windows 10, use the following to statements.

import sys

sys.path.extend(['C:\\Python27\\Lib\\idlelib',

'C:\\Windows\\SYSTEM32\\python27.zip', 'C:\\Python27\\DLLs',

'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win',

'C:\\Python27\\lib\\lib-tk', 'C:\\Python27',

'C:\\Python27\\lib\\site-packages'])

import boto3

Or

import sys  
sys.path.append("C:\\Python27\\Lib\\idlelib")  
sys.path.append("C:\\Windows\\SYSTEM32\\python27.zip")  
sys.path.append("C:\\Python27\\DLLs")  
sys.path.append("C:\\Python27\\lib")  
sys.path.append("C:\\Python27\\lib\\plat-win")  
sys.path.append("C:\\Python27\\lib\\lib-tk")  
sys.path.append("C:\\Python27")  
sys.path.append("C:\\Python27\\lib\\site-packages")  
import boto3

When the Python interpreter executes an [**import**](https://docs.python.org/2/reference/simple_stmts.html#import) statement, it searches for both Python code and extension modules along a search path. A default value for the path is configured into the Python binary when the interpreter is built. You can determine the path by importing the [**sys**](https://docs.python.org/2/library/sys.html#module-sys)module and printing the value of sys.path.

C:\Users\yasuh>**python**

Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

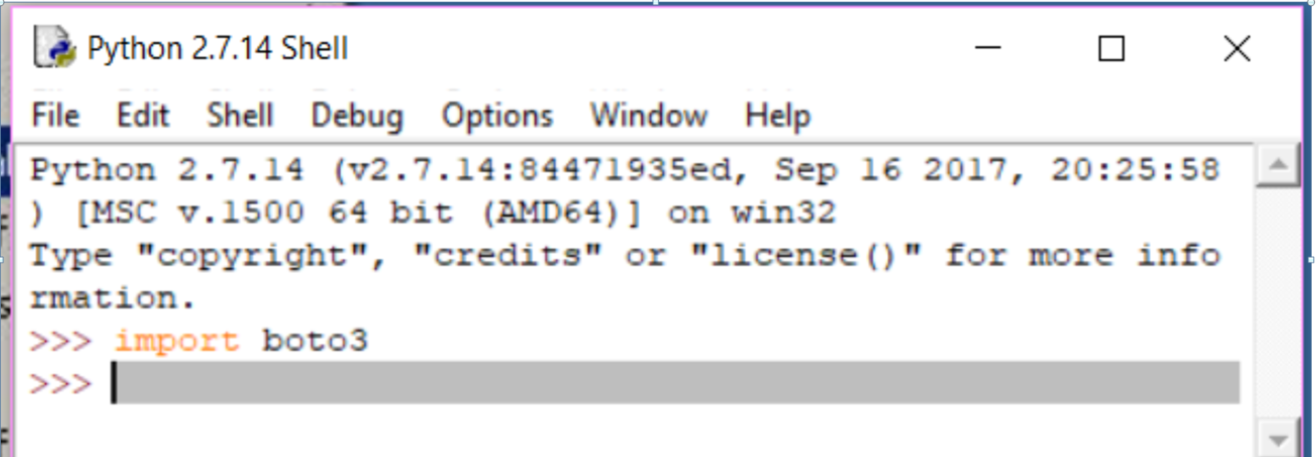
>>> **import sys**

>>> **sys.path**

['', 'C:\\WINDOWS\\SYSTEM32\\python27.zip', 'C:\\Python27\\DLLs', 'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win', 'C:\\Python27\\lib\\lib-tk', 'C:\\Python27', 'C:\\Users\\yasuh\\AppData\\Roaming\\Python\\Python27\\site-packages', 'C:\\Python27\\lib\\site-packages']

>>>

The null string in sys.path represents the current working directory.



**(5) Usage**

Add the following statement if you want to run it under Linux

#!/usr/bin/env python

**5. Reference Links**

* Studying Python

<https://sites.google.com/site/studyingpython/misc/boto>

* Automating AWS With Python and Boto3

<https://linuxacademy.com/howtoguides/posts/show/topic/14209-automating-aws-with-python-and-boto3>

* Introduction to AWS with Python and boto3  
  <http://2017.compciv.org/guide/topics/aws/intro-to-aws-boto3.html>
* Amazon AWS sites

The original Boto (AWS SDK for Python Version 2) can still be installed using pip

(pip install boto):

<https://github.com/boto/boto>

* [boto v2.48.0](https://boto.readthedocs.io/en/latest/)

<https://boto.readthedocs.io/en/latest/>

Quick Start:

<https://github.com/boto/boto3>

Python Developer Center - Discover more about using Python

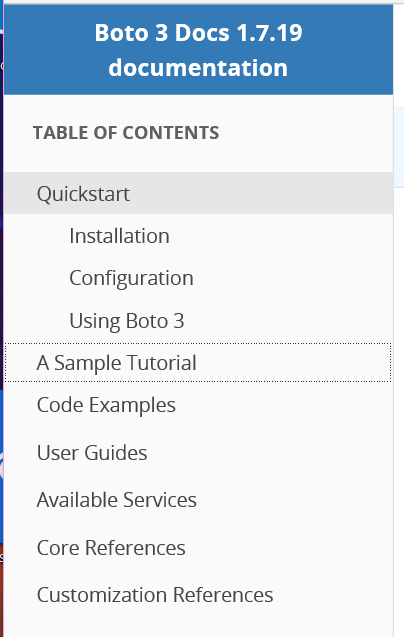
<https://aws.amazon.com/python/>

Dig through the source code

<https://aws.amazon.com/sdk-for-python/>

BOTO 3 DOCS 1.7.19

http://boto3.readthedocs.io/en/latest/



**6. Quick Run**

**(1) EC2**

* **list\_instances.py – list all EC2 instances**

ec2 = boto3.resource('ec2')

for instance in ec2.instances.all():

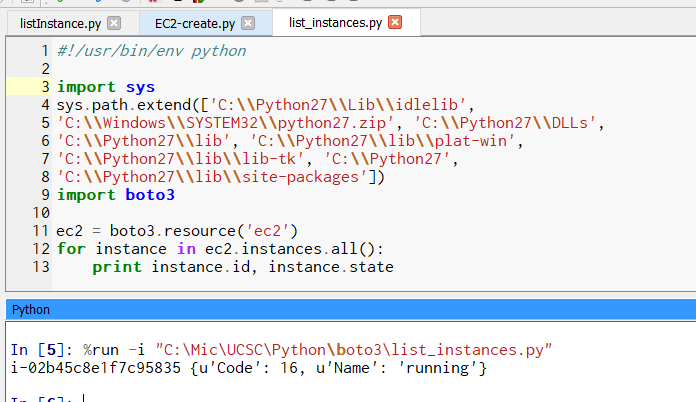
print instance.id, instance.state

Run on DOS command line

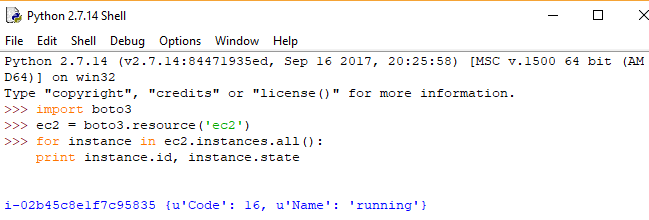
PS C:\mic\ucsc\python\boto3> **python list\_instances.py**

i-02b45c8e1f7c95835 {u'Code': 16, u'Name': 'running'}

Run on Canopy



Run each command on IDLE



* **Terminate EC2 - terminateEC2.py**

ec2 = boto3.resource('ec2')  
for instance\_id in sys.argv[1:]:  
 instance = ec2.Instance(instance\_id)  
 response = instance.terminate()  
 print response

PS C:\mic\ucsc\python\boto3> **python terminateEC2.py i-02b45c8e1f7c95835**

{u'TerminatingInstances': [{u'InstanceId': 'i-02b45c8e1f7c95835', u'CurrentState': {u'Code': 32, u'Name': 'shutting-down'}, u'PreviousState': {u'Code': 16, u'Name': 'running'}}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': 'fb4baf29-29a3-4b27-bae6-acf6497984b5', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Fri, 25 May 2018 18:44:21 GMT'}}}

Python "C:\Mic\UCSC\Python\boto3\**list\_instances.py**

i-02b45c8e1f7c95835 {u'Code': 32, u'Name': **'shutting-down'**}

* Create EC2 – createEC2.py

ec2 = boto3.resource('ec2')

instance = ec2.create\_instances(

ImageId='ami-25110f45',

MinCount=1,

MaxCount=1,

InstanceType='t2.micro')

print instance[0].id

%run -i "C:\Mic\UCSC\Python\boto3\**createEC2.py**"

i-0fb8c87c56e96f2f5

%run -i "C:\Mic\UCSC\Python\boto3\**list\_instances.py**"

i-02b45c8e1f7c95835 {u'Code': 48, u'Name': 'terminated'}

i-0fb8c87c56e96f2f5 {u'Code': 16, u'Name': 'running'}

**(2) S3**

* **List Bucket and its content– listS3Buvket.py**

s3 = boto3.resource('s3')

for bucket in s3.**buckets.all**():

print bucket.name

print "---"

for item in **bucket.objects.all**():

print "\t%s" % item.key#!/usr/bin/env python

%run -i "C:\Mic\UCSC\Python\boto3\listS3Bucket.py"

cf-templates-1b1hpaivisk3x-us-west-1

---

2018128BDk-EC2InstanceWithSecurityGroup.json

2018128Bi6-assignment9.json

2018128Lm6-creatteRDS.json

2018128XGr-assignment9.json

2018128ctY-EC2InstanceWithSecurityGroupSample.json

2018128dc1-assignment9.json

2018128eKh-EC2InstanceWithSecurityGroup.json

2018128eeM-EC2InstanceWithSecurityGroupSample.json

2018128gZG-EC2InstanceWithSecurityGroupSample.json

2018128oR4-assignment9.json

2018128tM9-EC2InstanceWithSecurityGroup.json

2018128tRf-EC2InstanceWithSecurityGroup-2.json

2018128y0n-EC2InstanceWithSecurityGroup-2.json

2018128y5I-assignment9.json

cf-templates-1b1hpaivisk3x-us-west-2

---

2018128RSU-EC2InstanceWithSecurityGroupSample.json

2018128zTx-sharedinfrastructure.yaml

edureka-s3appbucket-10tjfieolas3d

---

elasticbeanstalk-us-east-2-984717818485

---

myasuhara

---

* **Create S3 bucket - createS3Bucket.py**

s3 = boto3.resource("s3")

for bucket\_name in sys.argv[1:]:

try:

response = s3.create\_bucket(

Bucket=bucket\_name,

CreateBucketConfiguration={'LocationConstraint': 'us-west-1'}

)

**# Regions outside of us-east-1 require the appropriate LocationConstraint to be**

**# specified in order to create the bucket in the desired region**

**print response**

**except Exception as error:**

**print error**

PS C:\mic\ucsc\python\boto3> **python createS3Bucket.py mic-20180525**

**s3.Bucket(name='mic-20180525')**

PS C:\mic\ucsc\python\boto3> **python createS3Bucket.py projectx-bucket1-$(date +%F-%s)**

s3.Bucket(name='projectx-bucket1-2018-05-25-1527284257')

Note: Because simple bucket names like “my\_bucket” are usually not available, a good way to get a unique bucket name is to use a name, a number, and the date. For example:

* **Put a file into a S3 bucket - putFiletoS3Bucket.py**

s3 = boto3.resource("s3")

bucket\_name = sys.argv[1]

object\_name = sys.argv[2]

try:

response = s3.Object(bucket\_name, object\_name).put(Body=open(object\_name, 'rb'))

print response

except Exception as error:

print error

PS C:\mic\ucsc\python\boto3> **python putFiletoS3Bucket.py myasuhara a.txt**

{u'ETag': '"4acc8e0d6e2084a8e32af7050071eba9"', 'ResponseMetadata': {'HTTPStatusCode': 200, 'RetryAttempts': 1, 'HostId': 'tEEN5+0mXAM/3Xc3YwU/GeVcb3UJ75lkU3Kq8k/473hwA8thw7lQt0eJgIhDEH10nXId2cjqlWg=', 'RequestId': '430A88CB75A21C98', 'HTTPHeaders': {'content-length': '0', 'x-amz-id-2': 'tEEN5+0mXAM/3Xc3YwU/GeVcb3UJ75lkU3Kq8k/473hwA8thw7lQt0eJgIhDEH10nXId2cjqlWg=', 'server': 'AmazonS3', 'x-amz-request-id': '430A88CB75A21C98', 'etag': '"4acc8e0d6e2084a8e32af7050071eba9"', 'date': 'Fri, 25 May 2018 21:51:13 GMT'}}}

%run -i "C:\Mic\UCSC\Python\boto3\**listS3Bucket.py**"

cf-templates-1b1hpaivisk3x-us-west-1

---

**myasuhara**

**---**

**a.txt**

projectx-bucket1-2018-05-25-1527284257

---

* **Delete contents from a bucket - deleteBuketContent.py**

s3 = boto3.resource("s3")

bucket\_name = sys.argv[1]

object\_name = sys.argv[2]

try:

response = s3.Object(bucket\_name, object\_name).put(Body=open(object\_name, 'rb'))

print response

except Exception as error:

print error

PS C:\mic\ucsc\python\boto3> python deleteBuketContent.py myasuhara

{'ResponseMetadata': {'HTTPStatusCode': **204**, 'RetryAttempts': 0, 'HostId': 'ixCfpwWYdss9D5mNZ8aF3egPRK36e12NSWWp9fdOUMwYmZrlWhnTOWfdH3QXpGU6EeSoMn8Fj6M=', 'RequestId': 'D1B51864E9BAEBE1', 'HTTPHeaders': {'x-amz-id-2': 'ixCfpwWYdss9D5mNZ8aF3egPRK36e12NSWWp9fdOUMwYmZrlWhnTOWfdH3QXpGU6EeSoMn8Fj6M=', 'date': 'Fri, 25 May 2018 22:01:54 GMT', 'x-amz-request-id': 'D1B51864E9BAEBE1', 'server': 'AmazonS3'}}}

**(3) Database**

List DB Instances

* **Create DB Instance - CreateDBInstance.py**

rds = boto3.client('rds')

try:

response = rds.create\_db\_instance(

DBInstanceIdentifier='dbserver',

MasterUsername='dbadmin',

MasterUserPassword='abcdefg123456789',

DBInstanceClass='db.t2.micro',

Engine='mariadb',

AllocatedStorage=5)

print response

%run -i "C:\Mic\UCSC\Python\boto3\listDBInstance.py"

File "C:\Mic\UCSC\Python\boto3\listDBInstance.py", line 14

for db in dbs['DBInstances']:

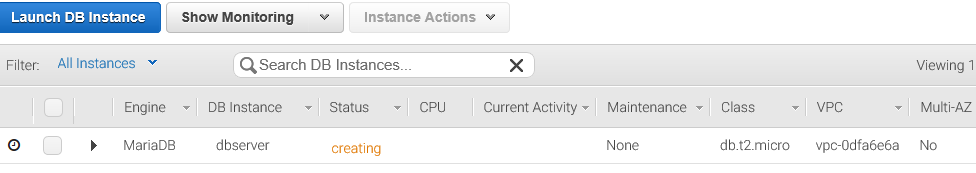
^

SyntaxError: invalid syntax

* **Create DB Instance - CreateDBInstance.py**

%run "C:\Mic\UCSC\Python\boto3\**CreateDBInstance.py**"

{u'DBInstance': {u'PubliclyAccessible': True, u'MasterUsername': 'dbadmin', u'MonitoringInterval': 0, u'LicenseModel': 'general-public-license', u'VpcSecurityGroups': [{u'Status': 'active', u'VpcSecurityGroupId': 'sg-895a6af0'}], u'CopyTagsToSnapshot': False, u'OptionGroupMemberships': [{u'Status': 'in-sync', u'OptionGroupName': 'default:mariadb-10-1'}], u'PendingModifiedValues': {u'MasterUserPassword': '\*\*\*\*'}, u'Engine': 'mariadb', u'MultiAZ': False, u'DBSecurityGroups': [], u'DBParameterGroups': [{u'DBParameterGroupName': 'default.mariadb10.1', u'ParameterApplyStatus': 'in-sync'}], u'PerformanceInsightsEnabled': False, u'AutoMinorVersionUpgrade': True, u'PreferredBackupWindow': '12:02-12:32', u'DBSubnetGroup': {u'Subnets': [{u'SubnetStatus': 'Active', u'SubnetIdentifier': 'subnet-b2d3bae9', u'SubnetAvailabilityZone': {u'Name': 'us-west-1a'}}, {u'SubnetStatus': 'Active', u'SubnetIdentifier': 'subnet-72cf9315', u'SubnetAvailabilityZone': {u'Name': 'us-west-1b'}}], u'DBSubnetGroupName': 'default', u'VpcId': 'vpc-0dfa6e6a', u'DBSubnetGroupDescription': 'default', u'SubnetGroupStatus': 'Complete'}, u'ReadReplicaDBInstanceIdentifiers': [], u'AllocatedStorage': 5, u'DBInstanceArn': 'arn:aws:rds:us-west-1:984717818485:db:dbserver', u'BackupRetentionPeriod': 1, u'PreferredMaintenanceWindow': 'tue:11:06-tue:11:36', u'DBInstanceStatus': 'creating', u'IAMDatabaseAuthenticationEnabled': False, u'EngineVersion': '10.1.31', u'DomainMemberships': [], u'StorageType': 'standard', u'DbiResourceId': 'db-EQ5QQD6UH7E3Q33J5KNLNJEU5E', u'CACertificateIdentifier': 'rds-ca-2015', u'StorageEncrypted': False, u'DBInstanceClass': 'db.t2.micro', u'DbInstancePort': 0, u'DBInstanceIdentifier': 'dbserver'}, 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '68893d0f-4b6f-4ef3-afd0-8c94b496e16d', 'HTTPHeaders': {'x-amzn-requestid': '68893d0f-4b6f-4ef3-afd0-8c94b496e16d', 'vary': 'Accept-Encoding', 'content-length': '3382', 'content-type': 'text/xml', 'date': 'Fri, 25 May 2018 22:20:07 GMT'}}}



* **Delete DB Instance – deleteDBInstance.py**

db = sys.argv[1]

rds = boto3.client('rds')

try:

response = rds.delete\_db\_instance(

DBInstanceIdentifier=db,

SkipFinalSnapshot=True)

print response

except Exception as error:

print error

%run "C:\Mic\UCSC\Python\boto3\deleteDBInstance.py"

IndexErrorTraceback (most recent call last)

C:\Mic\UCSC\Python\boto3\deleteDBInstance.py in <module>()

8 import boto3

9

---> 10 db = sys.argv[1]

11 rds = boto3.client('rds')

12 try:

IndexError: list index out of range

**7. Code Examples**

<https://boto3.readthedocs.io/en/latest/guide/examples.html>

This section provides code examples that demonstrate common Amazon Web Services scenarios using the Amazon Web Services (AWS) SDK for Python.

* [**Amazon CloudWatch Examples**](https://boto3.readthedocs.io/en/latest/guide/cw-examples.html)
* [**Amazon EC2 Examples**](https://boto3.readthedocs.io/en/latest/guide/ec2-examples.html)
* [**AWS Identity and Access Management Examples**](https://boto3.readthedocs.io/en/latest/guide/iam-examples.html)
* [**Amazon S3 Examples**](https://boto3.readthedocs.io/en/latest/guide/s3-examples.html)
* [**Amazon SQS Examples**](https://boto3.readthedocs.io/en/latest/guide/sqs-examples.html)

Code Samples:

[**https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/python/example\_code**](https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/python/example_code)

* 1. **EC2**

**(1) Managing Amazon EC2 Instances**

* **Describe Instances**

**import** **boto3**

ec2 **=** boto3**.**client('ec2')

response **=** ec2**.**describe\_instances()

**print**(response)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\ec2\describe\_instances.py"

{u'Reservations': [{u'Instances': [{u'Monitoring': {u'State': 'disabled'}, u'PublicDnsName': 'ec2-52-53-165-80.us-west-1.compute.amazonaws.com', u'State': {u'Code': 16, u'Name': 'running'}, u'EbsOptimized': False, u'LaunchTime': datetime.datetime(2018, 5, 25, 20, 51, 50, tzinfo=tzutc()), u'PublicIpAddress': '52.53.165.80', u'PrivateIpAddress': '172.31.8.13', u'ProductCodes': [], u'VpcId': 'vpc-0dfa6e6a', u'StateTransitionReason': '', u'InstanceId': 'i-08eb1b2d0cc695d70', u'EnaSupport': True, u'ImageId': 'ami-25110f45', u'PrivateDnsName': 'ip-172-31-8-13.us-west-1.compute.internal', u'SecurityGroups': [{u'GroupName': 'default', u'GroupId': 'sg-895a6af0'}], u'ClientToken': '', u'SubnetId': 'subnet-b2d3bae9', u'InstanceType': 't2.micro', u'NetworkInterfaces': [{u'Status': 'in-use', u'MacAddress': '06:88:b8:6c:47:9e', u'SourceDestCheck': True, u'VpcId': 'vpc-0dfa6e6a', u'Description': '', u'NetworkInterfaceId': 'eni-f2e45bf6', u'PrivateIpAddresses': [{u'PrivateDnsName': 'ip-172-31-8-13.us-west-1.compute.internal', u'PrivateIpAddress': '172.31.8.13', u'Primary': True, u'Association': {u'PublicIp': '52.53.165.80', u'PublicDnsName': 'ec2-52-53-165-80.us-west-1.compute.amazonaws.com', u'IpOwnerId': 'amazon'}}], u'PrivateDnsName': 'ip-172-31-8-13.us-west-1.compute.internal', u'Attachment': {u'Status': 'attached', u'DeviceIndex': 0, u'DeleteOnTermination': True, u'AttachmentId': 'eni-attach-25826cca', u'AttachTime': datetime.datetime(2018, 5, 25, 20, 51, 50, tzinfo=tzutc())}, u'Groups': [{u'GroupName': 'default', u'GroupId': 'sg-895a6af0'}], u'Ipv6Addresses': [], u'OwnerId': '984717818485', u'PrivateIpAddress': '172.31.8.13', u'SubnetId': 'subnet-b2d3bae9', u'Association': {u'PublicIp': '52.53.165.80', u'PublicDnsName': 'ec2-52-53-165-80.us-west-1.compute.amazonaws.com', u'IpOwnerId': 'amazon'}}], u'SourceDestCheck': True, u'Placement': {u'Tenancy': 'default', u'GroupName': '', u'AvailabilityZone': 'us-west-1a'}, u'Hypervisor': 'xen', u'BlockDeviceMappings': [{u'DeviceName': '/dev/xvda', u'Ebs': {u'Status': 'attached', u'DeleteOnTermination': True, u'VolumeId': 'vol-0e810ff7969ee4304', u'AttachTime': datetime.datetime(2018, 5, 25, 20, 51, 51, tzinfo=tzutc())}}], u'Architecture': 'x86\_64', u'RootDeviceType': 'ebs', u'RootDeviceName': '/dev/xvda', u'VirtualizationType': 'hvm', u'AmiLaunchIndex': 0}], u'ReservationId': 'r-0a4da5862a4e86b0e', u'Groups': [], u'OwnerId': '984717818485'},

* **Start and Stop Instances**

import sys

import boto3

from botocore.exceptions import ClientError

instance\_id = sys.argv[2]

action = sys.argv[1].upper()

ec2 = boto3.client('ec2')

if action == 'ON':

# Do a dryrun first to verify permissions

try:

ec2.start\_instances(InstanceIds=[instance\_id], DryRun=True)

except ClientError as e:

if 'DryRunOperation' not in str(e):

raise

# Dry run succeeded, run start\_instances without dryrun

try:

response = ec2.start\_instances(InstanceIds=[instance\_id], DryRun=False)

print(response)

except ClientError as e:

print(e)

else:

# Do a dryrun first to verify permissions

try:

ec2.stop\_instances(InstanceIds=[instance\_id], DryRun=True)

except ClientError as e:

if 'DryRunOperation' not in str(e):

raise

# Dry run succeeded, call stop\_instances without dryrun

try:

response = ec2.stop\_instances(InstanceIds=[instance\_id], DryRun=False)

print(response)

except ClientError as e:

print(e)

**[LEARN]**

**try and except -** [**http://www.pythonforbeginners.com/error-handling/python-try-and-except**](http://www.pythonforbeginners.com/error-handling/python-try-and-except)

If an error is encountered, a try block code execution is stopped and transferred down to the except block. In addition to using an except block after the try block, you can also use the finally block. The code in the finally block will be executed regardless of whether an exception occurs.

**raise: https - //stackoverflow.com/questions/13957829/how-to-use-raise-keyword-in-python**

You can use it to raise errors as part of error-checking:

if (a < b):

raise ValueError()

Or handle some errors, and then pass them on as part of error-handling:

try:

f = open('file.txt', 'r')

except IOError:

# do some processing here

# and then pass the error on

raise

**sys.argv :** [**http://www.pythonforbeginners.com/system/python-sys-argv**](http://www.pythonforbeginners.com/system/python-sys-argv)

sys.argv is a list in Python, which contains the command-line arguments passed to the script. With the len(sys.argv) function you can count the number of arguments. If you are gonna work with command line arguments, you probably want to use sys.argv. To use sys.argv, you will first have to import the sys module.

PS C:\mic\ucsc\python\boto3\sample\ec2> **python starting\_\_and\_stopping\_instances.py 'OFF' 'i-08eb1b2d0cc695d70'**

{u'StoppingInstances': [{u'InstanceId': 'i-08eb1b2d0cc695d70', u'CurrentState': {u'Code': 64, u'Name': 'stopping'}, u'PreviousState': {u'Code': 16, u'Name': 'running'}}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '9fa9aac1-4461-43b4-a440-fc8fe8403c85', 'HTTPHeaders': {'date': 'Sat, 26 May 2018 00:09:58 GMT', 'content-length': '579', 'content-type': 'text/xml;charset=UTF-8', 'server': 'AmazonEC2'}}}

PS C:\mic\ucsc\python\boto3\sample\ec2> python starting\_\_and\_stopping\_instances.py **'ON' 'i-08eb1b2d0cc695d70'**

{u'StartingInstances': [{u'InstanceId': 'i-08eb1b2d0cc695d70', u'CurrentState': {u'Code': 0, u'Name': 'pending'}, u'PreviousState': {u'Code': 80, u'Name': 'stopped'}}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': 'a512232c-a670-4fab-8448-15460bf5dbb3', 'HTTPHeaders': {'date': 'Sat, 26 May 2018 00:11:31 GMT', 'content-length': '579', 'content-type': 'text/xml;charset=UTF-8', 'server': 'AmazonEC2'}}}

* **Reboot Instances – rebooting.py**

**import** **boto3**

**from** **botocore.exceptions** **import** ClientError

ec2 **=** boto3**.**client('ec2')

**try**:

ec2**.**reboot\_instances(InstanceIds**=**['i-08eb1b2d0cc695d70'], DryRun**=**True)

**except** ClientError **as** e:

**if** 'DryRunOperation' **not** **in** str(e):

**print**("You don't have permission to reboot instances.")

**raise**

**try**:

response **=** ec2**.**reboot\_instances(InstanceIds**=**['i-08eb1b2d0cc695d70'], DryRun**=**False)

**print**('Success', response)

**except** ClientError **as** e:

**print**('Error', e)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**rebooting.py**"

('**Success**', {'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '78649633-04f2-4a15-9e11-34f87399adfc', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Sat, 26 May 2018 00:37:08 GMT'}}})

**(2) Working with Amazon EC2 Key Pairs**

* **Create a Key Pair**

**import** **boto3**

ec2 **=** boto3**.**client('ec2')

response **=** ec2**.**create\_key\_pair(KeyName**=**'KEY\_PAIR\_NAME')

**print**(response)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**create\_keypair.py"**

{u'KeyMaterial': '-----BEGIN RSA PRIVATE KEY-----\nMIIEpAIBAAKCAQEA1Md/XBpUN9077DUxcw3Uyk2DjgeKR1pxmyWNXcpX7EOgD2/mbCQjaYS7c6Un\nua2c1mdeAIMmXlv5eqfTgc07nqIZE+/h6RQ6o6lS+ZbiBOHZUS7Ym0Xh7KSabC8yLZBNgq/2RGjE\n/1DrtIrxP5/IqEjhfE2b9wVvxeI91zO4T5P00dWKBfY6N1eyzdYEmJvKGg4JkWCAWD0d8H4DQyoJ\nfQybcbweKwj5u6J2dX+pJMrJnVGcQitai9HcmOyzgOAHfawUhVwfus4Zdz7O05JW197VCv/xXK3e\nLC1wm9wC1GhnN18nB1JELgZybipZLKqVTLH5TU4SE8RInyvvEJ9DhQIDAQABAoIBAC7qNcim/X67\nxuST7e1CVXKtmJ4OR4zoAjiFFSBi5vPvDTGaKKIbbbtsSrB678Fe6D/V1sIiKKmRefZzXo02JTka\n4JCmyZbMoQlI4kSp4ffVnEbsH1JNHD1DJdFIrrWhJZX1GkWHHDh3LSFHqV2v61w/hYgUzJiwmC53\nbc8vViolO/xaahocNzIdxIpkiWpDGDQy+GCH2ltR52og8IRlow5mrHC6erSUGzOV2CprKv8ZAzNK\n/+O1RR4RC9E4jMnUTvA5ux2HeJKZATQAIsYCv3vLM92V7EfCEEFVjujAzX+4GFJ+70fv7L1weINd\n7m1iYzvHs6m/bkrimj4L+8MiG9kCgYEA/pdioMcAp7268wwh8JIbN1rhJz+q9AG5cTAIdeIfVbxK\nlLnfl4fKULxbL6kNWKfv5V6jGYynSuwwLmf/klVBiSN9QdUSKk/730hNiYESFQvcNvp3IY3NB4xH\njtr5uJguGMvCJiuhQwL2Su8znjLaztnR6CihXhZE8zRj3nJ4g+sCgYEA1fTjQnSrWnNJyjTXDbRK\nLRFktyvT5e0QApGcchuXYQRwVuCOnq14a0d4rIVPUUM0x1OVVSFaGCOdZSjyjPMMo8Jy751biAkt\nOLZ7zFpgIPD9WBJGPIVQc1p0GovYt/743JnCdaDaITXiTjDu+WuJHJ4fLqID+QHW+UfyhDbeKk8C\ngYEAwRi7Eqy9P3y1DeD24QtAH0ULnoQ4umeQalv/HsIJ+DIe8xzjjH/OoFusYcK7v7FoT7TuydMX\nk8j+ZkQ04J1PXp+PQl5EL9mIdPrW7GL7/wCG3yArOUTe7EmmgKBvZWXvGJl6wQ64eiYF/iHdKbDy\nqo8lGQuwELedRkVHNhHmfE8CgYEAgkvTRnpUjA3fpyPMv4zHC3EnO6GWNtTKxjW054zEAJI6c73n\nPNz5k6N/6th4eE2MS8MsEx7vhfnSdXOdWpb9YJEfe6sBvfOQtZP0CnwTRJxePI9MdLjfSJCSWkRG\nI69EENwWn5nonQb6nZGRRxXk9yeYZ8bE0pT7PO5C05+yZmMCgYBOwRXIfTA+IRr3/hNploReYZCH\nsLGJXYnJizlaP6PMDZLadDVkqaXrKv/wLgpDI9/CIc68LuVG+d5BqtUawBC7Myc9hCLH6gHdRrmt\nOUVw7FPPQ1CCKOMVyWGEaHUs5msiwWZlIRYJx75ntzP3BXsYSmW8JSiiGC/MUvPhEC34Og==\n-----END RSA PRIVATE KEY-----', u'KeyName': 'KEY\_PAIR\_NAME', u'KeyFingerprint': 'f6:f5:22:33:14:11:d7:38:2a:6a:db:1c:a1:fb:dc:8d:7b:dd:fa:c2', 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '48075a06-d9ea-405d-b18c-359fe9c9bdd5', 'HTTPHeaders': {'date': 'Sat, 26 May 2018 00:47:52 GMT', 'content-length': '2041', 'content-type': 'text/xml;charset=UTF-8', 'server': 'AmazonEC2'}}}

* **Delete a Key Pair**

**import** **boto3**

ec2 **=** boto3**.**client('ec2')

response **=** ec2**.**delete\_key\_pair(KeyName**=**'KEY\_PAIR\_NAME')

**print**(response)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**delete\_keypair.py**"

{'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': 'c33509eb-c608-4e7a-b0e1-1ad0a3f37f38', 'HTTPHeaders': {'date': 'Sat, 26 May 2018 00:48:44 GMT', 'content-length': '227', 'content-type': 'text/xml;charset=UTF-8', 'server': 'AmazonEC2'}}}

**(3) Describe Amazon EC2 Regions and Availability Zones**

* **Describe Regions and Availability Zones**

**import** **boto3**

ec2 **=** boto3**.**client('ec2')

*# Retrieves all regions/endpoints that work with EC2*

response **=** ec2**.**describe\_regions()

**print**('Regions:', response['Regions'])

*# Retrieves availability zones only for region of the ec2 object*

response **=** ec2**.**describe\_availability\_zones()

**print**('Availability Zones:', response['AvailabilityZones'])

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**describe\_regions\_azs.py**"

('Regions:', [{u'Endpoint': 'ec2.ap-south-1.amazonaws.com', u'RegionName': 'ap-south-1'}, {u'Endpoint': 'ec2.eu-west-3.amazonaws.com', u'RegionName': 'eu-west-3'}, {u'Endpoint': 'ec2.eu-west-2.amazonaws.com', u'RegionName': 'eu-west-2'}, {u'Endpoint': 'ec2.eu-west-1.amazonaws.com', u'RegionName': 'eu-west-1'}, {u'Endpoint': 'ec2.ap-northeast-2.amazonaws.com', u'RegionName': 'ap-northeast-2'}, {u'Endpoint': 'ec2.ap-northeast-1.amazonaws.com', u'RegionName': 'ap-northeast-1'}, {u'Endpoint': 'ec2.sa-east-1.amazonaws.com', u'RegionName': 'sa-east-1'}, {u'Endpoint': 'ec2.ca-central-1.amazonaws.com', u'RegionName': 'ca-central-1'}, {u'Endpoint': 'ec2.ap-southeast-1.amazonaws.com', u'RegionName': 'ap-southeast-1'}, {u'Endpoint': 'ec2.ap-southeast-2.amazonaws.com', u'RegionName': 'ap-southeast-2'}, {u'Endpoint': 'ec2.eu-central-1.amazonaws.com', u'RegionName': 'eu-central-1'}, {u'Endpoint': 'ec2.us-east-1.amazonaws.com', u'RegionName': 'us-east-1'}, {u'Endpoint': 'ec2.us-east-2.amazonaws.com', u'RegionName': 'us-east-2'}, {u'Endpoint': 'ec2.us-west-1.amazonaws.com', u'RegionName': 'us-west-1'}, {u'Endpoint': 'ec2.us-west-2.amazonaws.com', u'RegionName': 'us-west-2'}])

('Availability Zones:', [{u'State': 'available', u'ZoneName': 'us-west-1a', u'Messages': [], u'RegionName': 'us-west-1'}, {u'State': 'available', u'ZoneName': 'us-west-1b', u'Messages': [], u'RegionName': 'us-west-1'}])

**(4) Working with Security Groups in Amazon EC2**

* **Describe Security Groups**

**import** **boto3**

**from** **botocore.exceptions** **import** ClientError

ec2 **=** boto3**.**client('ec2')

**try**:

response **=** ec2**.**describe\_security\_groups(GroupIds**=**['sg-895a6af0'])

**print**(response)

**except** ClientError **as** e:

**print**(e)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**describe\_security\_groups.py**"

{u'SecurityGroups': [{u'IpPermissionsEgress': [{u'IpProtocol': '-1', u'PrefixListIds': [], u'IpRanges': [{u'CidrIp': '0.0.0.0/0'}], u'UserIdGroupPairs': [], u'Ipv6Ranges': []}], u'Description': 'default VPC security group', u'IpPermissions': [{u'IpProtocol': '-1', u'PrefixListIds': [], u'IpRanges': [], u'UserIdGroupPairs': [{u'UserId': '984717818485', u'GroupId': 'sg-895a6af0'}], u'Ipv6Ranges': []}], u'GroupName': 'default', u'VpcId': 'vpc-0dfa6e6a', u'OwnerId': '984717818485', u'GroupId': 'sg-895a6af0'}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '45175be0-951c-4a7c-ad79-86edf282b324', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Sat, 26 May 2018 01:02:26 GMT'}}}

* **Create a Security Group and Rules**

import boto3

from botocore.exceptions import ClientError

ec2 = boto3.client('ec2')

response = ec2.describe\_vpcs()

vpc\_id = response.get('Vpcs', [{}])[0].get('VpcId', '')

try:

response = ec2.create\_security\_group(GroupName='SECURITY\_Mic,

Description='DESCRIPTION',

VpcId=vpc\_id)

security\_group\_id = response['GroupId']

print('Security Group Created %s in vpc %s.' % (security\_group\_id, vpc\_id))

data = ec2.authorize\_security\_group\_ingress(

GroupId=security\_group\_id,

IpPermissions=[

{'IpProtocol': 'tcp',

'FromPort': 80,

'ToPort': 80,

'IpRanges': [{'CidrIp': '0.0.0.0/0'}]},

{'IpProtocol': 'tcp',

'FromPort': 22,

'ToPort': 22,

'IpRanges': [{'CidrIp': '0.0.0.0/0'}]}

])

print('Ingress Successfully Set %s' % data)

except ClientError as e:

print(e)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**create\_security\_group.py"**

Security Group Created **sg-c610cdbe** in vpc vpc-0dfa6e6a.

Ingress Successfully Set {'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': 'db705329-fedf-437c-89c6-63e9ce269e09', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Sat, 26 May 2018 01:06:07 GMT'}}}

* **Delete a Security Group**

**import** **boto3**

**from** **botocore.exceptions** **import** ClientError

*# Create EC2 client*

ec2 **=** boto3**.**client('ec2')

*# Delete security group*

**try**:

response **=** ec2**.**delete\_security\_group(GroupId**=**'**sg-c610cdbe’**)

**print**('Security Group Deleted')

**except** ClientError **as** e:

**print**(e)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\delete\_security\_group.py"

Security Group Deleted

**(5) Using Elastic IP Addresses in Amazon EC2**

* **Describe Elastic IP Addresses**

**import** **boto3**

ec2 **=** boto3**.**client('ec2')

filters **=** [

{'Name': 'domain', 'Values': ['vpc']}

]

response **=** ec2**.**describe\_addresses(Filters**=**filters)

**print**(response)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**describe\_addresses.py**"

{u'Addresses': [], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '8b24a53d-7105-4d3a-8073-729b4c13516e', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Sat, 26 May 2018 02:20:32 GMT'}}}

* **Allocate and Associate an Elastic IP Address with an Amazon EC2 Instance**

**import** **boto3**

**from** **botocore.exceptions** **import** ClientError

ec2 **=** boto3**.**client('ec2')

**try**:

allocation **=** ec2**.**allocate\_address(Domain**=**'vpc')

response **=** ec2**.**associate\_address(AllocationId**=**allocation['AllocationId'],

InstanceId**=**'i-08eb1b2d0cc695d70')

**print**(response)

**except** ClientError **as** e:

**print**(e)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\ec2\**allocate\_address.py**"

{u'AssociationId': 'eipassoc-727288a8', 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '77db540f-233a-4c39-9321-177af21373aa', 'HTTPHeaders': {'transfer-encoding': 'chunked', 'vary': 'Accept-Encoding', 'server': 'AmazonEC2', 'content-type': 'text/xml;charset=UTF-8', 'date': 'Sat, 26 May 2018 02:23:57 GMT'}}}

* **Release an Elastic IP Address**

**import** **boto3**

**from** **botocore.exceptions** **import** ClientError

ec2 **=** boto3**.**client('ec2')

**try**:

response **=** ec2**.**release\_address(AllocationId**=’**eipalloc-6f7af652’)

**print**('Address released')

**except** ClientError **as** e:

**print**(e)

%run "C:\Mic\UCSC\Python\boto3\Sample\ec2\**release\_elastic\_ip.py**"

Address released

* 1. **IAM**

**(1) Managing IAM Users**

* **Create a User**

iam = boto3.client('iam')

# Create user

response = iam.create\_user(

UserName='KenYasuhara'

)

print(response)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\create\_user.py"

{u'User': {u'UserName': 'KenYasuhara', u'Path': '/', u'CreateDate': datetime.datetime(2018, 5, 31, 1, 6, 36, 292000, tzinfo=tzutc()), u'UserId': 'AIDAIGZMK324AUGTO6NZI', u'Arn': 'arn:aws:iam::984717818485:user/KenYasuhara'}, 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': 'dda5067b-646e-11e8-8a14-65705c3116c1', 'HTTPHeaders': {'x-amzn-requestid': 'dda5067b-646e-11e8-8a14-65705c3116c1', 'date': 'Thu, 31 May 2018 01:06:36 GMT', 'content-length': '487', 'content-type': 'text/xml'}}}

* **List Users in Your Account**

# Create IAM client

iam = boto3.client('iam')

# List users with the pagination interface

paginator = iam.get\_paginator('list\_users')

for response in paginator.paginate():

print(response)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\list\_users.py"

{u'Users': [{u'UserName': 'admin', u'PasswordLastUsed': datetime.datetime(2018, 4, 18, 0, 16, 38, tzinfo=tzutc()), u'CreateDate': datetime.datetime(2018, 3, 30, 23, 17, 52, tzinfo=tzutc()), u'UserId': 'AIDAJNJWU6FITP2FY4FYQ', u'Path': '/', u'Arn': 'arn:aws:iam::984717818485:user/admin'}, {u'UserName': 'KenYasuhara', u'Path': '/', u'CreateDate': datetime.datetime(2018, 5, 31, 1, 6, 36, tzinfo=tzutc()), u'UserId': 'AIDAIGZMK324AUGTO6NZI', u'Arn': 'arn:aws:iam::984717818485:user/KenYasuhara'}, {u'UserName': 'mic', u'PasswordLastUsed': datetime.datetime(2018, 4, 7, 14, 17, 27, tzinfo=tzutc()), u'CreateDate': datetime.datetime(2018, 4, 7, 14, 15, 32, tzinfo=tzutc()), u'UserId': 'AIDAIMUU4HV2VLL3JLZ24', u'Path': '/', u'Arn': 'arn:aws:iam::984717818485:user/mic'}, {u'UserName': 'mycli', u'Path': '/', u'CreateDate': datetime.datetime(2018, 4, 7, 23, 12, 1, tzinfo=tzutc()), u'UserId': 'AIDAIRGPT64EGT6LGVRGW', u'Arn': 'arn:aws:iam::984717818485:user/mycli'}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '59a87675-646f-11e8-9840-612b7c021f50', 'HTTPHeaders': {'x-amzn-requestid': '59a87675-646f-11e8-9840-612b7c021f50', 'date': 'Thu, 31 May 2018 01:10:03 GMT', 'content-length': '1425', 'content-type': 'text/xml'}}, u'IsTruncated': False}

* **Update a User's Name**

iam = boto3.client('iam')

# Update a user name

iam.update\_user(

UserName='KenYasuhara',

NewUserName='NanamiYasuhara'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\**update\_user.py**"

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\**list\_users.py**"

{u'Users': [{u'UserName': 'admin', u'PasswordLastUsed': datetime.datetime(2018, 4, 18, 0, 16, 38, tzinfo=tzutc()), u'CreateDate': datetime.datetime(2018, 3, 30, 23, 17, 52, tzinfo=tzutc()), u'UserId': 'AIDAJNJWU6FITP2FY4FYQ', u'Path': '/', u'Arn': 'arn:aws:iam::984717818485:user/admin'}, {u'UserName': 'mic', u'PasswordLastUsed': datetime.datetime(2018, 4, 7, 14, 17, 27, tzinfo=tzutc()), u'CreateDate': datetime.datetime(2018, 4, 7, 14, 15, 32, tzinfo=tzutc()), u'UserId': 'AIDAIMUU4HV2VLL3JLZ24', u'Path': '/', u'Arn': 'arn:aws:iam::984717818485:user/mic'}, {u'UserName': 'mycli', u'Path': '/', u'CreateDate': datetime.datetime(2018, 4, 7, 23, 12, 1, tzinfo=tzutc()), u'UserId': 'AIDAIRGPT64EGT6LGVRGW', u'Arn': 'arn:aws:iam::984717818485:user/mycli'}, {u'UserName': 'NanamiYasuhara', u'Path': '/', u'CreateDate': datetime.datetime(2018, 5, 31, 1, 6, 36, tzinfo=tzutc()), u'UserId': 'AIDAIGZMK324AUGTO6NZI', u'Arn': 'arn:aws:iam::984717818485:user/NanamiYasuhara'}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '600bc92d-6470-11e8-af7e-cbdd9950e9ec', 'HTTPHeaders': {'x-amzn-requestid': '600bc92d-6470-11e8-af7e-cbdd9950e9ec', 'date': 'Thu, 31 May 2018 01:17:23 GMT', 'content-length': '1431', 'content-type': 'text/xml'}}, u'IsTruncated': False}

* **Delete a User**

iam = boto3.client('iam')

# Delete a user

iam.delete\_user(

UserName='NanamiYasuhara'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\delete\_user.py"

**(2) Working with IAM Policies**

* **Create an IAM Policy**

import sys

sys.path.extend(['C:\\Python27\\Lib\\idlelib',

'C:\\Windows\\SYSTEM32\\python27.zip', 'C:\\Python27\\DLLs',

'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win',

'C:\\Python27\\lib\\lib-tk', 'C:\\Python27',

'C:\\Python27\\lib\\site-packages'])

import boto3

import json

# Create IAM client

iam = boto3.client('iam')

# Create a policy

my\_managed\_policy = {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "logs:CreateLogGroup",

"Resource": "RESOURCE\_ARN"

},

{

"Effect": "Allow",

"Action": [

"dynamodb:DeleteItem",

"dynamodb:GetItem",

"dynamodb:PutItem",

"dynamodb:Scan",

"dynamodb:UpdateItem"

],

"Resource": "RESOURCE\_ARN"

}

]

}

response = iam.create\_policy(

PolicyName='myDynamoDBPolicy',

PolicyDocument=json.dumps(my\_managed\_policy)

)

print(response)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\create\_policy.py"

MalformedPolicyDocumentExceptionTraceback (most recent call last)

C:\Mic\UCSC\Python\boto3\Sample\iam\create\_policy.py in <module>()

49 response = iam.create\_policy(

50 PolicyName='myDynamoDBPolicy',

---> 51 PolicyDocument=json.dumps(my\_managed\_policy)

52 )

53 print(response)

C:\Python27\lib\site-packages\botocore\client.pyc in \_api\_call(self, \*args, \*\*kwargs)

312 "%s() only accepts keyword arguments." % py\_operation\_name)

313 # The "self" in this scope is referring to the BaseClient.

--> 314 return self.\_make\_api\_call(operation\_name, kwargs)

315

316 \_api\_call.\_\_name\_\_ = str(py\_operation\_name)

C:\Python27\lib\site-packages\botocore\client.pyc in \_make\_api\_call(self, operation\_name, api\_params)

610 error\_code = parsed\_response.get("Error", {}).get("Code")

611 error\_class = self.exceptions.from\_code(error\_code)

--> 612 raise error\_class(parsed\_response, operation\_name)

613 else:

614 return parsed\_response

MalformedPolicyDocumentException: An error occurred (MalformedPolicyDocument) when calling the CreatePolicy operation: Resource RESOURCE\_ARN must be in ARN format or "\*".

* **Get an IAM Policy**

iam = boto3.client('iam')

# Get a policy

response = iam.get\_policy(

PolicyArn='arn:aws:iam::aws:policy/AWSLambdaExecute'

)

print(response['Policy'])

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\get\_policy.py"

{u'PolicyName': 'AWSLambdaExecute', u'Description': 'Provides Put, Get access to S3 and full access to CloudWatch Logs.', u'CreateDate': datetime.datetime(2015, 2, 6, 18, 40, 46, tzinfo=tzutc()), u'AttachmentCount': 0, u'IsAttachable': True, u'PolicyId': 'ANPAJE5FX7FQZSU5XAKGO', u'DefaultVersionId': 'v1', u'Path': '/', u'Arn': 'arn:aws:iam::aws:policy/AWSLambdaExecute', u'UpdateDate': datetime.datetime(2015, 2, 6, 18, 40, 46, tzinfo=tzutc())}

* **Attach a Managed Role Policy**

*# Create IAM client*

iam **=** boto3**.**client('iam')

*# Attach a role policy*

iam**.**attach\_role\_policy(

PolicyArn**=**'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',

RoleName**=**'AmazonDynamoDBFullAccess'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\attach\_role\_policy.py"

NoSuchEntityExceptionTraceback (most recent call last)

C:\Mic\UCSC\Python\boto3\Sample\iam\attach\_role\_policy.py in <module>()

27 iam.attach\_role\_policy(

28 PolicyArn='arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',

---> 29 RoleName='AmazonDynamoDBFullAccess'

30 )

C:\Python27\lib\site-packages\botocore\client.pyc in \_api\_call(self, \*args, \*\*kwargs)

312 "%s() only accepts keyword arguments." % py\_operation\_name)

313 # The "self" in this scope is referring to the BaseClient.

--> 314 return self.\_make\_api\_call(operation\_name, kwargs)

315

316 \_api\_call.\_\_name\_\_ = str(py\_operation\_name)

C:\Python27\lib\site-packages\botocore\client.pyc in \_make\_api\_call(self, operation\_name, api\_params)

610 error\_code = parsed\_response.get("Error", {}).get("Code")

611 error\_class = self.exceptions.from\_code(error\_code)

--> 612 raise error\_class(parsed\_response, operation\_name)

613 else:

614 return parsed\_response

NoSuchEntityException: An error occurred (NoSuchEntity) when calling the AttachRolePolicy operation: Role not found for AmazonDynamoDBFullAccess

* **Detach a Managed Role Policy**

*# Create IAM client*

iam **=** boto3**.**client('iam')

*# Detach a role policy*

iam**.**detach\_role\_policy(

PolicyArn**=**'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',

RoleName**=**'AmazonDynamoDBFullAccess'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\detach\_role\_policy.py"

NoSuchEntityExceptionTraceback (most recent call last)

C:\Mic\UCSC\Python\boto3\Sample\iam\detach\_role\_policy.py in <module>()

26 iam.detach\_role\_policy(

27 PolicyArn='arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess',

---> 28 RoleName='AmazonDynamoDBFullAccess'

29 )

C:\Python27\lib\site-packages\botocore\client.pyc in \_api\_call(self, \*args, \*\*kwargs)

312 "%s() only accepts keyword arguments." % py\_operation\_name)

313 # The "self" in this scope is referring to the BaseClient.

--> 314 return self.\_make\_api\_call(operation\_name, kwargs)

315

316 \_api\_call.\_\_name\_\_ = str(py\_operation\_name)

C:\Python27\lib\site-packages\botocore\client.pyc in \_make\_api\_call(self, operation\_name, api\_params)

610 error\_code = parsed\_response.get("Error", {}).get("Code")

611 error\_class = self.exceptions.from\_code(error\_code)

--> 612 raise error\_class(parsed\_response, operation\_name)

613 else:

614 return parsed\_response

NoSuchEntityException: An error occurred (NoSuchEntity) when calling the DetachRolePolicy operation: Role not found for AmazonDynamoDBFullAccess

**(3) Managing IAM Access Keys**

* **Create Access Keys for a User**

# Create IAM client

iam = boto3.client('iam')

# Create an access key

response = iam.create\_access\_key(

UserName='IAM\_USER\_NAME'

)

print(response['AccessKey'])

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\create\_access\_key.py"

{u'UserName': 'nanami', u'Status': 'Active', u'CreateDate': datetime.datetime(2018, 6, 7, 23, 31, 35, 948000, tzinfo=tzutc()), u'SecretAccessKey': 'EocpMl9hpSg+Sf/ocmp9E80UHMx16vKr+Vej3MuE', u'AccessKeyId': 'AKIAI3NYIYQCYY7W2Z6Q'}

* **List a User's Access Keys**

# Create IAM client

iam = boto3.client('iam')

# List access keys through the pagination interface.

paginator = iam.get\_paginator('list\_access\_keys')

for response in paginator.paginate(UserName='IAM\_USER\_NAME'):

print(response)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\list\_access\_keys.py"

{u'AccessKeyMetadata': [{u'UserName': 'nanami', u'Status': 'Active', u'CreateDate': datetime.datetime(2018, 6, 7, 23, 31, 19, tzinfo=tzutc()), u'AccessKeyId': 'AKIAIEEY5YL7H6QBZ4AQ'}, {u'UserName': 'nanami', u'Status': 'Active', u'CreateDate': datetime.datetime(2018, 6, 7, 23, 31, 35, tzinfo=tzutc()), u'AccessKeyId': 'AKIAI3NYIYQCYY7W2Z6Q'}], 'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId': '6d459b25-6aab-11e8-a14d-91354768a5f8', 'HTTPHeaders': {'x-amzn-requestid': '6d459b25-6aab-11e8-a14d-91354768a5f8', 'date': 'Thu, 07 Jun 2018 23:35:13 GMT', 'content-length': '763', 'content-type': 'text/xml'}}, u'IsTruncated': False}

* **Get the Access Key Last Used**

# Create IAM client

iam = boto3.client('iam')

# Get last use of access key

response = iam.get\_access\_key\_last\_used(

AccessKeyId='AKIAI3NYIYQCYY7W2Z6Q'

)

print(response['AccessKeyLastUsed'])

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\get\_access\_key\_last\_used.py"

{u'Region': 'N/A', u'ServiceName': 'N/A'}

* **Update Access Key Status**

# Create IAM client

iam = boto3.client('iam')

# Update access key to be inactive

iam.update\_access\_key(

AccessKeyId='AKIAI3NYIYQCYY7W2Z6Q',

Status='Inactive',

UserName='nanami'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\Update\_Access\_Key\_Status.py"

iam.update\_access\_key(

AccessKeyId='AKIAI3NYIYQCYY7W2Z6Q',

Status='Active',

UserName='nanami'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\Update\_Access\_Key\_Status.py"

* **Delete an Access Key**

# Create IAM client

iam = boto3.client('iam')

# Delete access key

iam.delete\_access\_key(

AccessKeyId='AKIAI3NYIYQCYY7W2Z6Q',

UserName='nanami'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\delete\_access\_key.py"

**(4) Managing IAM Account Aliases**

* **Create an Account Alias**

# Create IAM client

iam = boto3.client('iam')

# Create an account alias

iam.create\_account\_alias(

AccountAlias='alias-mic'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\create\_account\_alias.py"

* **List an Account Alias**

# Create IAM client

iam = boto3.client('iam')

# List account aliases through the pagination interface

paginator = iam.get\_paginator('list\_account\_aliases')

for response in paginator.paginate():

print(response['AccountAliases'])

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\list\_account\_aliases.py"

['alias-mic']

* **Delete an Account Alias**

# Create IAM client

iam = boto3.client('iam')

# Delete an account alias

iam.delete\_account\_alias(

AccountAlias='alias-mic'

)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\iam\delete\_account\_alias.py"

* 1. **Amazon S3**

**Creating and Using Amazon S3 Buckets**

* **Display a List of Amazon S3 Buckets**

# Create an S3 client

s3 = boto3.client('s3')

# Call S3 to list current buckets

response = s3.list\_buckets()

# Get a list of all bucket names from the response

buckets = [bucket['Name'] for bucket in response['Buckets']]

# Print out the bucket list

print("Bucket List: %s" % buckets)

%run -i "C:\Mic\UCSC\Python\boto3\Sample\s3\s3-python-example-list-buckets.py"

Bucket List: ['cf-templates-1b1hpaivisk3x-us-west-1', 'cf-templates-1b1hpaivisk3x-us-west-2', 'edureka-s3appbucket-10tjfieolas3d', 'elasticbeanstalk-us-east-2-984717818485', 'mic-20180525', 'myasuhara', 'projectx-bucket1-2018-05-25-1527284257']

* **Create an Amazon S3 Bucket**

s3 = boto3.client('s3')

s3.create\_bucket(Bucket='my-boto3-20180607',

**CreateBucketConfiguration={'LocationConstraint': 'us-west-1'**})

%run -i "C:\Mic\UCSC\Python\boto3\Sample\s3\s3-python-example-create-bucket.py"