# **Section 1: Introduction and Environmet setup for boto3**

## **Boto3 Environment setup on Linux Server…**

Configure credentials of your AWS account on windows server using awscli commands.

* Install awscli

pip3 install awscli

* Configure root/IAM user access-keys/credentials using:

**aws configure --profile root**

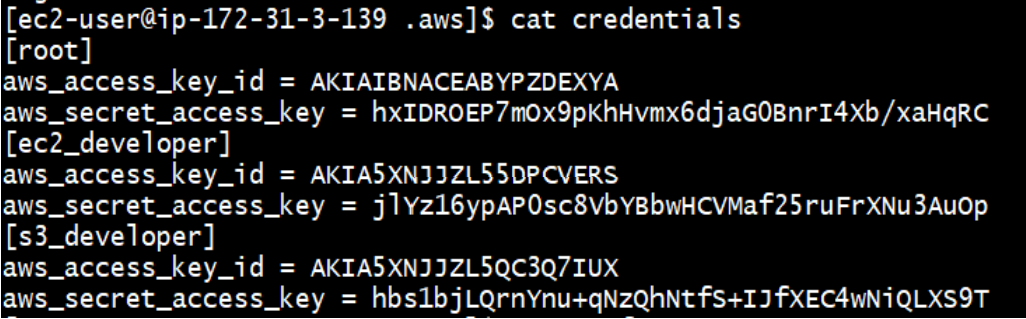
**aws configure --profile non\_prod**

**aws configure --profile prod**

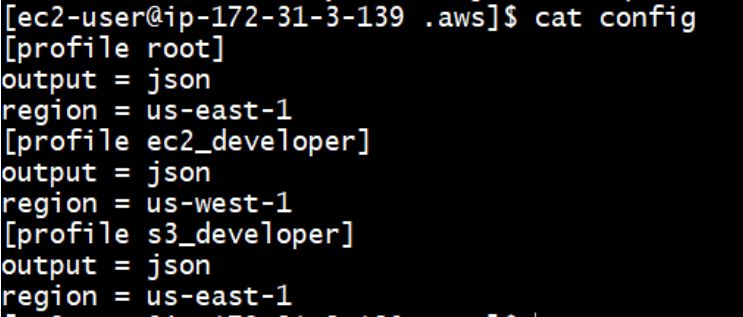
* Verifying Boto3 Environment setup:

Our Environment setup is: Python3.7.4, boto3 and awscli

.aws> cat credentials



.aws> cat config



Manual Steps to see/list all iam users:

========================================

**step1: Get AWS Management Console**

**Step2: Get IAM Console**

Options: Users, Groups, roles......

===============================================

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

iam\_con=aws\_mag\_con.resource(**'iam'**)

for each\_user in iam\_con.users.all():

print(each\_user.name)

================================================

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

s3\_con=aws\_mag\_con.resource(**'s3'**)

=================================================

## **Concepts of boto3**

* **Session**
* **Resource**
* **Client**
* **Meta**
* **Collections**
* **Waiters**
* **Paginators**
* **Session:**
* It is an AWS Management Console in our terms.
* stores configuration information (primarily credentials)
* allows us to create service clients and resources
* boto3 creates a default session for us when needed
* **Resource and Client:**
* We can create particular AWS Service Console like iam console, ec2 console, sns console…
* Resource is higher-level object-oriented service access and it is available for some of the aws services.
* Client is low-level service access

Boto3 Session Concept:

The are two types of Sessions. They are:

* Custom Session
* Default Session

### **about\_session\_client\_resouce**

Manual Steps to see/list all iam users:

========================================

step1: Get AWS Management Console

Step2: Get IAM Console

Options: Users, Groups, roles......

========================================

import boto3

aws\_mag\_con\_root=boto3.session.Session(profile\_name="root")

#aws\_mag\_con\_root=boto3.session.Session(profile\_name="ec2\_developer")

iam\_con\_re=aws\_mag\_con\_root.resource(service\_name='iam',region\_name="us-east-2")

iam\_con\_cli=aws\_mag\_con\_root.client(service\_name='iam',region\_name="us-east-2")

**#Listing iam users with resource object:**

for each\_user in iam\_con\_re.users.all():

print(each\_user.name)

**#Listing iam users with client object:**

for each in iam\_con\_cli.list\_users()['Users']:

print(each['UserName'])

======================================================

### **custom\_and\_default\_sessions**

aws configure --profile p\_name

[default]

aws\_access\_key\_id = KKIAJRCMPX5GEZQLWWTA

aws\_secret\_access\_key = d7CKM9XCllDPaTNC5QbWO4Bgul5EXEPMu+T6oXX/

[root]

aws\_access\_key\_id = AKIAJRCMPX5GEZQLWWTA

aws\_secret\_access\_key = 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

[ec2\_developer]

aws\_access\_key\_id = AKIA5XNJJZL57MQ4WGWM

aws\_secret\_access\_key = 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

[s3\_developer]

aws\_access\_key\_id = AKIA5XNJJZL5QC3Q7IUX

aws\_secret\_access\_key = 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

====================================================================

**Custom Session:**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

iam\_con\_re=aws\_mag\_con.resource(service\_name='iam',region\_name="us-east-2")

iam\_con\_client=aws\_mag\_con.client(service\_name='iam',region\_name="us-east-2")

=====================================================================

**Default:**

import boto3

iam\_con\_re=boto3.resource(service\_name="iam",region\_name="us-east-1")

=====================================================================

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2 = aws\_mag\_con.resource('ec2')

---------------------------------------------------------

# **Section 2: Boto3 Documentation help to implement Python boto3 Scripts**

## **12. How to implement Python boto3 Scripts using client object ?**

### **help\_on\_client\_object.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

#iam,ec2 and s3

iam\_con\_cli=aws\_mag\_con.client(service\_name="iam",region\_name="us-east-1")

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name="us-east-1")

s3\_con\_cli=aws\_mag\_con.client(service\_name="s3",region\_name="us-east-1")

'''

#List all iam users using client object

response=iam\_con\_cli.list\_users()

for each\_item in response['Users']:

print(each\_item['UserName'])

'''

'''

#List all ec2 instaces ids

response=ec2\_con\_cli.describe\_instances()

for each\_item in response['Reservations']:

for each\_instance in each\_item['Instances']:

print(each\_instance['InstanceId'])

'''

#List all s3 buckets

response=s3\_con\_cli.list\_buckets()

for each\_item in response['Buckets']:

print(each\_item['Name'])

#print(each\_item.get('Name'))

## **13. How to implement Python boto3 Scripts using resource object ?**

### **help\_on\_resource\_object.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

iam\_con\_re=aws\_mag\_con.resource(service\_name="iam",region\_name="us-east-1")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

s3\_con\_re=aws\_mag\_con.resource(service\_name="s3",region\_name="us-east-1")

'''

#List all iam users

for each\_item in iam\_con\_re.users.all():

print(each\_item.user\_name)

'''

for each\_item in s3\_con\_re.buckets.limit(10):

print(each\_item.name)

# **Section 3: Working with STS**

## **14. Get AWS Account Id**

### **get\_aws\_account\_id.py**

import boto3

aws\_mag\_con\_root=boto3.session.Session(profile\_name="root")

sts\_con\_cli=aws\_mag\_con\_root.client(service\_name="sts",region\_name="us-east-1")

response=sts\_con\_cli.get\_caller\_identity()

print(response.get('Account'))

aws\_mag\_con\_ec2\_dev=boto3.session.Session(profile\_name="ec2\_developer")

sts\_con\_cli=aws\_mag\_con\_ec2\_dev.client(service\_name="sts",region\_name="us-east-1")

response=sts\_con\_cli.get\_caller\_identity()

print(response['Account'])

# **Section 4: EC2-PART-1: Working with EC2 Services**

## **15. Simple Python Boto3 Scripts on EC2 Service (Like ec2 instances, volumes…)**

### **listing\_some\_of\_the\_ec2\_services\_using\_client.py**

import boto3

from pprint import pprint

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name="us-east-1")

'''

response=ec2\_con\_cli.describe\_instances()['Reservations']

for each\_item in response:

for each in each\_item['Instances']:

print("=============================")

print("The Image Id is: {}\nThe Instance Id Is: {}\nThe Instance Launch Time is: {}".format(each['ImageId'],each['InstanceId'],each['LaunchTime'].strftime("%Y-%m-%d")))

'''

response=ec2\_con\_cli.describe\_volumes()['Volumes']

for each\_item in response:

print("=======================")

print("The volume id is: {}\nThe AvailabilityZone is: {}\nThe VolumeType is: {}".format(each\_item['VolumeId'],each\_item['AvailabilityZone'],each\_item['VolumeType']))

### **listing\_some\_of\_the\_ec2\_services\_using\_resource\_object.py**

import boto3

from pprint import pprint

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

'''

for each\_instance in ec2\_con\_re.instances.all():

#print(dir(each\_instance))

print("The Image Id is: {}\nThe Instance Id Is: {}\nThe Instance Launch Time is: {}".format(each\_instance.image\_id,each\_instance.instance\_id,each\_instance.launch\_time.strftime("%Y-%m-%d")))

print("-------------------")

'''

for each\_volume in ec2\_con\_re.volumes.all():

#print(dir(each\_volume))

print("The volume id is: {}\nThe AvailabilityZone is: {}\nThe VolumeType is: {}".format(each\_volume.volume\_id,each\_volume.availability\_zone,each\_volume.volume\_type))

print("-----------------------------")

## **16. Menu Driven script to perform different status actions on EC2 Instance**

### **menu\_driven\_script\_on\_ec2\_actions\_using\_resource.py**

import boto3

import sys

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name="us-east-1")

while True:

print("This script performs the following actions on ec2 instance")

print("""

1. start

2. stop

3. terminate

4. Exit""")

opt=int(input("Enter your option: "))

if opt==1:

instance\_id=input('Enter your EC2 Instance Id: ')

my\_req\_instance\_object=ec2\_con\_re.Instance(instance\_id)

#print(dir(my\_req\_instance\_object))

print("Starting ec2 instance.....")

my\_req\_instance\_object.start()

elif opt==2:

instance\_id=input('Enter your EC2 Instance Id: ')

my\_req\_instance\_object=ec2\_con\_re.Instance(instance\_id)

print("Stopping ec2 instance.....")

my\_req\_instance\_object.stop()

elif opt==3:

instance\_id=input('Enter your EC2 Instance Id: ')

my\_req\_instance\_object=ec2\_con\_re.Instance(instance\_id)

print("Terminating ec2 instance.....")

my\_req\_instance\_object.terminate()

elif opt==4:

print("Thank you for using this script")

sys.exit()

else:

print("Your option is invalid. Please try once again")

### **menu\_driven\_script\_on\_ec2\_actions\_using\_client.py**

import boto3

import sys

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name="us-east-1")

while True:

print("This script performs the following actions on ec2 instance")

print("""

1. start

2. stop

3. terminate

4. Exit""")

opt=int(input("Enter your option: "))

if opt==1:

instance\_id=input('Enter your EC2 Instance Id: ')

#print(dir(my\_req\_instance\_object))

print("Starting ec2 instance.....")

ec2\_con\_cli.start\_instances(InstanceIds=[instance\_id])

elif opt==2:

instance\_id=input('Enter your EC2 Instance Id: ')

print("Stopping ec2 instance.....")

ec2\_con\_cli.stop\_instances(InstanceIds=[instance\_id])

elif opt==3:

instance\_id=input('Enter your EC2 Instance Id: ')

print("Terminating ec2 instance.....")

ec2\_con\_cli.terminate\_instances(InstanceIds=[instance\_id])

elif opt==4:

print("Thank you for using this script")

sys.exit()

else:

print("Your option is invalid. Please try once again")

# **Section 5 EC2 Waiters**

## **17. What is a Waiter ? How to use waiters EC2 Instance ?**

### **introduction\_to\_waiter.py**

import boto3

import time

aws\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_re=aws\_con.resource(service\_name="ec2",region\_name="us-east-1")

ec2\_con\_cli=aws\_con.client(service\_name="ec2",region\_name="us-east-1")

'''

while True:

my\_inst\_ob=ec2\_con\_re.Instance("i-002d4110f1199166f")

print("The current status of ec2 is: ",my\_inst\_ob.state['Name'])

if my\_inst\_ob.state['Name']=="running":

break

print("Wating to get running status....")

time.sleep(5)

'''

'''

my\_inst\_ob=ec2\_con\_re.Instance("i-002d4110f1199166f")

print("Starting given instance....")

my\_inst\_ob.start()

my\_inst\_ob.wait\_until\_running() #Resource waiter waits for 200sec(40 checks after every 5 sec)

print("Now your instance is up and running")

'''

'''

print("Starting ec2 instace...")

ec2\_con\_cli.start\_instances(InstanceIds=['i-002d4110f1199166f'])

waiter=ec2\_con\_cli.get\_waiter('instance\_running')

waiter.wait(InstanceIds=['i-002d4110f1199166f']) #40 checks after every 15 sec

print("Now your ec2 instance is up and running")

'''

my\_inst\_ob=ec2\_con\_re.Instance("i-002d4110f1199166f")

print("Starting given instance....")

my\_inst\_ob.start()

waiter=ec2\_con\_cli.get\_waiter('instance\_running')

waiter.wait(InstanceIds=['i-002d4110f1199166f'])

print("Now your ec2 instace is up and running")

# **Section 6 Usage of Meta object**

## **18. Usage of meta object**

### **list\_all\_regions\_for\_ec2.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2")

for each\_item in ec2\_con\_re.meta.client.describe\_regions()['Regions']:

print(each\_item['RegionName'])

**Note**: By using like above, we can enter client from resource object.

# **Section 7: EC2 Collections**

## **19. Boto3 Collections Concepts**

### **introduction\_to\_collections.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

f1={"Name": "instance-state-name", "Values":['running','stopped']}

f2={"Name":"instance-type","Values":['t2.micro']}

for each in ec2\_con\_re.instances.filter(Filters=[f1,f2]):

print(each)

## **20. Boto3 Collection Concept to start, stop, terminate, monitor all instances once**

### **start\_stop\_terminate\_all\_instances\_at\_once.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name="us-east-1")

'''

all\_instances\_ids=[]

for each\_in in ec2\_con\_re.instances.all():

all\_instances\_ids.append(each\_in.id)

#print(dir(ec2\_con\_re.instances))

waiter=ec2\_con\_cli.get\_waiter('instance\_running')

print("Starting all instances ......")

ec2\_con\_re.instances.start()

waiter.wait(InstanceIds=all\_instances\_ids)

print("your all instaces are up and running")

'''

'''

np\_sers\_ids=[]

f1={"Name": "tag:Name", "Values":['Non\_Prod']}

for each\_in in ec2\_con\_re.instances.filter(Filters=[f1]):

np\_sers\_ids.append(each\_in.id)

print(np\_sers\_ids)

print("----------------------------")

'''

np\_sers\_ids=[]

f1={"Name": "tag:Name", "Values":['Non\_Prod']}

for each\_item in ec2\_con\_cli.describe\_instances(Filters=[f1])['Reservations']:

for each\_in in each\_item['Instances']:

np\_sers\_ids.append(each\_in['InstanceId'])

print(np\_sers\_ids)

print("Starting intances with ids of : ",np\_sers\_ids)

ec2\_con\_cli.start\_instances(InstanceIds=np\_sers\_ids)

waiter=ec2\_con\_cli.get\_waiter('instance\_running')

waiter.wait(InstanceIds=np\_sers\_ids)

print("Your np instances are up and running....")

# **Section 8: EC2-PART-2: Practice with EC2 Services**

## **21. Delete Unused and Untagged EBS Volumes**

### **delete\_unused\_untagged\_volumes\_using\_resource.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

'''

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name='us-east-1')

f\_ebs\_unused={"Name":"status","Values":["available"]}

for each\_volume in ec2\_con\_re.volumes.filter(Filters=[f\_ebs\_unused]):

if not each\_volume.tags:

print(each\_volume.id, each\_volume.state,each\_volume.tags)

print("Deleting unused and untagged volumes.....")

each\_volume.delete()

print("Delted all unused unatageed volumes.")

'''

ec2\_con\_cli=aws\_mag\_con.client(service\_name="ec2",region\_name='us-east-1')

for each\_item in ec2\_con\_cli.describe\_volumes()['Volumes']:

if not "Tags" in each\_item and each\_item['State']=='available':

print('Deleting ',each\_item['VolumeId'])

ec2\_con\_cli.delete\_volume(VolumeId=each\_item['VolumeId'])

print("Deleted all unused and untagged volumes.")

## **22. EC2 Instances Inventory**

### **EC2\_Instance\_Discovery\_Script\_using\_resource.py**

import boto3

import csv

aws\_mag\_con=boto3.session.Session(profile\_name="ec2\_developer")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-east-1")

cnt=1

csv\_ob=open("inventory\_info.csv","w",newline='')

csv\_w=csv.writer(csv\_ob)

csv\_w.writerow(["S\_NO","Instance\_Id",'Instance\_Type','Architecture','LaunchTime','Privat\_Ip'])

for each in ec2\_con\_re.instances.all():

print(cnt,each,each.instance\_id,each.instance\_type,each.architecture,each.launch\_time.strftime("%Y-%m-%d"),each.private\_ip\_address)

csv\_w.writerow([cnt,each.instance\_id,each.instance\_type,each.architecture,each.launch\_time.strftime("%Y-%m-%d"),each.private\_ip\_address])

cnt+=1

csv\_ob.close()

## **23. List all Available Snapshots**

### **list\_all\_snaps.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-west-1")

sts\_con\_cli=aws\_mag\_con.client(service\_name="sts",region\_name="us-east-1")

response=sts\_con\_cli.get\_caller\_identity()

my\_own\_id=response.get('Account')

for each\_snap in ec2\_con\_re.snapshots.filter(OwnerIds=[my\_own\_id]):

print(each\_snap)

## **24. List snapshots based on size**

### **list\_all\_snaps\_based\_on\_size.py**

import boto3

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-west-1")

sts\_con\_cli=aws\_mag\_con.client(service\_name="sts",region\_name="us-east-1")

response=sts\_con\_cli.get\_caller\_identity()

my\_own\_id=response.get('Account')

f\_size={"Name":"volume-size","Values":['10']}

for each\_snap in ec2\_con\_re.snapshots.filter(OwnerIds=[my\_own\_id],Filters=[f\_size]):

print(each\_snap)

## **25. List snapshots based on start time**

### **list\_all\_snaps\_based\_on\_start\_time.py**

import boto3

import datetime

aws\_mag\_con=boto3.session.Session(profile\_name="root")

ec2\_con\_re=aws\_mag\_con.resource(service\_name="ec2",region\_name="us-west-1")

sts\_con\_cli=aws\_mag\_con.client(service\_name="sts",region\_name="us-east-1")

my\_own\_id=sts\_con\_cli.get\_caller\_identity().get('Account')

today=datetime.datetime.now()

start\_time=str(datetime.datetime(today.year,today.month,today.day,4,15,44))

print start\_time

print "Below is using resource object"

for each\_snap in ec2\_con\_re.snapshots.filter(OwnerIds=[my\_own\_id]):

if each\_snap.start\_time.strftime("%Y-%m-%d %H:%M:%S")==start\_time:

print each\_snap.id,each\_snap.start\_time.strftime("%Y-%m-%d %H:%M:%S")

# **Section 9: IAM Roles to Execute python boto3 scripts**

## **26. Executing Python boto3 script from ec2 instance which is attached with IAM Role**

### **Executing\_Python\_boto3\_script\_using\_IAM\_roles.py**

#Use default session while working with IAM Roles

#Create one role "BOTO3\_EX\_FOR\_EC2\_SERVICE" with policy "AmazonEC2FullAccess"

import boto3

ec2\_con=boto3.resource(service\_name="ec2",region\_name="us-east-1")

for each\_instance in ec2\_con.instances.all():

print(each\_instance.id,each\_instance.state)

# **Section 10: Exception Handling with boto3 and botocore modules**

## **27. simple Exception handling**

### **simple\_error\_handling\_with\_boto3.py**

import sys

'''

try:

import boto3

except Exception as e:

print(e)

'''

try:

import boto3

import botocore

except ModuleNotFoundError:

print("Boto3 is not installed. Please intall boto3 and try againg")

sys.exit(1)

except Exception as e:

print(e)

sys.exit(2)

try:

aws\_mag\_con=boto3.session.Session(profile\_name="root")

except botocore.exceptions.ProfileNotFound:

print("root profile is not configured on your .aws credential file. Use other profile or please configure root profile")

sys.exit(3)

except Exception as e:

print(e)

sys.exit(4)

try:

iam\_con\_re=aws\_mag\_con.resource(service\_name="iam")

for each\_user in iam\_con\_re.users.all():

print(each\_user)

except botocore.exceptions.ClientError as e:

if e.response['Error']['Code'] == "AccessDenied":

print("Your profile is not having access to work with IAM Users")

else:

print(e.response['Error']['Code'])

sys.exit(5)

except Exception as e:

print(e)

sys.exit(6)

# **Section 11: Working with IAM Users**

## **28. Get IAM User Details and Get All IAM Users Details**

### **get\_iam\_user\_details\_and\_all\_iam\_users\_details.py**

import boto3

import datetime

session=boto3.session.Session(profile\_name="root")

iam\_con\_re=session.resource(service\_name="iam")

#Get details of any iam user

'''

iam\_user\_ob=iam\_con\_re.User("s3\_developer")

print(iam\_user\_ob.user\_name,iam\_user\_ob.user\_id,iam\_user\_ob.arn,iam\_user\_ob.create\_date.strftime("%Y-%m-%d"))

'''

for iam\_user\_ob in iam\_con\_re.users.all():

print(iam\_user\_ob)

print(iam\_user\_ob.user\_name,iam\_user\_ob.user\_id,iam\_user\_ob.arn,iam\_user\_ob.create\_date.strftime("%Y-%m-%d"))

## **30. Get All Groups Info**

## **31. Random Password Generator using Python**

### **Random\_Password\_Generator\_using\_Python.py**

#The random password generator is useful while creating IAM user password.

from random import choice

len\_of\_password=8

valid\_chars\_for\_password="abcdjkdwdjwdkk8378040\*$$^\*66t6(';'.,``~@#%\*)\_&%"

password=[]

'''

for each\_char in range(len\_of\_password):

password.append(choice(valid\_chars\_for\_password))

random\_pass="".join(password)

print(random\_pass)

'''

random\_pass="".join(choice(valid\_chars\_for\_password) for each\_char in range(len\_of\_password))

print(random\_pass)

## **32. Create an IAM User with Console Login access Using boto3 of Python**

### **create\_an\_iam\_user\_console\_login\_access.py**

import boto3

from random import choice

import sys

def get\_iam\_client\_object():

session=boto3.session.Session(profile\_name="dev\_root")

iam\_client=session.client(service\_name="iam",region\_name="us-east-1")

return iam\_client

def get\_random\_password():

len\_of\_password=8

valid\_chars\_for\_password="abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&\*()?"

return "".join(choice(valid\_chars\_for\_password) for each\_char in range(len\_of\_password))

def main():

iam\_client=get\_iam\_client\_object()

Iam\_user\_name="dowithpython@gmail.com"

passwrd=get\_random\_password()

PolicyArn="arn:aws:iam::aws:policy/AdministratorAccess"

try:

iam\_client.create\_user(UserName=Iam\_user\_name)

except Exception as e:

if e.response['Error']['Code']=="EntityAlreadyExists":

print "Already Iam User with {} is exist".format(Iam\_user\_name)

sys.exit(0)

else:

print "Please verify the following error and retry"

print e

sys.exit(0)

iam\_client.create\_login\_profile(UserName=Iam\_user\_name,Password=passwrd,PasswordResetRequired=False)

iam\_client.attach\_user\_policy(UserName=Iam\_user\_name,PolicyArn=PolicyArn)

print "IAM User Name={} and Password={}".format(Iam\_user\_name,passwrd)

return None

if \_\_name\_\_=="\_\_main\_\_":

main()

## **33. Create an IAM User with programatic Access Keys**

### **Create\_an\_IAM\_User\_with\_programatic\_Access\_Keys.py**

import boto3

from random import choice

import sys

def get\_iam\_client\_object():

session=boto3.session.Session(profile\_name="dev\_root")

iam\_client=session.client(service\_name="iam",region\_name="us-east-1")

return iam\_client

def main():

iam\_client=get\_iam\_client\_object()

Iam\_user\_name="dowithpython@gmail.com"

PolicyArn="arn:aws:iam::aws:policy/AdministratorAccess"

try:

iam\_client.create\_user(UserName=Iam\_user\_name)

except Exception as e:

if e.response['Error']['Code']=="EntityAlreadyExists":

print "Already Iam User with {} is exist".format(Iam\_user\_name)

sys.exit(0)

else:

print "Please verify the following error and retry"

print e

sys.exit(0)

response = iam\_client.create\_access\_key(UserName=Iam\_user\_name)

print "IAM User Name={}".format(Iam\_user\_name)

print "AccessKeyId={}\nSecretAccessKey={}".format(response['AccessKey']['AccessKeyId'],response['AccessKey']['SecretAccessKey'])

iam\_client.attach\_user\_policy(UserName=Iam\_user\_name,PolicyArn=PolicyArn)

return None

if \_\_name\_\_=="\_\_main\_\_":

main()

## **34. Create an IAM User with Programatic and AWS Managment Console Login Access**

### **Create\_an\_IAM\_User\_with\_Programatic\_and\_AWS\_Managment\_Console\_Login\_Access.py**

#It is an assignment: you can complete it by adding previous two lectures.

## **35. Create mutliple IAM Users using python boto3**

### **Create\_mutliple\_IAM\_Users\_using\_python\_boto3.py**

'''

Take list of iam users in a csv file like

S\_NO, IAM\_User\_Name,Programatic\_Access,Console\_Access,PolicyARN

1,XYZ, Yes,No,arn:aws:iam::aws:policy/AdministratorAccess

2.pqr,Yes,Yes,arn:aws:iam::aws:policy/AdministratorAccess

3.abc,No,Yes,arn:aws:iam::aws:policy/AmazonAPIGatewayInvokeFullAccess

'''

import boto3,sys

from pprint import pprint

while True:

session=boto3.session.Session(profile\_name="dev\_root")

iam\_re=session.resource(service\_name="iam")

for each in range(701,1100):

try:

iam\_re.create\_user(UserName="ixasisiidemo"+str(each))

if each==509:

sys.exit()

except:

continue

## **36. IAM Users Inventory Script**

### **IAM\_Users\_Inventory\_Script.py**

'''

Write a Python boto3 script to export IAM User Details into a csv file.

CSV file content is like IAM User Name, User Id, User ARN, User Creation Date,

Attached Policies and Groups associated for IAM Users

'''

import boto3

session=boto3.session.Session(profile\_name="dev\_root")

'''

iam\_re=session.resource(service\_name="iam",region\_name="us-east-1")

for each\_user in iam\_re.users.all():

print "UserName={} UserId={} User ARN={} User Creation Date={}".format(each\_user.user\_name,each\_user.user\_id,each\_user.arn,each\_user.create\_date)

'''

iam\_cli=session.client(service\_name="iam",region\_name='us-east-1')

for each\_user in iam\_cli.list\_users()['Users']:

print each\_user['UserName'],each\_user['Arn']

# **Section 12 Lambda-PART-1 AWS Lambda Functions**

## **37. Introduction to AWS Lambda Functions**

AWS Lambda is a serverless computing platform that allows engineers to create a small function

, configure the function in AWS console, and have the code executed without the need to provision servers- paying only for the resources used during execution

Simply, it is like an editor(vim, Pycharm, Sublimetext, atom) with some extra features.

It supports to run different languages like- python, go, java, Node.js etc.

It is installed or running on Amazon Linux server and we can access /tmp using Lambda function

* A Lambda function has few requirements.
* The first requirement you need to satisfy is to provide a handler.
* The handler is the entry point for the Lambda
* A Lambda function accepts JSON-formatted input and usually return the same.
* The second requirement is that you will need to specify the runtime environment for the Lambda. The runtime will usually correlate directly with the language you selected to write your function.
* The final requirement is a trigger
* Manual trigger or run by us
* You can configure a Lambda invocation in response to an event, such as a new file uploadded to S3, a change in the DynamoDB table, or a similar AWS event. You can also configure Lambda to respond to requests to AWS API Gateway or based on a timer triggered by AWS Cloudwatch

**How AWS Lambda function executes the code for AWS services**

Two ways:

* Use Programmatic Access keys
* Create a AWS IAM Role and attach the role to AWS Lambda

**DemoPythonLambdaFunction**

import boto3

def lambda\_handler(event, context):

s3\_con=boto3.resource("s3","us-east-1")

for each\_bu in s3\_con.buckets.all():

print each\_bu.name

## **38. Automating the start and stop EC2 Instances for Test Environment**

**Everyday**

Start EC2 Instances at 8am Mon-Fri

Stop EC2 Instances at 5pm Mon-Fri

**Step-1: Create a Role for Lambda Function**

Role name: LambdaRoleToWorkWithEC2

Role description: Allows Lambda function to call AWS services on your behalf.

Policies: AWSEC2FullAccess

**Step-2: Write a Lambda Function using boto3 of python**

AutoStartTestEC2Instances\_8am

AutoStopTestEC2Instances\_5pm

lambda\_function

import boto3

def lambda\_handler(event,context):

ec2\_con\_re=boto3.resource(service\_name="ec2",region\_name="us-east-1")

test\_env\_filter={"Name":"tag:Env","Values":["Test"]}

for each\_in in ec2\_con\_re.instances.filter(Filters=[test\_env\_filter]):

each\_in.start()

return "Success"

**Step-3: Schedule the job**

Goto CloudWatch -> Rules -> Create Rule-> Schedule -> Cron Expression (0 8 ? \* MON-FRI \*)

Add Target -> Select the Lambda function "AutoStartTestEC2Instances\_8am"

Rule Definition -> Name="AutoStartTestEC2Instances\_8am" -> Create Rule

**AutoStopTestEC2Instances\_5pm**

lambda\_function

import boto3

def lambda\_handler(event,context):

ec2\_con\_re=boto3.resource(service\_name="ec2",region\_name="us-east-1")

test\_env\_filter={"Name":"tag:Env","Values":["Test"]}

for each\_in in ec2\_con\_re.instances.filter(Filters=[test\_env\_filter]):

each\_in.stop()

return "Success"

## **39. Automatic Mail Alert When Instance state is reached to Stopped**

**Mail Alert: get AWS EC2 instances status when it stopped**

**Step-1**

Lambda Function --> IAM Roles(EC2,IAM)

**Step-2**

Create a lambda with boto3 of python

MailAlertForProdServers

lambda\_function

import json

import boto3

def lambda\_handler(event,context):

#Thecommented lines are not required if scheduled through cloudwatch

#ec2\_con=boto3.resource("ec2","us-east-1")

sns\_client=boto3.client("sns","us-east-1")

#my\_ins=ec2\_con.Instance("i-020029a18ce55706")

#print my\_ins.state['Name']

#The SNS topic is configured with some email address, copy the arn from that topic and use in below

#sns\_client.publish(TargetArn="arn:aws:sns:us-east-1:967636435446:Status\_of\_Ec2",Message=my\_ins.state['Name'])

sns\_client.publish(TargetArn="arn:aws:sns:us-east-1:967636435446:Status\_of\_Ec2",Message="Now instance is in stopped Status")

#return "Success"

**Step-3**

Configure Cloudwatch event

Create Rule -> Event Pattern -> Service Name=EC2 , Event Type=EC2 Instance State-change Notification ->

Specific state="Stopped"

Specific instance ids=i-020029a18ce55706

Add the target to lambda function

Create the rule name as "MailAlertForProdServers"

**Scenario:** You are having one Security Group and for that security group incase if the inbound got changed other than your required port

at that time automatically I want to send mail alert. This is used for compliance purpose.

## **40. Automate Snapshots for EBS Volumes using Lambda and Cloudwatch**

**Automated EBS Snapshots using AWS Lambda & Cloudwatch**

**Steps:**

Write a code to list all EBS Volumes based on requirement

Extend Script to take snapshots

Finally we will implement code to take snapshpts with lambda using cloudwatch trigger or we will schedule a job with lambda and cloudwatch.

**Steps-1**

Write a code to list all EBS Volumes based on requirement

#Note: This code works for a single specified region

import boto3

import pprint

session=boto3.session.Session(profile\_name="dev\_root")

ec2\_client=session.client(service\_name="ec2",region\_name="us-east-1")

#pprint(ec2\_client.describe\_volumes()['Volumes'])

list\_of\_volids=[]

f\_prod\_bkp={'Name':'tag:Prod','Values':['backup','Backup']}

#The below commented code works perfectly, if no. of volumes are less than 50. If more than 50 volumes, we have to use paginators

#for each\_vol in ec2\_client.describe\_volumes()['Volumes']:

# list\_of\_volids.append(each\_vol['volumeId'])

paginator=ec2\_client.get\_paginator('describe\_volumes')

for each\_page in paginator.paginate(Filters=[f\_prod\_bkp]):

#pprint(each\_page['Volumes'])

for each\_vol in each\_page['Volumes']:

list\_of\_volids.append(each\_vol['VolumeId'])

print "The list of volids are: ".list\_of\_volids

snapids=[]

for each\_volid in list\_of\_volids:

print "Taking snap of {}".format(each\_volid)

res=ec2\_client.create\_snapshot(

Description="Taking snap with Lambda and CW",

VolumeId=each\_volid,

TagSpecifications=[

{

'ResourceType':'snapshot',

'Tags': [

{

'Key': 'Delete-on',

'Value': '90'

}

]

}

]

)

#print(res.get('SnapshotId'))

snapids.append(res.get('SnapshotId'))

print "The snap ids are:",snapids

waiter=ec2\_client.get\_waiter('snapshot\_completed')

waiter.wait(SnapshotIds=snapids)

print "Successfully completed snaps for the volumes of {}".format(list\_of\_volids)

return "Success"

**Steps-2**

Lambda function name = AutomateSnapsForEBSVolumes

#Comment andupdate as below line from above script and place the code inside def lambda\_handler(event,context):

#session=boto3.session.Session(profile\_name="dev\_root")

ec2\_client=boto3.client(service\_name="ec2",region\_name="us-east-1")

...

...

return None

**Step-3**

Schedule a job using Cloudwatch event for automate the snapshots

## **41. Automate Snapshots for EBS Volumes using resource object**

#!/bin/python

import boto3

from pprint import pprint

session=boto3.session.Session(profile\_name="dev\_root")

ec2\_re=session.resource(service\_name="ec2",region\_name="us-east-1")

vol\_ids=[]

#colleting volume Ids

for each\_vol in ec2\_re.volumes.filter(Filters=[]):

#print each\_vol.id

vol\_ids.append(each\_vol.id)

print 'All volume ids are: ',vol\_ids

#Creating snapshots for volumes one by one

snap\_ids=[]

for each\_vo\_id in vol\_ids:

response= ec2\_re.create\_snapshot(

Description='Snap with Lambda',

VolumeId=each\_vo\_id,

TagSpecifications=[

{

'ResourceType': 'snapshot',

'Tags': [

{

'Key': 'Delete-on',

'Value':'90'

}

]

}

]

)

snap\_ids.append(response.id)

print snap\_ids

#Creating waiter using client

ec2\_cli=session.client(service\_name="ec2",region\_name="us-east-1")

waiter = ec2\_cli.get\_waiter('snapshot\_completed')

waiter.wait(SnapshotIds=snap\_ids)

## **42. Automate Snapshots of EBS Volumes for all regions**

import boto3

session=boto3.session.Session(profile\_name="dev\_root")

ec2\_client=session.client(service\_name="ec2",region\_name="us-east-1")

all\_regions=[]

for each\_region in ec2\_client.describe\_regions()['Regions']:

#print(each\_region.get('RegionName'))

all\_regions.append(each\_region.get('RegionName'))

for each\_region in all\_regions:

print("Working on {}".format(each\_region))

ec2\_client=session.client(service\_name="ec2",region\_name=each\_region)

list\_of\_volids=[]

f\_prod\_bkp={'Name':'tag:Prod','Values':['backup','Backup']}

paginator=ec2\_client.get\_paginator('describe\_volumes')

for each\_page in paginator.paginate(Filters=[f\_prod\_bkp]):

#pprint(each\_page['Volumes'])

for each\_vol in each\_page['Volumes']:

list\_of\_volids.append(each\_vol['VolumeId'])

print("The list of volids are: ",list\_of\_volids)

if bool(list\_of\_volids)==False:

continue

snapids=[]

for each\_volid in list\_of\_volids:

print("Taking snap of {}".format(each\_volid))

res=ec2\_client.create\_snapshot(

Description="Taking snap with Lambda and CW",

VolumeId=each\_volid,

TagSpecifications=[

{

'ResourceType':'snapshot',

'Tags': [

{

'Key': 'Delete-on',

'Value': '90'

}

]

}

]

)

#print(res.get('SnapshotId'))

snapids.append(res.get('SnapshotId'))

print("The snap ids are:",snapids)

waiter=ec2\_client.get\_waiter('snapshot\_completed')

waiter.wait(SnapshotIds=snapids)

print("Successfully completed snaps for the volumes of {}".format(list\_of\_volids))

#Copy the above code into lambda handler to automate

#Comment and update as below line from above script and place the code inside def lambda\_handler(event,context):

#session=boto3.session.Session(profile\_name="dev\_root")

ec2\_client=boto3.client(service\_name="ec2",region\_name="us-east-1")

...

...

return None

#The only challenge in automating this is based on the number of volumes, we have to calculate the time taken for taking snapshots and set the timeout value in Lambda function.

## **43. Automate Copy of EBS Snapshots between Regions**

**Why Copy?**

Why would you want to copy an EBS Snapshot from one AWS Region to another?

**Use Cases:**

**Geographic Expansion:** You want to be able to launch your application in a new Region.

**Migration:** You want to be able to migrate your application from one Region to another.

**Disaster Recovery:** You want to back up your data and your log files across different geographical locations at regular intervals to minimize data loss and recovery time.

**Our requirement is for Disaster recovery**

import os,sys

try:

import boto3

print("Imported boto3 successfully")

except Exception as e:

print(e)

sys.exit(1)

source\_region="us-east-1"

dest\_region="us-east-2"

session=boto3.session.Session(profile\_name="dev\_root")

ec2\_source\_client=session.client(service\_name="ec2",region\_name=source\_region)

sts\_client=session.client(service\_name="sts",region\_name=source\_region)

account\_id=sts\_client.get\_caller\_identity().get('Account')

bkp\_snap=[]

f\_bkp={'Name':'tag:backup','Values':['yes']}

for each\_snap in ec2\_source\_client.describe\_snapshots(OwnerIds=[account\_id],Filters=[f\_bkp]).get('Snapshots'):

#print(each\_snap.get('SnapshotId'))

bkp\_snap.append(each\_snap.get('SnapshotId'))

ec2\_dest\_client=session.client(service\_name="ec2",region\_name=dest\_region)

for each\_source\_snapid in bkp\_snap:

print("Taking backup for id of {} into a {}".format(each\_source\_snapid,dest\_region))

ec2\_dest\_client.copy\_snapshot(

Description="Disaster Recovery",

SourceRegion=source\_region,

SourceSnapshotId=each\_source\_snapid

)

print("EBS Snapshot copy to destination region is completed")

print("Modifying tags for the snapshots for which backup is completed")

for each\_source\_snapid in bkp\_snap:

print("Deleting old tags for {}".format(each\_source\_snapid))

ec2\_source\_client.delete\_tags(

Resources=[each\_source\_snapid],

Tags=[

{

'Key':'backup',

'Value':'yes'

}

]

)

print("Creating new tags for {}".format(each\_source\_snapid))

ec2\_source\_client.create\_tags(

Resources=[each\_source\_snapid],

Tags=[

{

'Key':'backup',

'Value':'completed'

}

]

)

#Copy the above code into lambda handler to automate

#Comment and update as below line from above script and place the code inside def lambda\_handler(event,context):

#session=boto3.session.Session(profile\_name="dev\_root")

ec2\_client=boto3.client(service\_name="ec2",region\_name="us-east-1")

...

...

return None

# **Section 13: Paginators of boto3**

## **44. boto3 client paginators**

**Paginators:**

* Paginators play a role when we use boto3 to query AWS resource.
* Like get all EC2 instances, IAM users, buckets, objects etc.
* For query, API calls are made to AWS through boto3.
* Generally, each API call will return 50 or 100 results.
* Note: S3 will return upto 1000 results.
* Actually each API call return a page and each page consist of information of 50 or 100 results(Except S3 objects)
* Example: To get all IAM users info using boto3 client method
* Default API Client call will give one page and that page consist of only 100 users info the what about remaining IAM users?
* Collect all pages to get info of all IAM users.
* If you are trying to retrieve more than one "page" of results you will need to use a paginator to issue multiple API requests on your behalf.
* Boto3 provides Paginators to automatically issue multiple API requests to retrieve all the pages(from page all results.)
* Paginators are straightforward to use.
* But not all Boto3 services provide paginator support. For those services you will need to write your own paginator in Python.

**How to use Paginators?**

Step1: Create a paginator

Step2: Paginate through created paginator to get pages one by one

import boto3

'''

session=boto3.session.Session(profile\_name="dev\_root")

iam\_re=session.resource("iam")

cnt=1

for each\_user in iam\_re.users.all():

print cnt,each\_user.user\_name

cnt=cnt+1

cnt=1

iam\_cli=session.client("iam")

for each\_user in iam\_cli.list\_users()['Users']:

print cnt, each\_user['UserName']

cnt=cnt+1

'''

'''

cnt=1

iam\_cli=session.client("iam")

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

for each\_page in paginator.paginate():

for each\_user in each\_page['Users']

print cnt,each\_user['UserName']

cnt=cnt+1

'''

iam\_cli=session.client("ec2")

paginatro=iam\_cli.get\_paginator('describe\_instances')

for each\_page in paginatro.paginate():

print each\_page

Till now below are covered

**resource**

**client**

**waiter**

**paginator**

# **Section 14: Working with S3**

## **45. Concepts of boto3 to work with AWS S3 Service**

* **Session(Custom session, default session)**
* **resource**
* **client**
* **meta**
* **collections(filters)**
* **waiters**
* **paginators**

## **46. List all buckets**

import boto3

custom\_session=boto3.session.Session(profile\_name="dev\_root")

s3\_re=custom\_session.resource(service\_name="s3",region\_name="us-east-1")

print("Using resource object:")

for each\_bucket\_info in s3\_re.buckets.all():

print(each\_bucket\_info)

s3\_cli=custom\_session.client(service\_name="s3",region\_name="us-east-1")

print("Using client object:")

for each\_bucket\_info in s3\_cli.list\_buckets().get('Buckets'):

print(each\_bucket\_info.get('Name'))

## **47. Paginators for s3**

List all objects of a bucket

import boto3

session=boto3.session.Session(profile\_name="dev\_root")

'''

s3\_re=session.resource(service\_name="s3",region\_name="us-east-1")

bucket\_name="dowithpythonapril"

bucket\_object=s3\_re.Bucket(bucket\_name)

cnt=1

for each\_obj in bucket\_object.objects.all():

print(cnt,each\_obj.key)

cnt=cnt+1

'''

'''

s3\_cli=session.client(service\_name="s3",region\_name="us-east-1")

bucket\_name="dowithpythonapril"

cnt+1

for each\_object in s3\_cli.list\_objects(Bucket=bucket\_name)['Contents']:

print(cnt,each\_object['Key'])

cnt=cnt+1

'''

cnt=1

s3\_cli=session.client(service\_name="s3",region\_name="us-east-1")

bucket\_name="dowithpythonapril"

paginator=s3\_cli.get\_paginator('list\_objects')

for each\_page in paginator.paginate(Bucket=bucket\_name):

for each\_object in each\_page['Contents']:

print(cnt,each\_object['Key'])

cnt=cnt+1

# **Section 15: Lambda-PART-2: AWS Automation with boto3 and Lambda Functions**

Automatically Assign Secondary IP of Master to Slave when Master downs

(How to create a server failover solution?)

Lambda function name= AutoAssignSecIPtoSlaveWhenMasterDown

Schedule it through CloudWatch

import json

import boto3

import sys

def lambda\_handler(event,context):

master\_id="i-0aca754c77390c67"

slave\_id="i-0d26g77723j21c345"

secondary\_ip="172.31.90.45"

ec2\_re=boto3.resource("ec2","us-east-1")

primary\_instance=ec2\_re.Instance(master\_id)

if primary\_instance.state['Name']=="Running":

print("Master is running.so no modifications")

else:

secondary\_instance=ec2\_re.Instance(slave\_id)

pnetwork\_interface\_Info=primary\_instance.network\_interfaces\_attribute[0]

snetwork\_interface\_Info=secondary\_instance.network\_interfaces\_attribute[0]

pnw\_interface\_id=pnetwork\_interface\_Info['NetworkInterfaceId']

snw\_interface\_id=snetwork\_interface\_Info['NetworkInterfaceId']

ec2\_cli=boto3.client("ec2","us-east-1")

ec2\_cli.unassign\_private\_ip\_addresses(

NetworkInterfaceId=pnw\_interface\_id,

PrivateIpAddresses=[secondary\_ip]

)

ec2\_cli.assign\_private\_ip\_addresses(

AllowReassignment=True,

NetworkInterfaceId=snw\_interface\_id,

PrivateIpAddresses=[secondary\_ip]

)

return None