IaC AWS python code will automatically construct highly resilient AWS infrastructure for AfterPay demo application using AutoSclaing and Elastic Load Balancing capabilities in AWS.

AWS IaC

Accent on Cloud & Automation

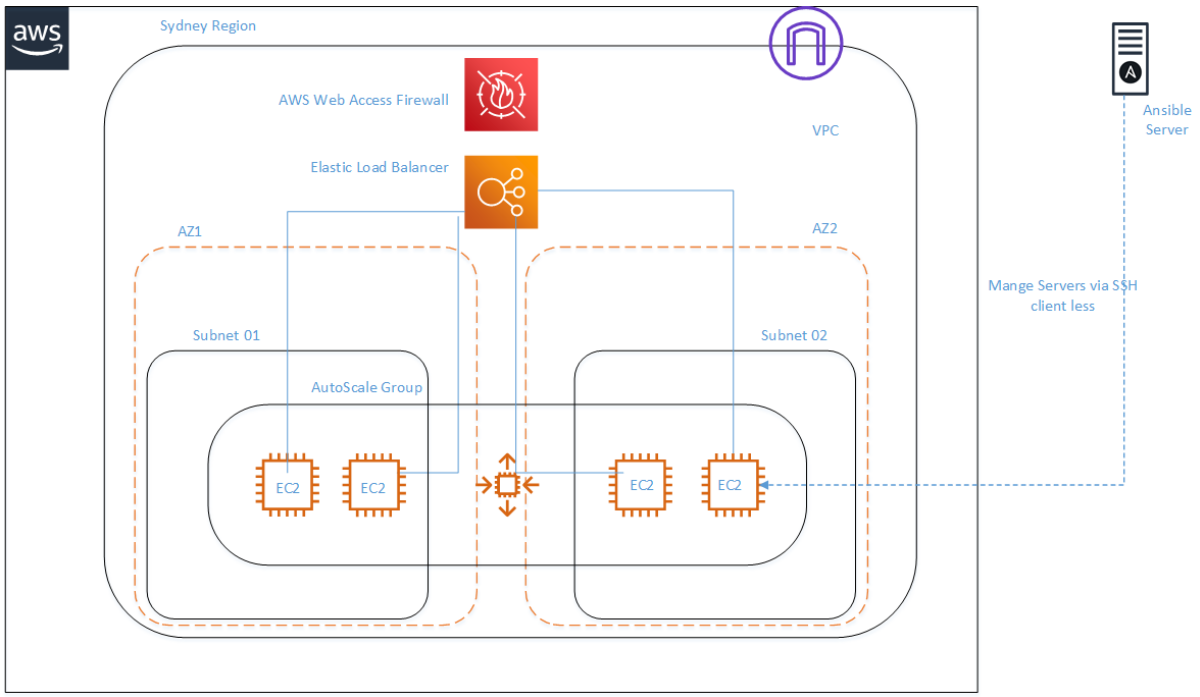
Randy Wijerathne

TABLE OF CONTENTS

1. ABSTRACT

AWS provides programming API (Application Programming Interface) interface and ability code infrastructure using SDK (Software Development Kit). It also provides ability deploy application in a highly resilient (anti-fragile) architecture. Python selected for coding AWS infrastructure resources. AWS provide AWS SDK, boto3 for python programmers. Depicted below is AWS infrastructure deployed for AfterPay demo application. However, existing code deploy only to one AZ (Availability Zone) and does not deploy Ansible Server nor AWS WAF. Furthermore, it is a prerequisite to deploy infrastructure on a clean slate AWS infrastructure with default VPC and subnets, since code being tested on similar AWS environment.

Author acknowledge coding examples and resources at AWS Boto3 documentation and StackOverflow articles used for coding.



2. FUNCTIONAL REQUIREMENTS

Functional Requirement	Reference/Evidence
Linux OS image with below preinstall packages and pre-configurations as per below, • make sure all packages are up to date and all pending security updates are applied against the default OS repositories. • disable IPv6 system wide. • install the following packages: ntp (NTP daemon) (make sure it is started at boot), telnet, mtr, tree. • set the max "open files" limit across all users/processes, soft & hard, to 65535.	Section 4, Amazon Linux AMI used as a base and then install and configure with required packages.

Deployed onto AWS.	Section 1 – 6, AWS used as an IaaS (Infrastructure as a Service).
Configuration management (Ansible/Chef/Puppet) to install & configure server/application.	Section 1 & 3, SSH port is open to connect to EC2 server instances and proposed to use agentless Ansible for ongoing configuration management.
Application available on port 80 (using an appropriate web server).	Section 3 & 4, Flash Development server used for AfterPay application. when code downloaded from GitHub to a EC2 instance, User Data script will update downloaded AfterPay code line app.run() to app.run(host='0.0.0.0', debug=True, port=80)
Server locked-down and secure.	Section 3 & 4, Separate AWS Security Group created with only allowing SSH and HTTP protocol. Also, sudo yum -y update –security applied during customized AMI build process.
Anti-fragility: if the server disappears, how does it automatically recover?	Section 1, 3 and 4, EC2 server instances placed inside an AutoScale group with an ELB (Elastic Load Balancer). AutoScale set to keep minimum two EC2 instance. During one or all EC2 instance failure, AWS wait for cooling down period (set to 30 seconds) and then spin up new EC2 instance using customised AMI and User Data script attached with its Launch configuration.
Code / Documentation layout.	This artifact and comments in code.
Ease of deployment.	Section 4 – 6, Code does not require any arguments to be passed nor request interactive inputs.
Simplicity.	Section 1 - 5

3. Design Decisions

Design	Alternative	Rationale
Amazon Linux AMI is used as base Linux AMI to create a customized AfterPay AMI.	Use specific Linux flavour such as RHEL.	Amazon Linux AMI is light weight and costs less with a t2.micro EC2 instance. Furthermore, it is similar to RHEL/CentOs.
AWS Security Groups open to SSH and HTTP for any IP address.	Lock down to IP address field to specific IP address for SSH. For example, SSH access lock down to Ansible and few bastion hosts.	Operational simplicity. However, it has increased attack surface.
Leverage default VPC and subnets.	Create separate VPC and subnets.	Code simplicity.
AWS infrastructure only deployed on one AZ.	AWS infrastructure deployed on two or more AZs.	Code simplicity.
AutoScale Group with maximum and minimum EC2 instance set to two with Classic ELB.	AutoScale Group with Application ELB with monitoring enabled to scale up with demand.	Code simplicity and Classic ELBs coding support.
AWS WAF (Web Application Firewall) and Ansible not deployed as part of the code.	Deploy AWS WAF and Ansible server.	Infrastructure deployment simplicity. However, it has introduce high risk of application exploitation from an adversary.
AutoScaling Group with aggressive cooling down timers.	AutoScaling Group with long cooling down timers.	This will provide an ability code accessor to terminate EC2 instance and assess application's anti-fragility with less waiting time.

Ansible proposed for ongoing configuration management.	Chef and Puppet.	Chef and Puppet require agents to be installed in EC2 instance, However, Ansible is agentless and only required SSH access.
Flask Development Server was relying on Flask apps code's app.run() on listening port and IP address thus each and every time Flask app downloaded from GitHub and then modified with Sed (Stream Editor – Linux command) to listen to port 80 and any IP address.	n/a	Due to time constraints that was the only successful solution found.
Longer pause timers selected between resource creation.	n/a	This will reduce deployment failures due to use of pending state resources. For example, AMI image availability depends on EC2 instance creation.
AutoScale Launch configuration derived by referring an EC2 instance in AutoScale.	Write separate code for Launch configuration.	Code simplicity.

4. THE CODE

This section provides high level actions executed by Python script,

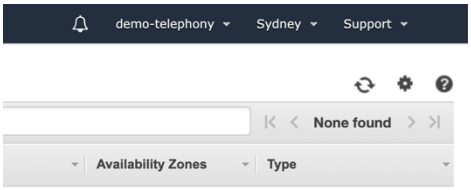
- Find an Amazon Linux AMI ID for an AWS region in interest.
- Create an SSH Key in AWS and then download PEM certificate to local computer.
- Spin up an EC2 instance from Amazon Linux AMI and then update packages, apply security update using YUM, install NTP, telnet, tee, mrt, set max open files and then disable IPv6 using user data attributes on EC2.
- Take a snapshot of above EC2 and create an AMI which will be the customized Linux AMI for AfterPay.
- Then use above customized AMI and spin up EC2 instance as an intermediate EC2 server.
- Install Flask Development server, download AfterPay app from GitHub, change app.run() in downloaded AfterPay application code and then start Flask Development Server by using user data attribute in EC2 instance.
- Create an AWS Security Group which is only to SSH and HTTP.
- Create a Classic Elastic Load Balancer (ELB).
- Create an AutoScale group based on intermediate EC2 server created on previous step.
- Attach Classic Elastic Load Balancer (ELB) created on previous step which will provide front end for load balanced AfterPay application running on EC2 instance.
- Terminate two EC2 instances which were used as intermediate EC2 servers during provisioning.

5. PREPARATION STEPS

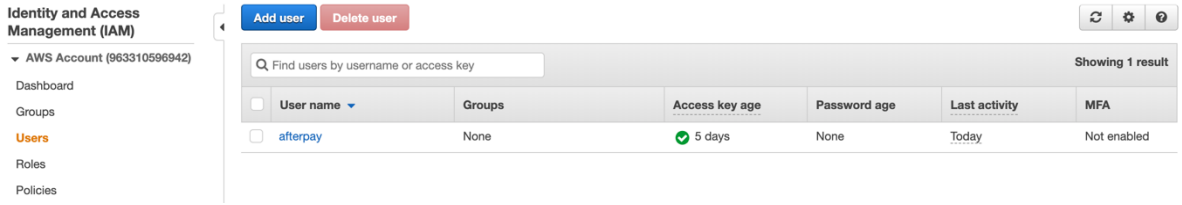
This document is written focusing on setting environment on a MacOS computer. However, preparation steps are not unique to MacOS computer. For a similar environment, run this code in a Python 2.7.10 version and install additional python packages such as boto3, matplotlib, tornado and nose. Furthermore, install AWS CLI to configure access keys, secret keys and default AWS region.

5.1. PREPARATION TASKS ON AWS ACCOUNT

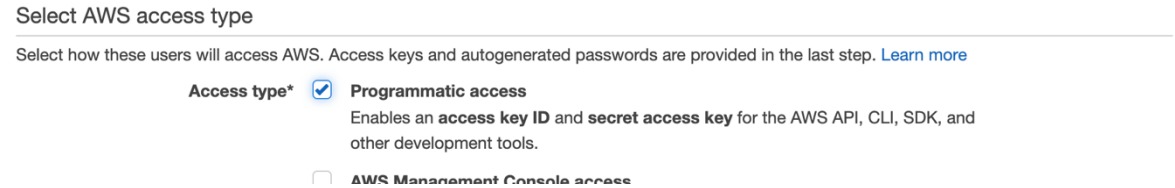
1. Login to AWS management console and change region to Sydney



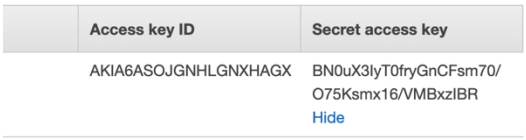
2. Go to IAM and create a new user called AfterPay



3. Select “Programming access”



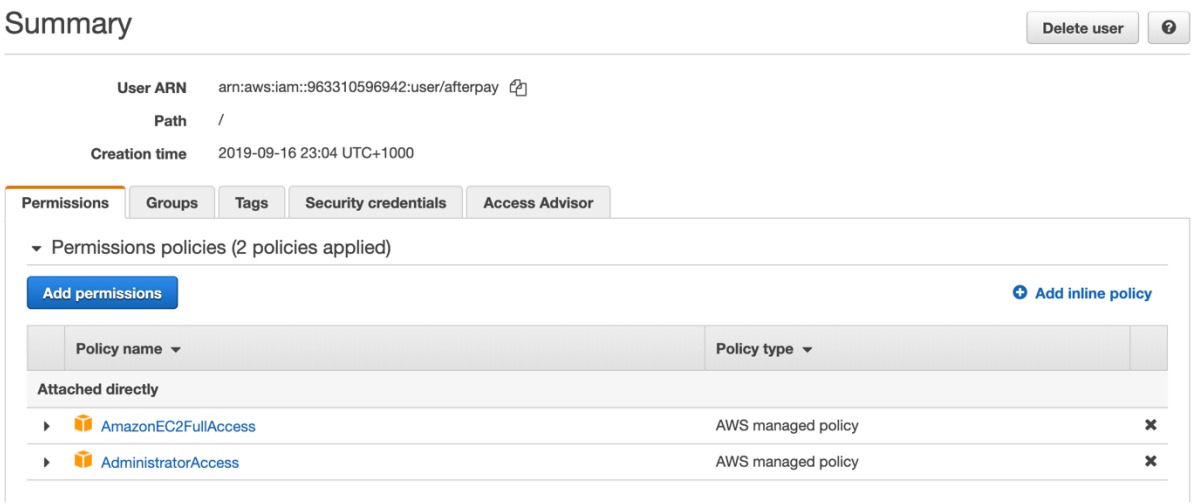
4. Copy Access Key and Secret access Key, which will be required on later stage.



5. Add depicted permissions to AfterPay API user,

AmazonEC2FullAccess

AdministratorAccess



5.2. PREPARATION TASKS ON LOCAL COMPUTER

5.2.1. INSTALL PYTHON AND RELATED PACKAGES

```
brew install python
sudo python -m pip install tornado
sudo python -m pip install nose
sudo python -m pip install boto3
```

5.2.2. INSTALL AWS CLI TOOL, CONFIGURE CREDENTIALS AND SET DEFAULT REGION

```
sudo pip install awscli
aws configure
Enter below,
AWS Access Key ID [****************ASUA]: {Enter previously copied AWS access key}
AWS Secret Access Key [****************OZTD]: {Enter previously copied AWS secret key}
Default region name [us-east-1]: ap-southeast-2
Default output format [None]:
```

6. EXECUTING AWS PYTHON SCRIPT

Download code from GitHub and run the python code. Please note that completing AWS infrastructure provisioning would take ~ 30 minutes.

GitHub code,

https://github.com/phenix888/AfterPayCode/blob/master/AWSIaC.py

To run the code type (Python 2.7),

```
python AWSIaC.py
```

```
2019-09-22 12:41:56,739 Script will use Amazon AMI ID : ami-07cc15c3ba6f8e287 which
will be used for  building base customized AMI for AfterPay
2019-09-22 12:41:57,029 Generated a SSH-key file AfterPaySSHKey.pem whcih can be used
to login to server and saved it on local folder used to run this script.
2019-09-22  12:41:57,030  Creating  a  customized  AMI  for  After  pay  with  below
specifications


*All packages are update to date and all pending security updates are applied against
the default OS repositories.

*disable IPv6 system wide.

*install the following packages: ntp (NTP daemon)(make sure it is started at boot),
telnet, mtr, tree.

*set the max "open files" limit across all users/processes, soft & hard, to 65535.


2019-09-22 12:41:57,031 Creating a customized AMI for AfterPay...
2019-09-22  12:41:58,494  Updatinge  packages,  installing  services  and  configuring
services in progress for customized AMI for AfterPay ...
2019-09-22 12:50:21,583 Customized AMI image build for  AfterPay completed ... AMI
image name is AfterPayAMI
2019-09-22  12:50:22,057  Security  Group  Created  sg-03a486b896203cd64  in  vpc  vpc-
60370107.
2019-09-22  12:54:52,448  Installing  Flask  Development  Server,Download  AfterPay  app
from Git and run server on port 80
2019-09-22 12:54:54,447 Hang on tight mate, It is almost there !
2019-09-22 12:59:54,453 Please use Elastic Load Balancer DNS name to access AfterPay
application : AWS-ELB-01-2017992867.ap-southeast-2.elb.amazonaws.com
2019-09-22 12:59:54,453 cleaning up enviroment ....
2019-09-22 13:00:25,222 AfterPay App @ AWS Ready !
```

As per above example, Python script hase created an AWS ELB with a DNS name, Which can be used to access AfterPay application.

http://aws-elb-01-2017992867.ap-southeast-2.elb.amazonaws.com

7. Post Script Execution Verification

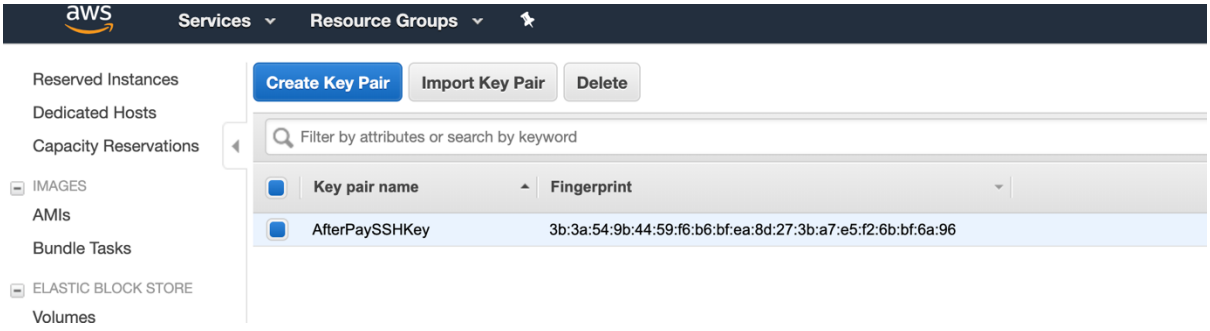
Elastic load balancer will be front end of AfterPay application. Application front end can be access via DNS name output by the python script. Script will create two EC2 instance attached to AutoScale group, which will provide

resilient AfterPay application. Furthermore, in order to access individual EC2 instance, use AfterPaySSHKey.pem certificate stored in local directory in bash (run “pwd” command to find current location). Depicted commands need to be executed to change permission and access EC2 instance.

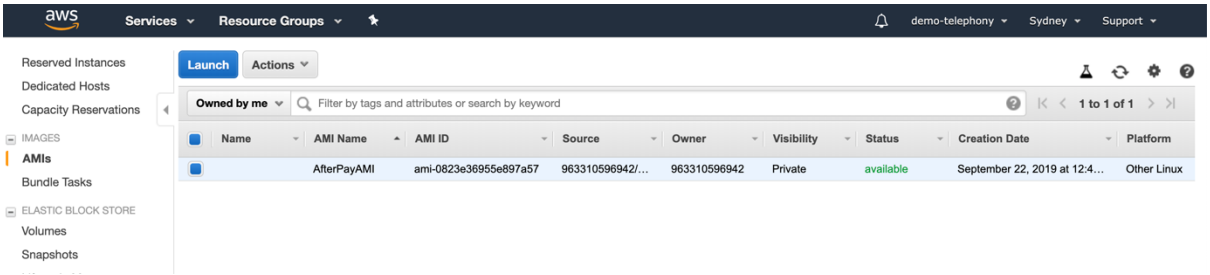
```
chmod 400 AfterPaySSHKey.pem
ssh -i AfterPaySSHKey.pem ec2-user@{Replace with EC2 IP or DNS}
```

7.1. AWS INFRASTRUCTURE FOR AFTERPAY APPLICATION

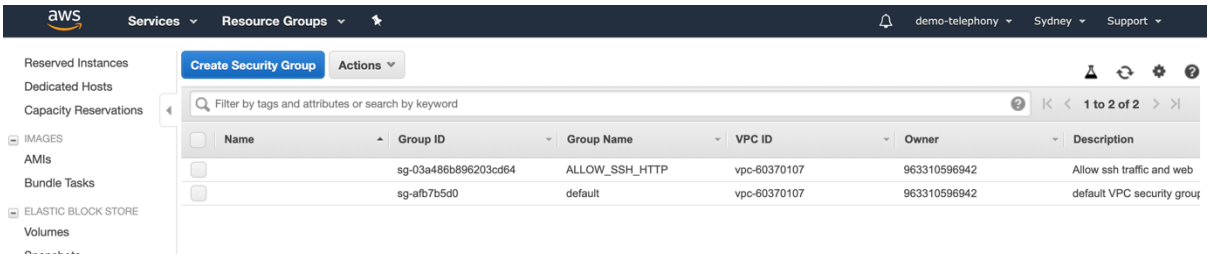
SSH Key for EC2 instance terminal access.



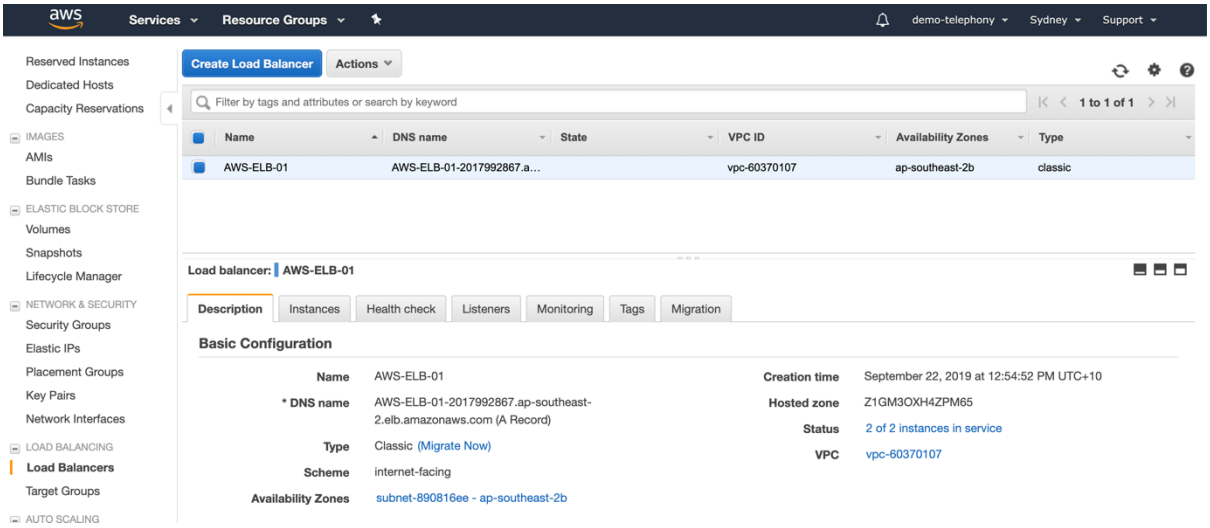
Customized AMI Linux OS image for AfterPay.



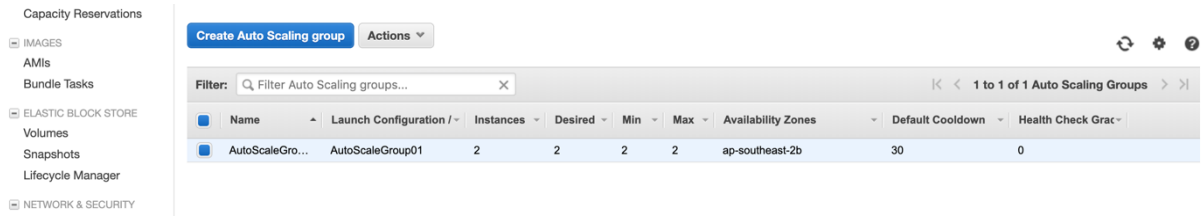
Security Group to allow only SSH and HTTP traffic.



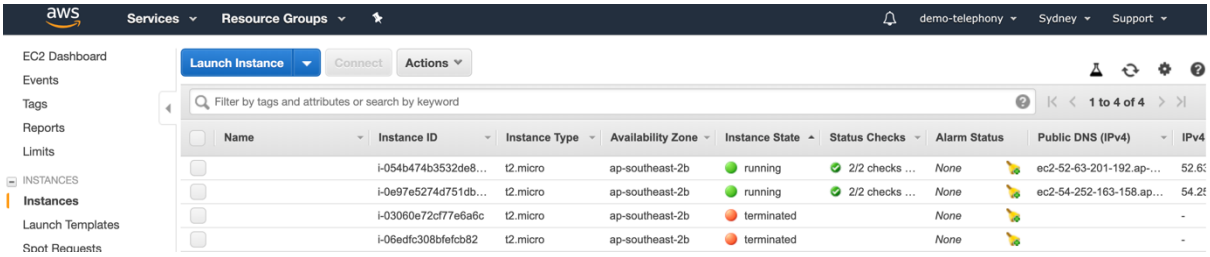
Elastic Load Balancer used for AfterPay app front end.



AfterPay AutoScale Group for Anti-Fragility



EC2 instance running under AfterPay AutoScale group. Reset of two EC2 instances with status terminated were used as part of intermediate steps.



APPENDIX

```
import time
import logging
import boto3
from operator import itemgetter


######### Create a filter to filter out Amazon Linux standard base image, which will be used to build
customized AMI for AfterPay. #########

findAmiClient = boto3.client('ec2')

filters = [ {
    'Name': 'name',
    'Values': ['amzn-ami-hvm-*']
},{
    'Name': 'description',
    'Values': ['Amazon Linux AMI*']
},{
    'Name': 'architecture',
    'Values': ['x86_64']
},{
    'Name': 'owner-alias',
    'Values': ['amazon']
},{
    'Name': 'state',
    'Values': ['available']
},{
    'Name': 'root-device-type',
    'Values': ['ebs']
},{
    'Name': 'virtualization-type',
    'Values': ['hvm']
},{
    'Name': 'hypervisor',
    'Values': ['xen']
},{
    'Name': 'image-type',
    'Values': ['machine']
} ]

######### Use above filter to search in Amazon for AMI ID ############
findAmiResponse = findAmiClient.describe_images(
  Filters=filters,
  Owners=[
      'amazon'
  ]
)

######### Sort on above results and creation AMI images on decending order. ###########

image_details = sorted(findAmiResponse['Images'],key=itemgetter('CreationDate'),reverse=True)
amiId = image_details[0]['ImageId']
logging.basicConfig(format='%(asctime)s %(message)s')


logging.warning('Script will use Amazon AMI ID : %s which will be used for  building base customized AMI
for AfterPay', amiId)

######## Creating Key Pair and saving it on local computer ##########
```

```
sshKeyClient = boto3.client('ec2')
sshKeyResponse = sshKeyClient.create_key_pair(KeyName='AfterPaySSHKey')
logging.warning('Generated a SSH-key file AfterPaySSHKey.pem whcih can be used to login to server and saved
it on local folder used to run this script.')

f= open("AfterPaySSHKey.pem","w+")
f.write(str(sshKeyResponse['KeyMaterial']))
f.close()

######## Creating an customized AMI as for the requirement given by AfterPay ########

printSpec = """Creating a customized AMI for After pay with below specifications
*All packages are update to date and all pending security updates are applied against the default OS
repositories.
*disable IPv6 system wide.
*install the following packages: ntp (NTP daemon)(make sure it is started at boot), telnet, mtr, tree.
*set the max "open files" limit across all users/processes, soft & hard, to 65535.
"""

logging.warning(printSpec)
logging.warning('Creating a customized AMI for AfterPay...')


userDataScriptAmi = """#!/bin/bash
sudo yum -y update --security
sudo yum -y update
sudo sysctl -w net.ipv6.conf.all.disable_ipv6=1
sudo sysctl -w net.ipv6.conf.default.disable_ipv6=1
sudo yum -y install ntp
sudo yum -y install telnet
sudo yum -y install mtr
sudo yum -y install tree
sudo systemctl start ntpd
sudo systemctl enable ntpd.service
echo "*         hard    nofile      65535" | sudo tee -a /etc/security/limits.conf
echo "*         soft    nofile      65535" | sudo tee -a /etc/security/limits.conf
echo "net.ipv6.conf.all.disable_ipv6 = 1" | sudo tee -a /etc/sysctl.conf
echo "net.ipv6.conf.default.disable_ipv6 = 1" | sudo tee -a /etc/sysctl.conf
echo "net.ipv6.conf.lo.disable_ipv6 = 1" | sudo tee -a /etc/sysctl.conf"""

ec2AmiResource = boto3.resource('ec2')
ec2InstanceAmi = ec2AmiResource.create_instances(ImageId=amiId,
                                                 InstanceType='t2.micro',
                                                 KeyName='AfterPaySSHKey',
                                                 MinCount=1,
                                                 MaxCount=1,
                                                 UserData=userDataScriptAmi)


logging.warning('Updatinge packages, installing services and configuring services in progress for customized
AMI for AfterPay ...')


checkStatusClient = boto3.client('ec2')
ec2AmiStatus = checkStatusClient.describe_instance_status(InstanceIds=[
                              ec2InstanceAmi[0].instance_id,
                            ])

while len(ec2AmiStatus['InstanceStatuses']) == 0:
       ec2AmiStatus = checkStatusClient.describe_instance_status(InstanceIds=[
                                     ec2InstanceAmi[0].instance_id,
                                   ])
#print(ec2AmiStatus['InstanceStatuses'][0]['InstanceState']['Name'])
```

```
time.sleep(240)

amiImage        =        boto3.client('ec2').create_image(InstanceId=ec2InstanceAmi[0].instance_id,
Name='AfterPayAMI')


if len(amiImage['ImageId']) == 0:
        time.sleep(60)

time.sleep(240)


logging.warning('Customized AMI image build for  AfterPay completed ... AMI image name is AfterPayAMI')




######## Creating Security Groups ##########

securityGroupClient = boto3.client('ec2')

sgResponseA = securityGroupClient.describe_vpcs()
vpcId = sgResponseA.get('Vpcs', [{}])[0].get('VpcId', '')

sgresponseB = securityGroupClient.create_security_group(GroupName='ALLOW_SSH_HTTP',
                                        Description='Allow ssh traffic and web',
                                        VpcId=vpcId)
securityGroupId = sgresponseB['GroupId']
logging.warning('Security Group Created %s in vpc %s.', securityGroupId, vpcId)

data = securityGroupClient.authorize_security_group_ingress(
        GroupId=securityGroupId,
        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'}]}
        ])


######### Creating Intermediate Server, installing Flask Development Server, download AfterPay app from Git
and run ########

userDataScript= """#!/bin/bash
mkdir myproject
cd myproject
python3 -m venv venv
. venv/bin/activate
sudo pip install Flask
wget https://raw.githubusercontent.com/AfterpayTouch/recruitment-challenge-1/master/tiny_app.py
export FLASK_APP=tiny_app.py
sed -i "s/app.run()/app.run(host='0.0.0.0', debug=True, port=80)/g" tiny_app.py
sudo python tiny_app.py"""

ec2ServerResource = boto3.resource('ec2')
ec2Server = ec2ServerResource.create_instances(ImageId=amiImage['ImageId'],
                                        InstanceType='t2.micro',
                                        KeyName='AfterPaySSHKey',
                                        MinCount=1,
                                        MaxCount=1,
```

```
                                        SecurityGroupIds=[
                                          securityGroupId,
                                        ],
                                        UserData=userDataScript)


checkStatusClient = boto3.client('ec2')
ec2ServerStatus = checkStatusClient.describe_instance_status(InstanceIds=[
                                    ec2Server[0].instance_id,
                                ])

while len(ec2ServerStatus['InstanceStatuses']) == 0:
        ec2ServerStatus = checkStatusClient.describe_instance_status(InstanceIds=[
                                    ec2Server[0].instance_id,
                                ])



time.sleep(240)

logging.warning('Installing Flask Development Server,Download AfterPay app from Git and run server on port
80')

######### Create an AutoScale Group with a Elastic Load Balancer ######


creatElbClient = boto3.client('elb')
elbResponse = creatElbClient.create_load_balancer(
    Listeners=[
        {
            'InstancePort': 80,
            'InstanceProtocol': 'HTTP',
            'LoadBalancerPort': 80,
            'Protocol': 'HTTP',
        },
    ],
    LoadBalancerName='AWS-ELB-01',
    SecurityGroups=[
        securityGroupId,
    ],
    Subnets=[
        ec2Server[0].subnet_id,
    ],
)



autoScaleClient = boto3.client('autoscaling')

creatASResponse = autoScaleClient.create_auto_scaling_group(
    AutoScalingGroupName='AutoScaleGroup01',
    InstanceId=ec2Server[0].instance_id,
    DesiredCapacity=2,
    DefaultCooldown=30,
    MaxSize=2,
    MinSize=2
)

attachElbResponse = autoScaleClient.attach_load_balancers(
    AutoScalingGroupName='AutoScaleGroup01',
    LoadBalancerNames=[
        'AWS-ELB-01',
    ]
)
```

```
######## Cleaning up intermediate EC2 instances #########


logging.warning('Hang on tight mate, It is almost there !')

time.sleep(300)

logging.warning('Please use Elastic Load Balancer DNS name to access AfterPay application : %s',
elbResponse['DNSName'])

logging.warning('cleaning up enviroment ....')

time.sleep(30)

cleanUpEc2Client = boto3.client('ec2')
cleanUpEc2Response = cleanUpEc2Client.terminate_instances(
    InstanceIds=[
        ec2Server[0].instance_id,
    ]
)

cleanUpEc2Client = boto3.client('ec2')
cleanUpEc2Response = cleanUpEc2Client.terminate_instances(
    InstanceIds=[
        ec2InstanceAmi[0].instance_id,
    ]
)

logging.warning('AfterPay App @ AWS Ready !')
```