**Graded Assignment on Monitoring, Scaling and Automation**

Using boto3 to create an S3 bucket to store your web application's static files.

import boto3

import os

# Initialize the S3 client

s3\_client = boto3.client('s3')

# Define the bucket nameclear

bucket\_name = 'prem-s3-static-bucket'

# Specify the region for the bucket

region = 'us-west-2' # Replace with your desired region

# Create the S3 bucket with region specified

s3\_client.create\_bucket(Bucket=bucket\_name, CreateBucketConfiguration={'LocationConstraint': region})

print(f'S3 bucket "{bucket\_name}" created successfully in region "{region}"!')

# Specify the directory containing static files

static\_files\_directory = '/Users/pkumar23/prem4eru/Monitorning\_Scaling\_Automation/staticwebpage'

# Initialize the S3 resource

s3\_resource = boto3.resource('s3')

# Upload static files to S3 bucket

for file\_name in os.listdir(static\_files\_directory):

file\_path = os.path.join(static\_files\_directory, file\_name)

with open(file\_path, 'rb') as file:

s3\_client.upload\_fileobj(file, bucket\_name, file\_name)

print(f"Uploaded '{file\_name}' to S3 bucket '{bucket\_name}'.")

print("Static files uploaded successfully.")

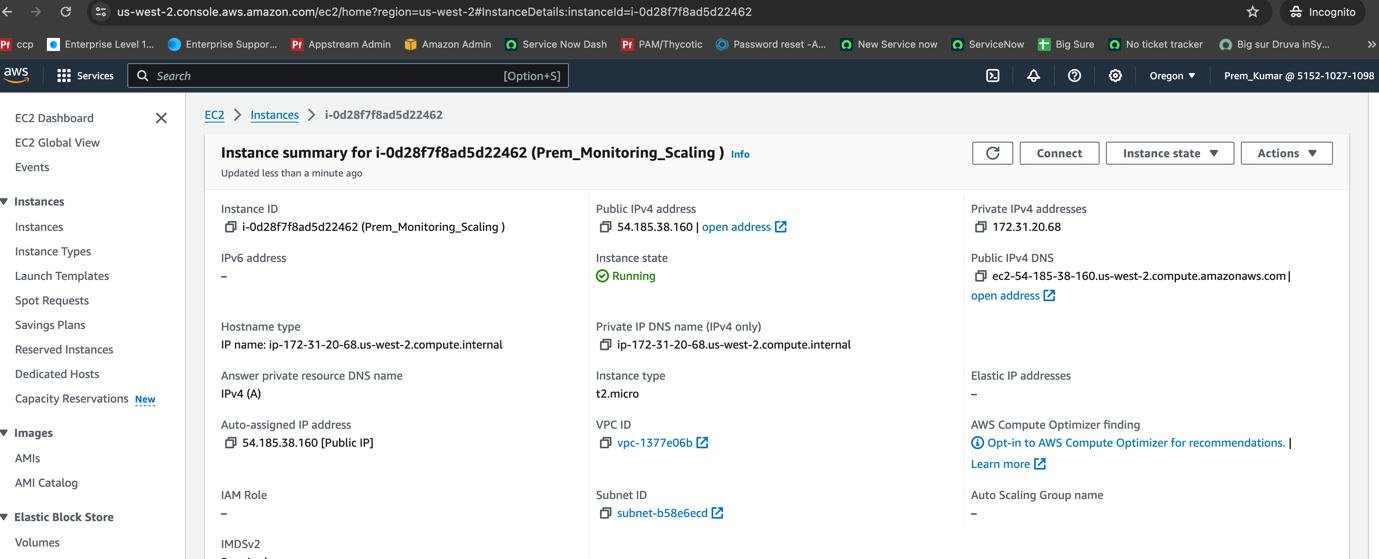
Output shows bucket created successfully and files uploaded to buckets.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated



We have created the Ec2 instance and installed the Nginx on Ec2 instance , Nginx server is on running state and able to deploy the static websites. It shows the output below Nginx server is running and able to access the static websites using the public IP of Ec2 server.

A screenshot of a computer program

Description automatically generated

A screenshot of a web page

Description automatically generated

Using boto3 creating the Application Load balancer and registering the Ec2 instance with ALB.

import boto3

# Initialize the EC2 client

ec2\_client = boto3.client('ec2')

# Create VPC

vpc\_response = ec2\_client.create\_vpc(CidrBlock='172.31.0.0/16')

vpc\_id = vpc\_response['Vpc']['VpcId']

# Initialize the ELBv2 client

elbv2\_client = boto3.client('elbv2')

# Create Subnets

subnet\_response = ec2\_client.create\_subnet(VpcId=vpc\_id, CidrBlock='172.31.16.0/20', AvailabilityZone='us-west-2b')

subnet\_id = subnet\_response['Subnet']['SubnetId']

subnet\_response = ec2\_client.create\_subnet(VpcId=vpc\_id, CidrBlock='172.31.48.0/20', AvailabilityZone='us-west-2a')

subnet\_id = subnet\_response['Subnet']['SubnetId']

# Create Security Group

security\_group\_response = ec2\_client.create\_security\_group(GroupName='ALBSecurityGroup', Description='ALB Security Group', VpcId=vpc\_id)

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

# Authorize Security Group Ingress (e.g., allow HTTP traffic)

ec2\_client.authorize\_security\_group\_ingress(

GroupId=security\_group\_id,

IpPermissions=[

{'IpProtocol': 'tcp',

'FromPort': 80,

'ToPort': 80,

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

]

)

# Create Target Group

target\_group\_response = elbv2\_client.create\_target\_group(

Name='MyTargetGroup',

Protocol='HTTP',

Port=80,

VpcId=vpc\_id

)

target\_group\_arn = target\_group\_response['TargetGroups'][0]['TargetGroupArn']

# Create Application Load Balancer

load\_balancer\_response = elbv2\_client.create\_load\_balancer(

Name='PremLoadBalancer',

Subnets=[subnet\_id],

SecurityGroups=[security\_group\_id],

Scheme='internet-facing',

Tags=[{'Key': 'Name', 'Value': 'PremLoadBalancer'}],

Type='application'

)

load\_balancer\_arn = load\_balancer\_response['LoadBalancers'][0]['LoadBalancerArn']

print('Application Load Balancer deployed successfully!')

A black background with white text

Description automatically generated

It shows that Able to deploy the Application Load balancer name : PremLoadBalancer.

A screenshot of a computer

Description automatically generated

Using the boto3 created an AGS and deployed Ec2 instance as template.

import boto3

# Initialize the Auto Scaling client

autoscaling\_client = boto3.client('autoscaling')

# Define the EC2 instance ID

instance\_id = 'i-0d28f7f8ad5d22462' # Replace with your EC2 instance ID

# Create a launch configuration

launch\_config\_response = autoscaling\_client.create\_launch\_configuration(

LaunchConfigurationName='MyLaunchConfig',

ImageId='ami-0cf2b4e024cdb6960', # Replace with your AMI ID

InstanceType='t2.micro', # Replace with your desired instance type

KeyName='tm.pem', # Replace with your key pair name

SecurityGroups=['launch-wizard-57'], # Replace with your security group(s)

#UserData='your-user-data', # Replace with your user data script

InstanceMonitoring={'Enabled': True} # Enable detailed monitoring if needed

)

# Define the Auto Scaling Group parameters

desired\_capacity = 1 # Set the desired capacity

min\_size = 1 # Set the minimum size

max\_size = 2 # Set the maximum size

# Create the Auto Scaling Group

asg\_response = autoscaling\_client.create\_auto\_scaling\_group(

AutoScalingGroupName='PremMyAutoScalingGroup',

LaunchConfigurationName='MyLaunchConfig',

MinSize=min\_size,

MaxSize=max\_size,

DesiredCapacity=desired capacity,

AvailabilityZones=['us-west-2b'], # Replace with your desired AZ(s)

VPCZoneIdentifier='subnet-b58e6ecd', # Replace with your subnet ID(s)

Tags=[

{

'Key': 'Name',

'Value': 'MyASGInstance'

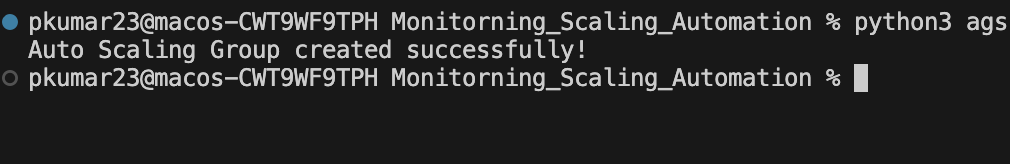
},

# Add more tags as needed

]

)

print('Auto Scaling Group created successfully!')



A screenshot of a computer

Description automatically generated

It shows that Autoscaling group has been deployed successfully and configured scaling policies to scale in/out based on metrics like CPU utilization or network traffic.

A screenshot of a computer

Description automatically generated

Setup different SNS topics for different alerts (e.g., health issues, scaling events, high traffic)

A screenshot of a computer

Description automatically generated

A screenshot of a email

Description automatically generated

Infrastructure automation : using boto3 deployed the entire infrastructure and updated any component as required tear down the everything when application is no longer needed.

import boto3

import time

ec2\_client = boto3.client('ec2')

s3\_client = boto3.client('s3')

def create\_infrastructure():

# Create VPC

vpc\_response = ec2\_client.create\_vpc(

CidrBlock='172.31.0.0/16',

TagSpecifications=[

{

'ResourceType': 'vpc',

'Tags': [

{

'Key': 'Name',

'Value': 'MyVPC'

},

]

},

]

)

vpc\_id = vpc\_response['Vpc']['VpcId']

# Create Subnet

subnet\_response = ec2\_client.create\_subnet(

AvailabilityZone='us-west-2b',

CidrBlock='172.31.16.0/20',

VpcId=vpc\_id,

TagSpecifications=[

{

'ResourceType': 'subnet',

'Tags': [

{

'Key': 'Name',

'Value': 'MySubnet'

},

]

},

]

)

subnet\_id = subnet\_response['Subnet']['SubnetId']

# Create Security Group

security\_group\_response = ec2\_client.create\_security\_group(

Description='MySecurityGroup',

GroupName='MySecurityGroup',

VpcId=vpc\_id

)

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

# Add Inbound Rule to Security Group

ec2\_client.authorize\_security\_group\_ingress(

GroupId=security\_group\_id,

IpPermissions=[

{

'IpProtocol': 'tcp',

'FromPort': 22,

'ToPort': 22,

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

}

]

)

# Create EC2 Instance

instance\_response = ec2\_client.run\_instances(

ImageId='ami-0cf2b4e024cdb6960', # Change to your desired AMI ID

InstanceType='t2.micro',

MinCount=1,

MaxCount=1,

KeyName='tm.pem', # Change to your key pair name

SecurityGroupIds=[security\_group\_id],

SubnetId=subnet\_id

)

instance\_id = instance\_response['Instances'][0]['InstanceId']

# Create S3 Bucket

s3\_client.create\_bucket(

Bucket='my-unique-bucket-name',

CreateBucketConfiguration={

'LocationConstraint': 'us-east-1'

}

)

print("Infrastructure deployed successfully!")

return vpc\_id, subnet\_id, security\_group\_id, instance\_id

def update\_infrastructure(security\_group\_id):

# Update Inbound Rule to Security Group

ec2\_client.authorize\_security\_group\_ingress(

GroupId=security\_group\_id,

IpPermissions=[

{

'IpProtocol': 'tcp',

'FromPort': 80,

'ToPort': 80,

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

}

]

)

print("Security group updated successfully!")

def teardown\_infrastructure(vpc\_id, subnet\_id, security\_group\_id, instance\_id):

# Terminate EC2 Instance

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

print("EC2 instance terminated successfully!")

# Delete Security Group

ec2\_client.delete\_security\_group(GroupId=security\_group\_id)

print("Security group deleted successfully!")

# Delete Subnet

ec2\_client.delete\_subnet(SubnetId=subnet\_id)

print("Subnet deleted successfully!")

# Delete VPC

ec2\_client.delete\_vpc(VpcId=vpc\_id)

print("VPC deleted successfully!")

# Delete S3 Bucket

s3\_client.delete\_bucket(Bucket='my-unique-bucket-name')

print("S3 bucket deleted successfully!")

print("Infrastructure teardown completed!")

# Deploy infrastructure

vpc\_id, subnet\_id, security\_group\_id, instance\_id = create\_infrastructure()

# Wait for a few seconds for the resources to be ready

time.sleep(10)

# Update infrastructure

update\_infrastructure(security\_group\_id)

# Wait for a few seconds for the update to take effect

time.sleep(10)

# Tear down infrastructure

teardown\_infrastructure(vpc\_id, subnet\_id, security\_group\_id, instance\_id)

it shows the below Output :

Infrastructure deployed successfully with all the component and teared down when application is no longer needed.

