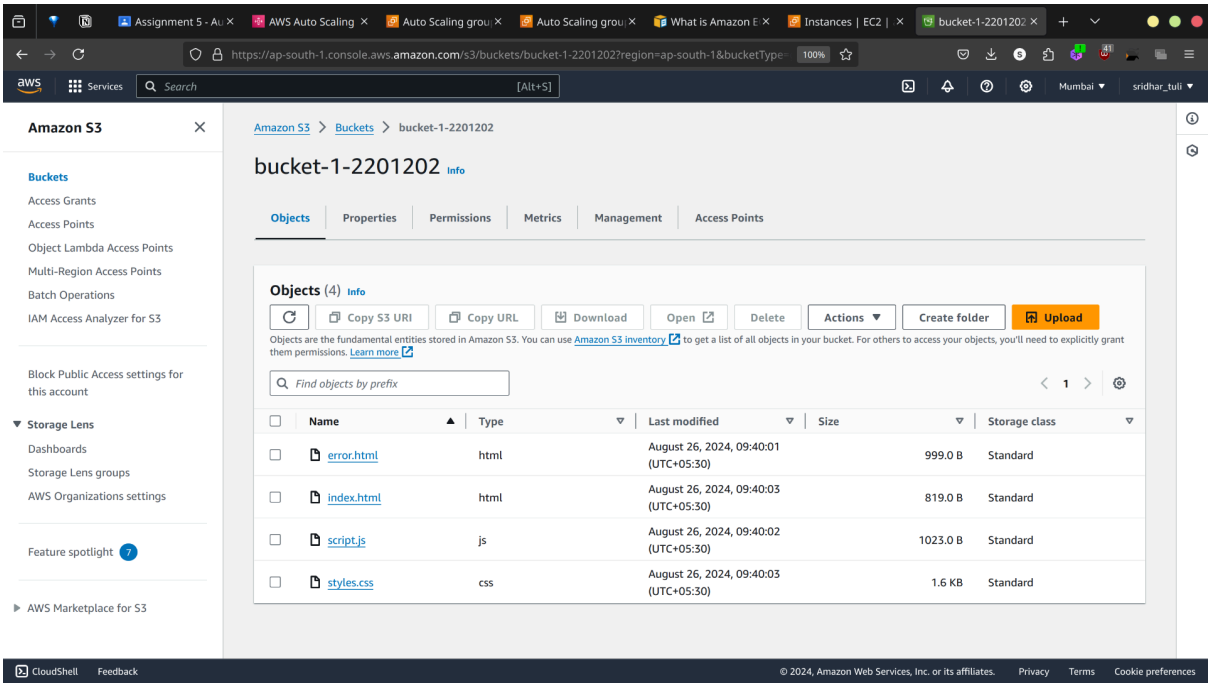
Cloud Lab Assignment 5
Sridhar Tuli
2201202

You need to create an auto-scaling configuration for the web tier of a cloud application. Follow the steps below.

● Create a static website with one or two HTML pages.



Create a startup script to install Apache server.

```
~/AWS/lab5 (0.007s)
ls
boto.py  startup.sh

~/AWS/lab5 (0.005s)
cat startup.sh
#!/bin/bash

yes|sudo apt-get update
yes|sudo apt-get install apache2 -y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install unzip
unzip awscliv2.zip
sudo ./aws/install
sudo aws configure set aws_access_key_id "AKIATWBJZ7ONRBVEGN7G"
sudo aws configure set aws_secret_access_key "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"
sudo aws configure set region "ap-south-1"
sudo aws s3 sync s3://bucket-1-2201202/ /var/www/html/
sudo systemctl start apache2
sudo systemctl enable apache2

~/AWS/lab5
```

Create a Python program using boto that creates an Amazon AutoScaling group. Define scale up and scale down policies and corresponding CloudWatch alarms. Supply the startup script you created in the previous step while launching a new instance from the program. Use a security group with port 80 open. Copy the previously created pages and dependent files in the newly launched EC2 Instance.

Initializing the Code

```python
import boto3
import base64
import webbrowser
import time

ec2 = boto3.client('ec2')
autoscaling = boto3.client('autoscaling')
cloudwatch = boto3.client('cloudwatch')

ami_id = 'ami-0522ab6e1ddcc7055'
instance_type = 't2.micro'
key_name = 'keypair-lab3-new'
security_group_name = 'MyWebServerSG'
launch_configuration_name = 'my-launch-config'
auto_scaling_group_name = 'my-auto-scaling-group'
description = 'Security group for web server allowing HTTP traffic on port 80'

with open('startup.sh', 'r') as file:
    user_data_script = file.read()

user_data_encoded = base64.b64encode(user_data_script.encode('utf-8')).decode('utf-8')
```

Creating Security Groups

```python
def create_security_group():
    try:
        existing_groups = ec2.describe_security_groups(GroupNames=[security_group_name])
        security_group_id = existing_groups['SecurityGroups'][0]['GroupId']
        print(f'Using existing security group with ID: {security_group_id}')
    except ec2.exceptions.ClientError as e:
        if 'InvalidGroup.NotFound' in str(e):
            response = ec2.create_security_group(GroupName=security_group_name, Description=description)
            security_group_id = response['GroupId']
            print(f'Created security group with ID: {security_group_id}')

            ec2.authorize_security_group_ingress(
                GroupId=security_group_id,
                IpPermissions=[
                    {
                        'IpProtocol': 'tcp',
                        'FromPort': 80,
                        'ToPort': 80,
                        'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
                    }
                ]
            )
            print(f'Added rule to allow inbound traffic on port 80 to security group: {security_group_name}')
        else:
            raise
    return security_group_id
```

Setting up the Launch Configuration

```python
def create_launch_configuration(security_group_id):
    try:
        existing_configs = autoscaling.describe_launch_configurations(LaunchConfigurationNames=[launch_configuration_name])
        print(f'Existing launch configuration found: {launch_configuration_name}')

        try:
            autoscaling.delete_launch_configuration(LaunchConfigurationName=launch_configuration_name)
            print(f'Deleted existing launch configuration: {launch_configuration_name}')
        except autoscaling.exceptions.ClientError as e:
            if 'ValidationError' in str(e) and 'Launch configuration name not found' in str(e):
                print(f'Launch configuration {launch_configuration_name} no longer exists, proceeding to create a new one.')
            else:
                raise
    except autoscaling.exceptions.ClientError as e:
        if 'LaunchConfigurationName.NotFound' in str(e):
            print(f'No existing launch configuration found with name: {launch_configuration_name}')
        else:
            raise

    autoscaling.create_launch_configuration(
        LaunchConfigurationName=launch_configuration_name,
        ImageId=ami_id,
        KeyName=key_name,
        SecurityGroups=[security_group_id],
        InstanceType=instance_type,
        UserData=user_data_encoded
    )
    print(f'Launch configuration {launch_configuration_name} created.')
```

Auto Scaling Groups and Scaling Policies

```python
def create_auto_scaling_group():
    autoscaling.create_auto_scaling_group(
        AutoScalingGroupName=auto_scaling_group_name,
        LaunchConfigurationName=launch_configuration_name,
        MinSize=1,
        MaxSize=5,
        DesiredCapacity=1,
        AvailabilityZones=['ap-south-1a', 'ap-south-1b'],
        HealthCheckType='EC2',
        HealthCheckGracePeriod=300
    )
    print(f'Auto Scaling Group {auto_scaling_group_name} created.')


def create_scaling_policies():
    scale_up_policy = autoscaling.put_scaling_policy(
        AutoScalingGroupName=auto_scaling_group_name,
        PolicyName='scale-up',
        ScalingAdjustment=1,
        AdjustmentType='ChangeInCapacity'
    )

    scale_down_policy = autoscaling.put_scaling_policy(
        AutoScalingGroupName=auto_scaling_group_name,
        PolicyName='scale-down',
        ScalingAdjustment=-1,
        AdjustmentType='ChangeInCapacity'
    )

    return scale_up_policy['PolicyARN'], scale_down_policy['PolicyARN']
```

Cloud Watch Alarm

```python
def create_cloudwatch_alarms(scale_up_policy_arn, scale_down_policy_arn):
    cloudwatch.put_metric_alarm(
        AlarmName='scale-up-alarm',
        MetricName='CPUUtilization',
        Namespace='AWS/EC2',
        Statistic='Average',
        Period=300,
        Threshold=75.0,
        ComparisonOperator='GreaterThanThreshold',
        EvaluationPeriods=2,
        AlarmActions=[scale_up_policy_arn],
        Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': auto_scaling_group_name}]
    )
    print('Scale-up CloudWatch alarm created.')

    cloudwatch.put_metric_alarm(
        AlarmName='scale-down-alarm',
        MetricName='CPUUtilization',
        Namespace='AWS/EC2',
        Statistic='Average',
        Period=300,
        Threshold=25.0,
        ComparisonOperator='LessThanThreshold',
        EvaluationPeriods=2,
        AlarmActions=[scale_down_policy_arn],
        Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': auto_scaling_group_name}]
    )
    print('Scale-down CloudWatch alarm created.')
```

Getting the public DNS to access the website hosted

```python
def get_instance_public_dns():
    print("Waiting for instance to be launched...")
    time.sleep(60)

    response = autoscaling.describe_auto_scaling_groups(
        AutoScalingGroupNames=[auto_scaling_group_name]
    )
    instance_ids = [instance['InstanceId'] for instance in response['AutoScalingGroups'][0]['Instances']]

    if not instance_ids:
        print("No instances found in the Auto Scaling group.")
        return None

    instance_info = ec2.describe_instances(InstanceIds=[instance_ids[0]])
    public_dns = instance_info['Reservations'][0]['Instances'][0]['PublicDnsName']

    return public_dns
```

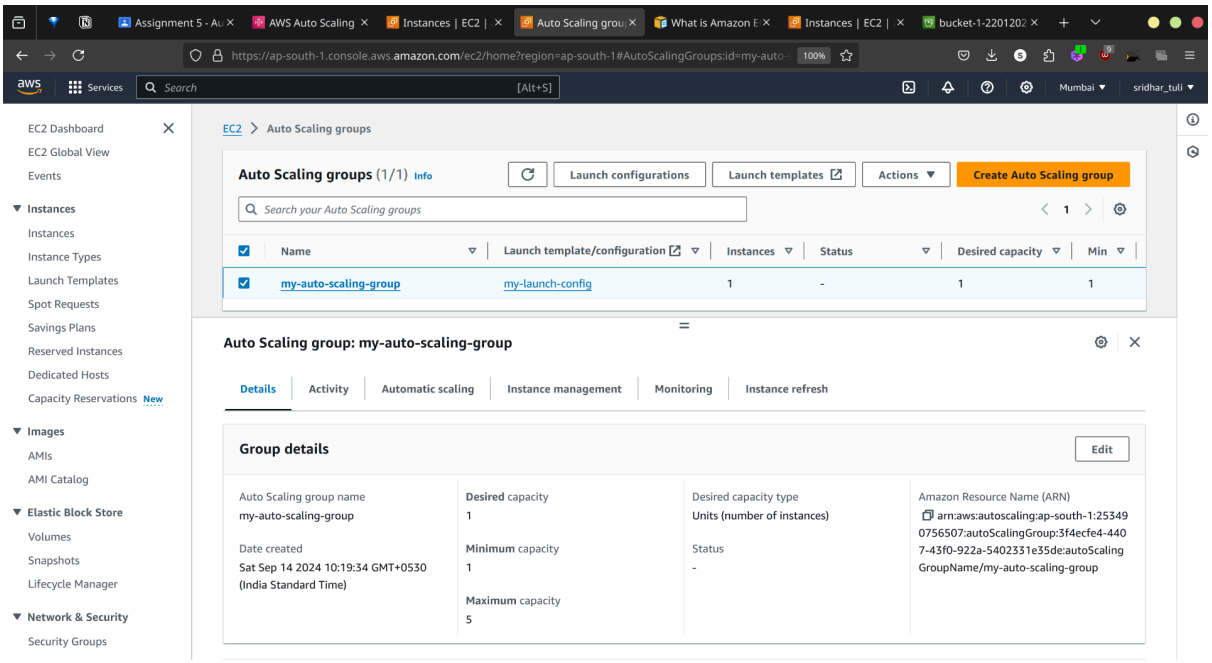
The main function

```python
if __name__ == "__main__":
    security_group_id = create_security_group()
    print("Security group created")
    create_launch_configuration(security_group_id)
    print("Launch configuration created")
                ()
    print("AutoScaling group created")
    scale_up_policy_arn, scale_down_policy_arn = create_scaling_policies()
    print("Scaling policies created")
    create_cloudwatch_alarms(scale_up_policy_arn, scale_down_policy_arn)
    print("CloudWatch alarms created")
    print("AutoScaling group, policies, and alarms created successfully.")

    public_dns = get_instance_public_dns()
    if public_dns:
        website_url = f"http://{public_dns}"
        print(f"Opening the website: {website_url}")
        webbrowser.open(website_url)
        print("Please check your browser to verify if the static website works.")
    else:
        print("Unable to retrieve the public DNS. Please check the AWS console for more information.")
```

Output of the code

```
BotoEnv ~/AWS/lab5 (1m 4.78s)
python boto.py
Using existing security group with ID: sg-083c6a7b961c341d9
Security group created
Existing launch configuration found: my-launch-config
Deleted existing launch configuration: my-launch-config
Launch configuration my-launch-config created.
Launch configuration created
Auto Scaling Group my-auto-scaling-group created.
AutoScaling group created
Scaling policies created
Scale-up CloudWatch alarm created.
Scale-down CloudWatch alarm created.
CloudWatch alarms created
AutoScaling group, policies, and alarms created successfully.
Waiting for instance to be launched...
Opening the website: http://ec2-13-201-46-240.ap-south-1.compute.amazonaws.com
Please check your browser to verify if the static website works.

BotoEnv ~/AWS/lab5
```

Auto Scaling Groups being created on AWS

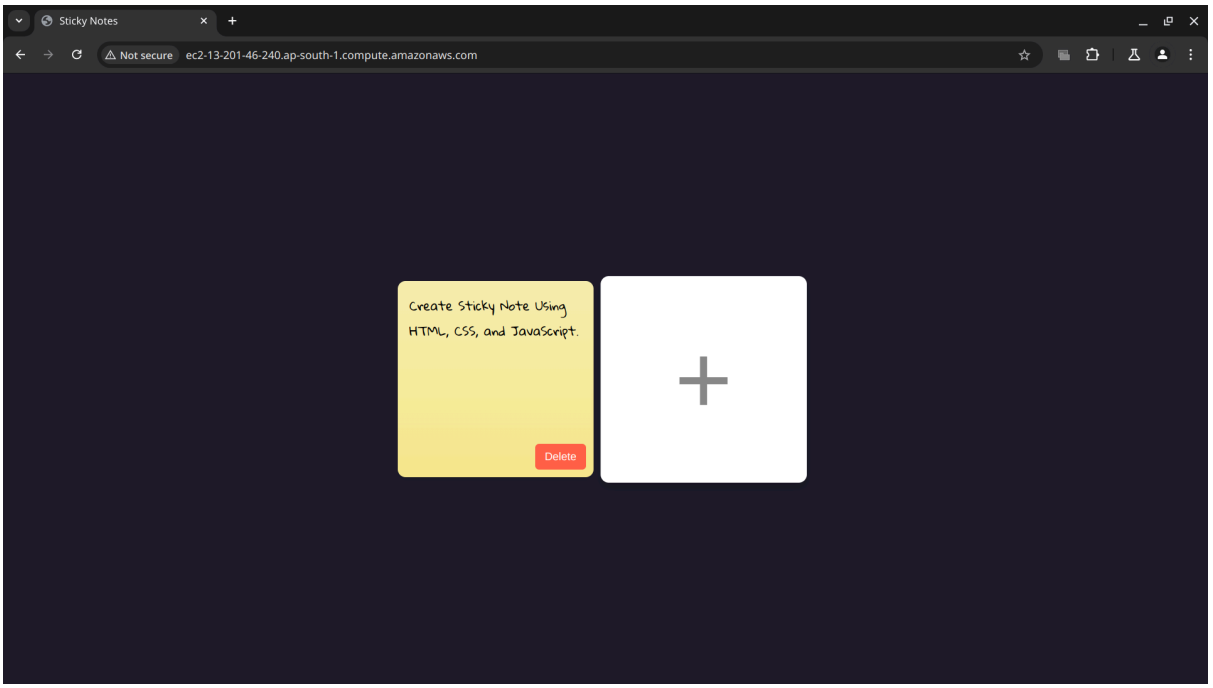


Q. Open the public DNS of the newly launched EC2 instance in a browser and verify if the static website works.

Opening the website

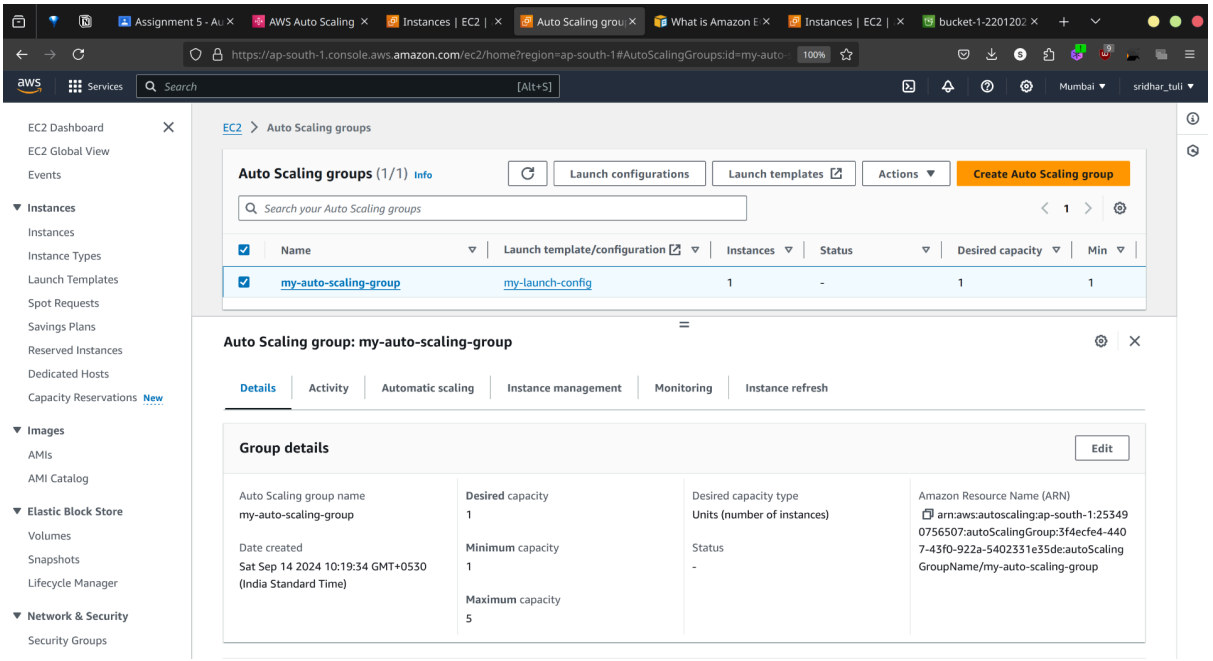
i) Launching of Apache Server

ii) After some time the website will launch

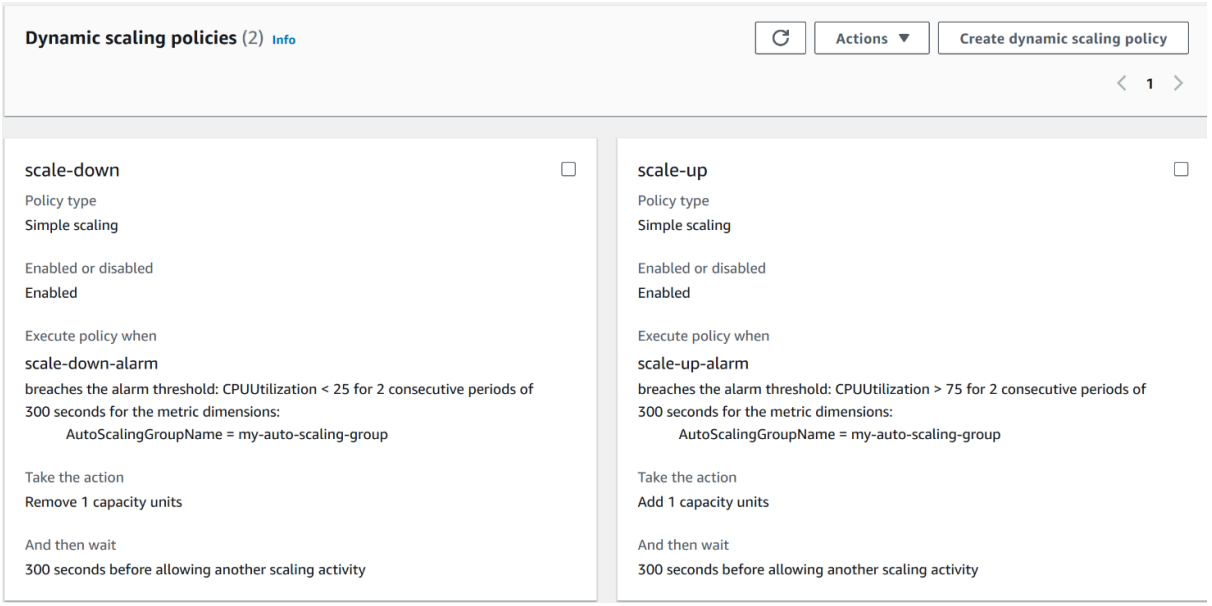


2nd Question

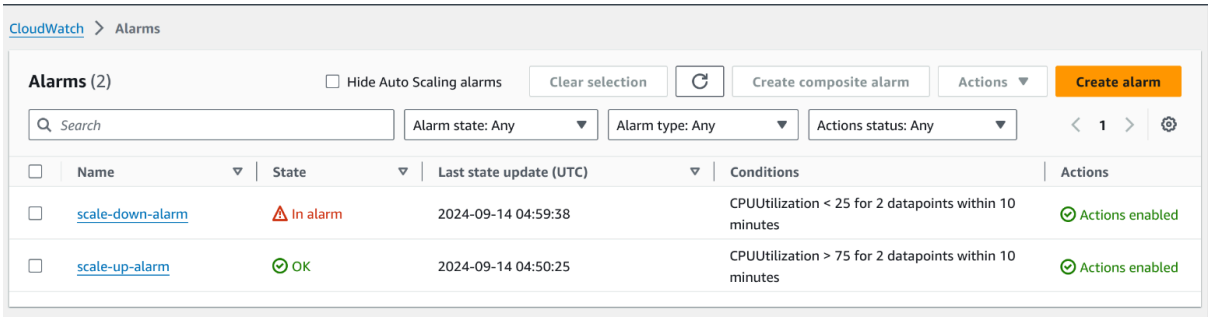
1. Go to the Amazon EC2 dashboard. Go to Auto Scaling Groups.
2. Define launch configuration. Create an Auto Scaling Group. Create the Scaling Policies (for Scale Up and Scale Down).
3. Create Scale Up Alarm and Scale Down Alarm using CloudWatch Alarms.
4. Take necessary screenshots. Delete the Auto Scaling Group and launch configuration after your work is done.



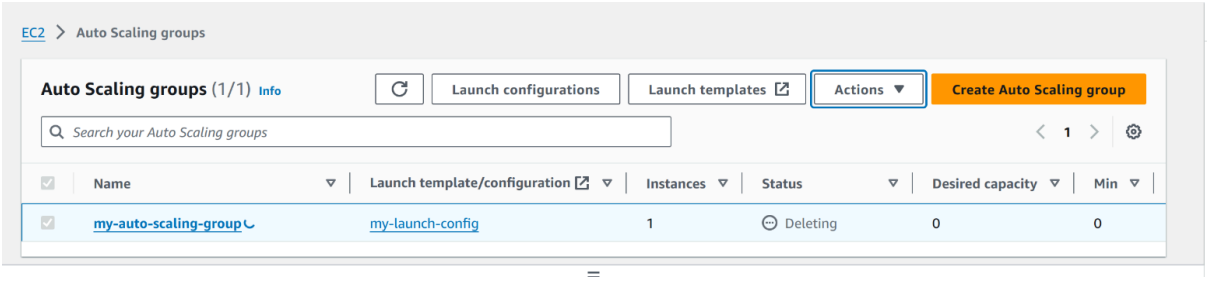
Scaling Policies on AWS Site



Cloud Watch Alarms



Deleting the AutoScaling Group



====END OF FILE====