|  |
| --- |
| Assignment 2  Auto-Scaling Web Application |
| Development Operations26670 (2023 - 2024) Student: Michael McKibbin  ID: 20092733 |



A black and white logo

Description automatically generated

Contents

[Planning: Application selection and architecture planning, including architecture diagram. AMI creation 4](#_Toc175529259)

[Installation of nvm, node, and app.js 5](#_Toc175529260)

[VPC and load balancer implementation. 8](#_Toc175529261)

[Testing: Test traffic and load balancing demonstration. 26](#_Toc175529262)

[Monitoring: Own script. 29](#_Toc175529263)

[APPENDIX A: Automated instance setup. 30](#_Toc175529264)

[APPENDIX B: monitor.sh - Instance info. 39](#_Toc175529265)

[APPENDIX C: memv2.sh - Cloudwatch. 40](#_Toc175529266)

Table of Figures

[Figure 1 Installing nvm via SSH 5](#_Toc175529029)

[Figure 2 Installing Node.js and running app.js 5](#_Toc175529030)

[Figure 3 Initial version of app.js - Hello World. 6](#_Toc175529031)

[Figure 4 Server running. 6](#_Toc175529032)

[Figure 5 Updated app.js - Server instance IP address. 7](#_Toc175529033)

[Figure 6 Killing the process and restarting app.js to start new version. 7](#_Toc175529034)

[Figure 7 Preview of VPC configuration 8](#_Toc175529035)

[Figure 8 VPC workflow complete. 9](#_Toc175529036)

[Figure 9 VPC configured and active. 10](#_Toc175529037)

[Figure 10 VPC Route Tables. 10](#_Toc175529038)

[Figure 11 Security Groups 11](#_Toc175529039)

[Figure 12 VPC Resource Map with a route highlighted. 11](#_Toc175529040)

[Figure 13 Editing Subnet Settings. 12](#_Toc175529041)

[Figure 14 Set IP address to Auto Assignment 12](#_Toc175529042)

[Figure 15 Launch Template Success. 13](#_Toc175529043)

[Figure 16 Launching an Instance - part 1 – choosing an AMI. 14](#_Toc175529044)

[Figure 17 Launching an Instance - part 2 – Key Pair, Network Settings (Subnets), and Security Group. 15](#_Toc175529045)

[Figure 18 Launching an Instance - part 3 - adding a script. 16](#_Toc175529046)

[Figure 19 Launch Instance with key pair and Network settings. 17](#_Toc175529047)

[Figure 20 Before Target Group setup. 18](#_Toc175529048)

[Figure 21 Load balancing Target Group. 18](#_Toc175529049)

[Figure 22 Load Balancer created successfully. 19](#_Toc175529050)

[Figure 23 Load Balancer details in browser. 19](#_Toc175529051)

[Figure 24 Creating an Auto Scaling group. 20](#_Toc175529052)

[Figure 25 Auto Scaling - Choosing a Launch Template and Network. 21](#_Toc175529053)

[Figure 26 Auto Scaling -Maximum instances, SNS Notifications, and Tags. 22](#_Toc175529054)

[Figure 27 Auto Scaling - First Auto Scaled Instance. 23](#_Toc175529055)

[Figure 28 Auto Scaling - First Instance Running. 23](#_Toc175529056)

[Figure 29 Auto Scaling - Targets registered. 24](#_Toc175529057)

[Figure 30 Auto Scaling - Cloudwatch Metrics. 25](#_Toc175529058)

[Figure 31 Auto Scaling - Cloudwatch Alarms in progress. 25](#_Toc175529059)

[Figure 32 Auto Scaling - Using an infinite loop to create traffic. 26](#_Toc175529060)

[Figure 33 Curl command generates 100 requests. 27](#_Toc175529061)

[Figure 34 Adding Listeners on port 80 and port 3000. 27](#_Toc175529062)

[Figure 35 Auto Scaled instances running. 28](#_Toc175529063)

[Figure 36 Auto Scaled instances - Running and Terminated. 28](#_Toc175529064)

### Planning: Application selection and architecture planning, including architecture diagram. AMI creation

Basic architecture:

Cloud – VPC – Availability Zones – Autoscaling group – Subnets.

A screenshot of a computer

Description automatically generated

I used a custom script to launch an EC2 instance, set the key pair and security group, install Node.js, upload app.js, run the app, open a web browser page to show instance information and another page to show the app in action. First ‘Hello World’ and later showing the Server instance IP address.

From this instance I created a custom AMI and updated the script to use this AMI for future instances which created instances with the index.html page, the monitoring script, and the application running on port 3000.

### Installation of nvm, node, and app.js

A computer screen shot of a program

Description automatically generated

Figure Installing nvm via SSH

A screenshot of a computer

Description automatically generated

Figure Installing Node.js and running app.js

A screenshot of a computer

Description automatically generated

Figure Initial version of app.js - Hello World.

A black background with white text

Description automatically generated

Figure Server running.

A screenshot of a computer

Description automatically generated

Figure Updated app.js - Server instance IP address.

A computer screen with white text

Description automatically generated

Figure Killing the process and restarting app.js to start new version.

### VPC and load balancer implementation.

I created a VPC with three availability zones, us-east-1a, 1b, and 1c ,with public and private subnets in each.

An Elastic Load Balancer was configured to use my custom AMI and linked to an Autoscaling Group

The Load Balancer used Simple Scaling triggered by Cloudwatch alarms to increase or decrease the number of instances based on CPU usage.

When usage is over 50% another instance is created, with a limit of three.

When the work creating loop is terminated the low usage alarm kills unneeded instances when CPU usage drops below 30%.

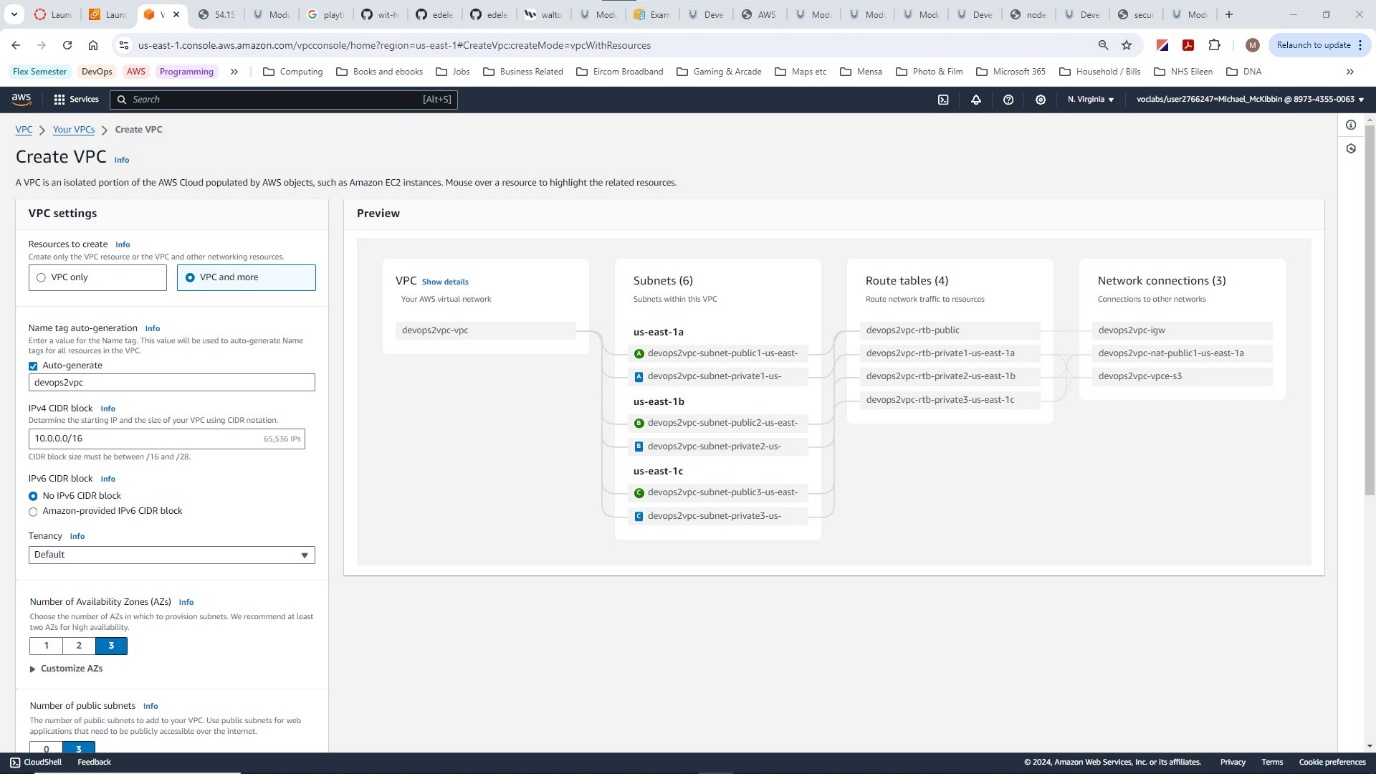


Figure Preview of VPC configuration

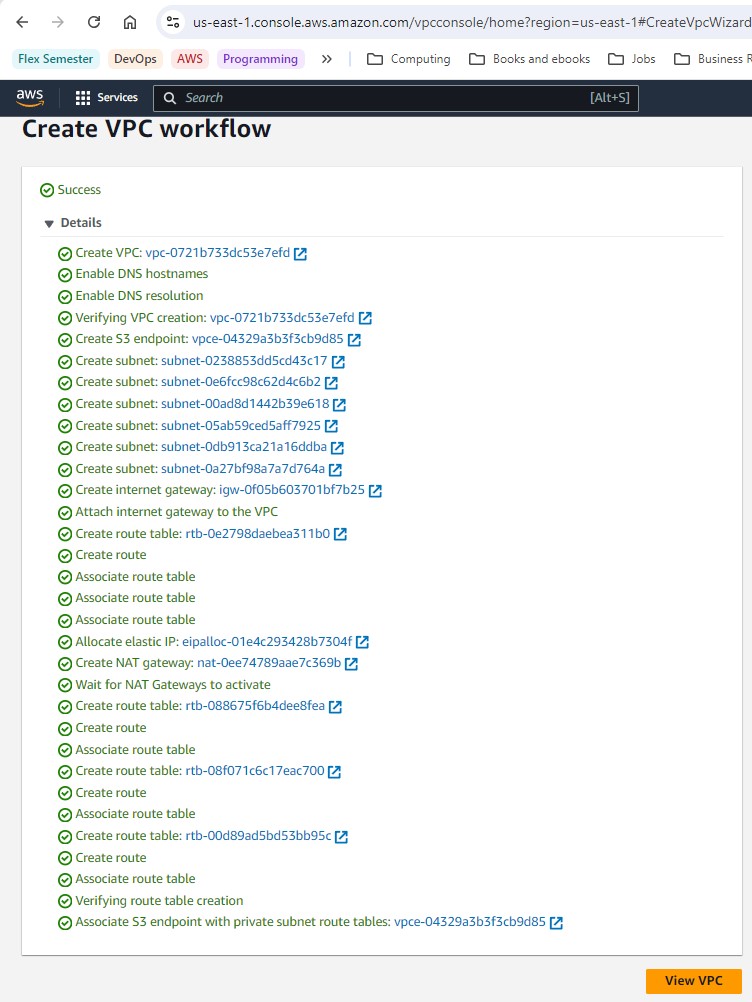
****

Figure VPC workflow complete.

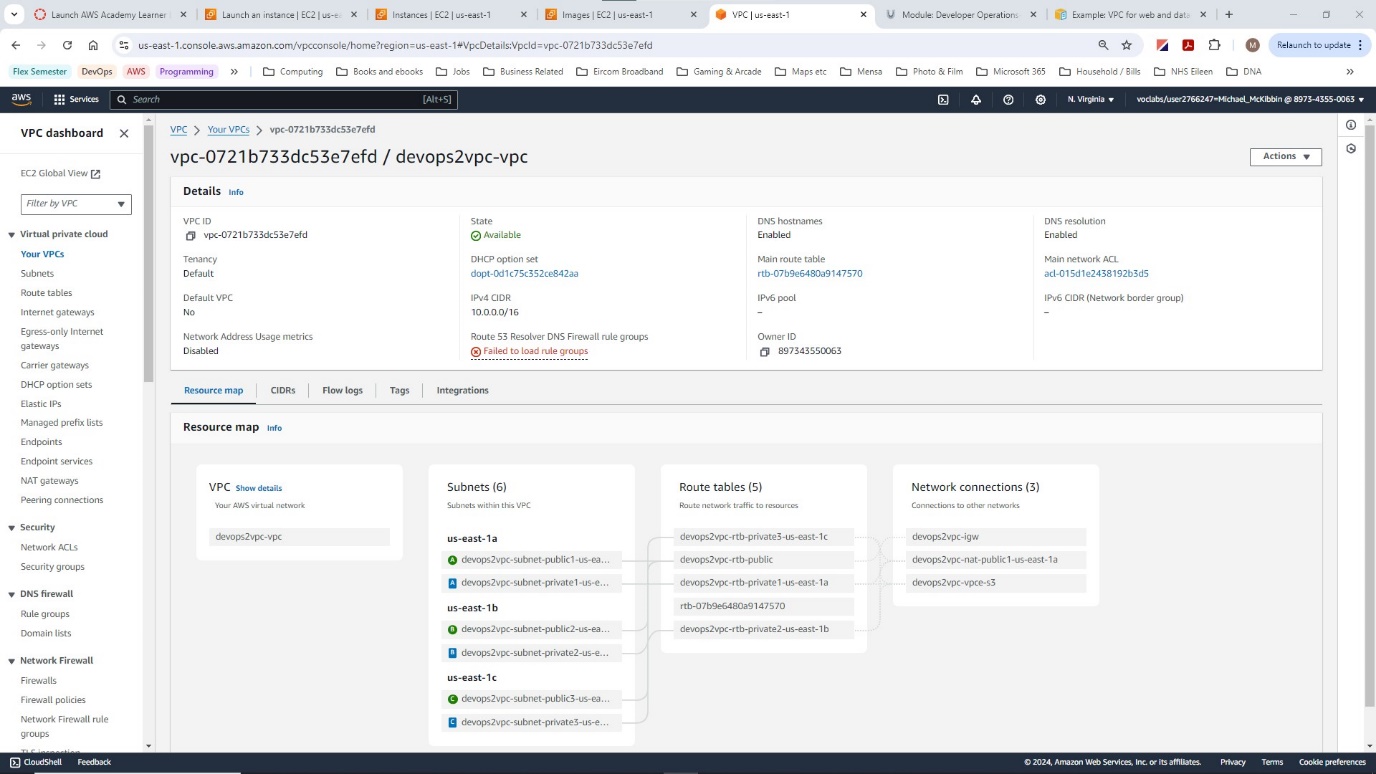
****

Figure VPC configured and active.

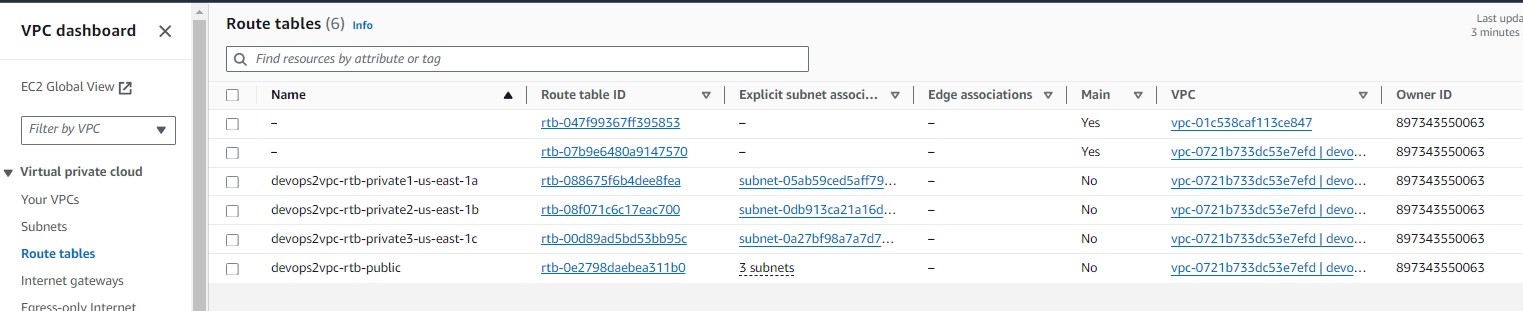


Figure VPC Route Tables.

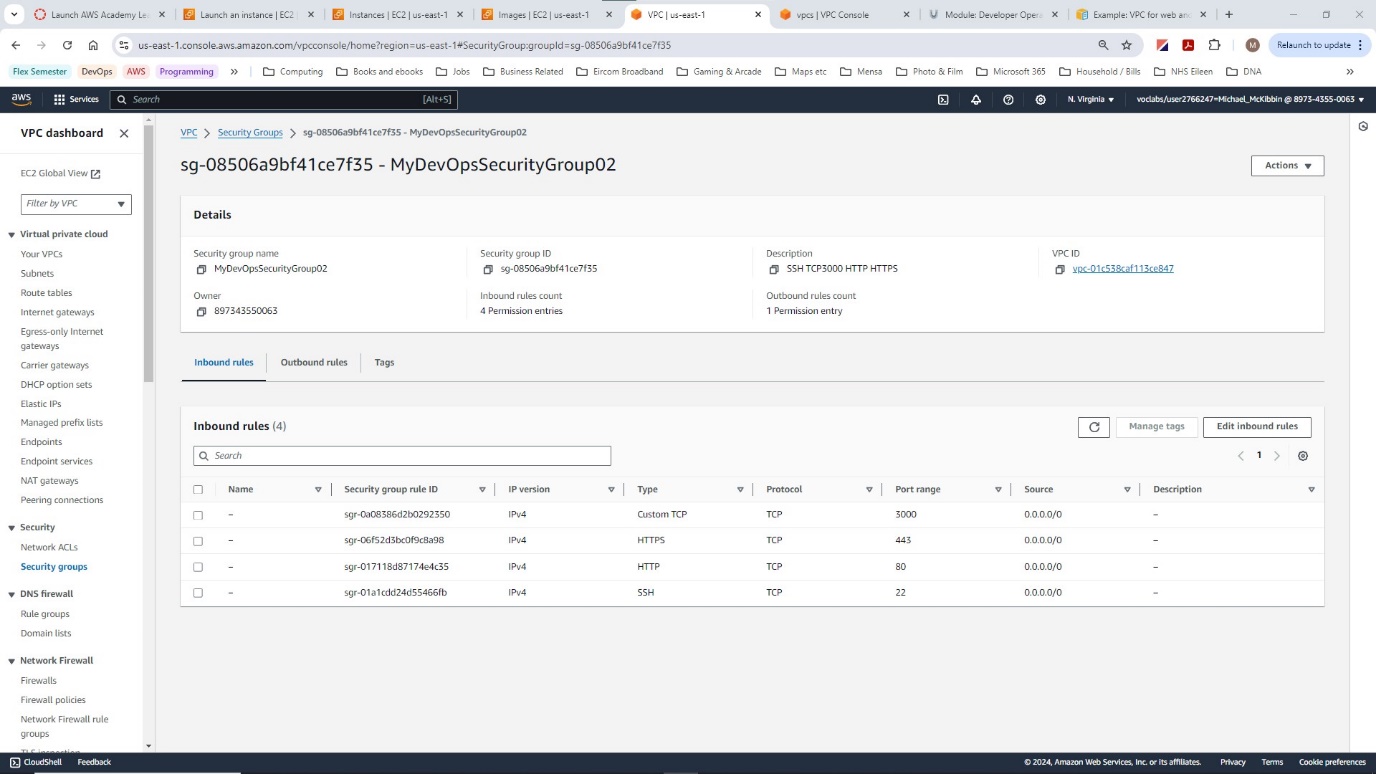
****

Figure Security Groups

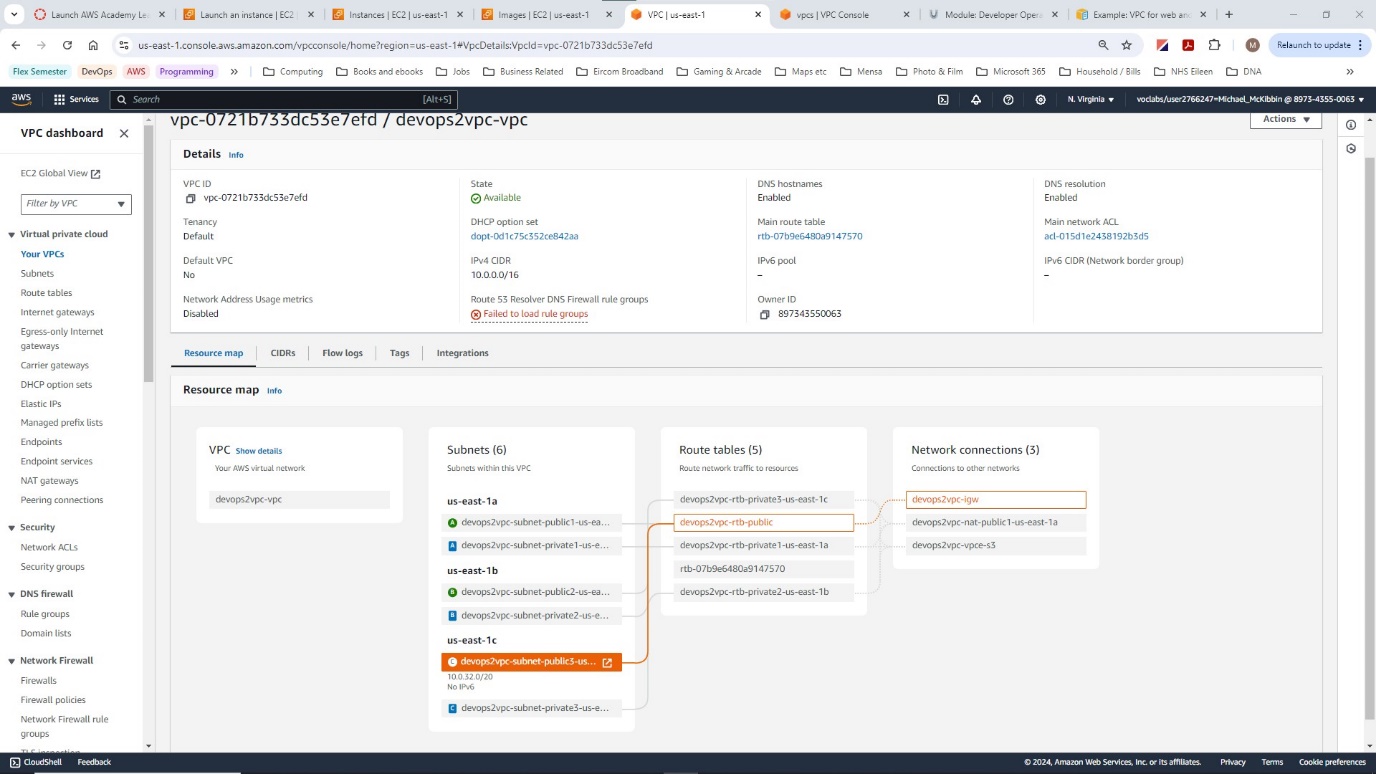


Figure VPC Resource Map with a route highlighted.

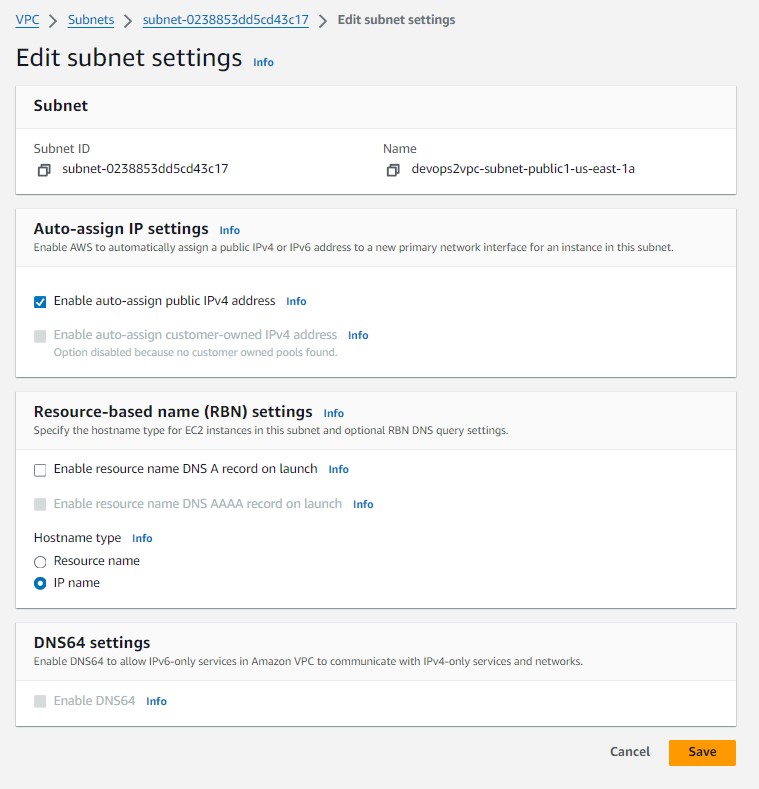
****

Figure Editing Subnet Settings.

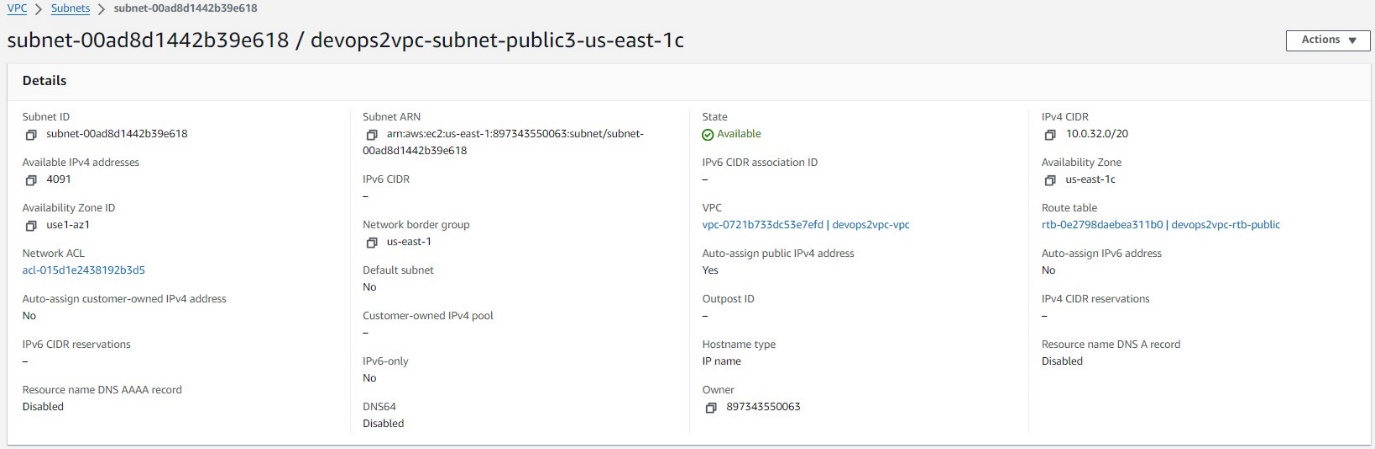
****

Figure Set IP address to Auto Assignment

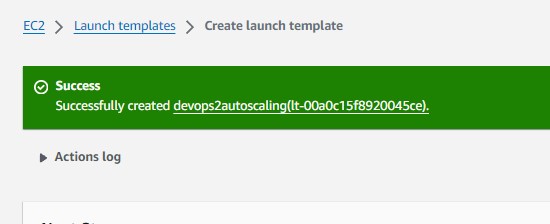
****

Figure Launch Template Success.

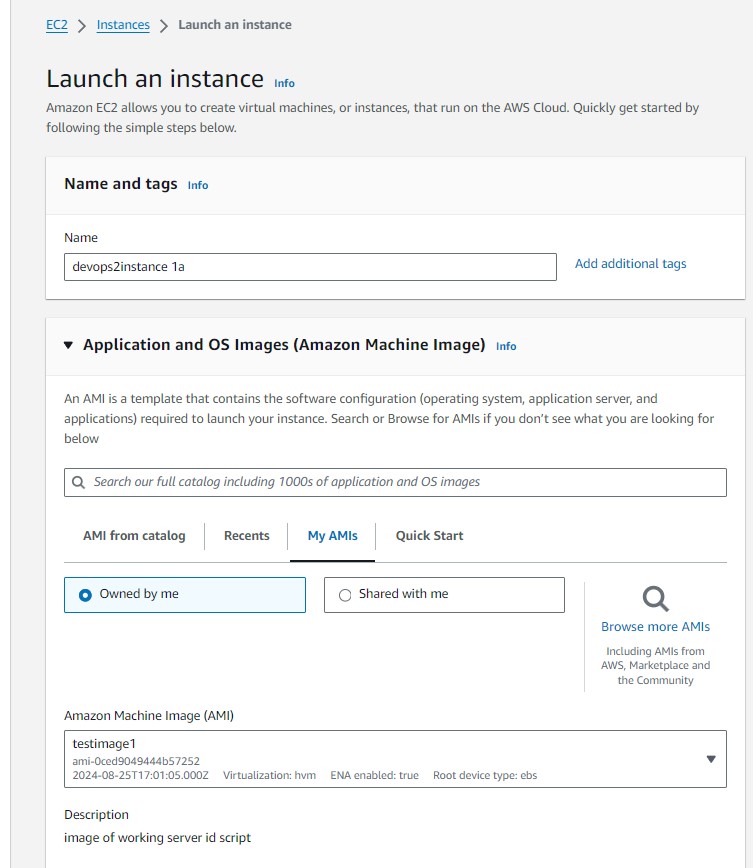
****

Figure Launching an Instance - part 1 – choosing an AMI.

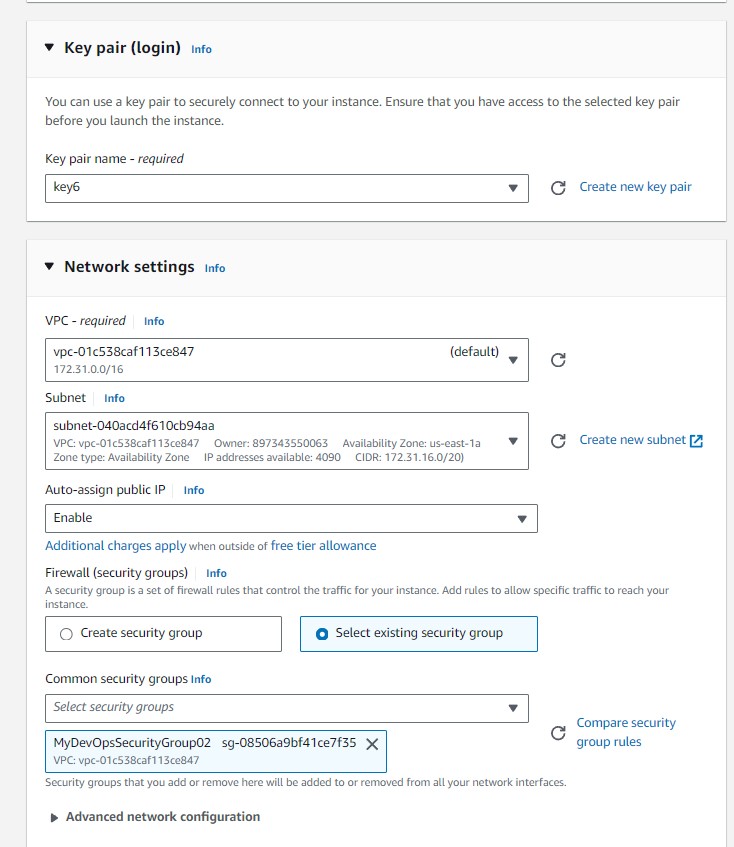
****

Figure Launching an Instance - part 2 – Key Pair, Network Settings (Subnets), and Security Group.

A screenshot of a computer

Description automatically generated

Figure Launching an Instance - part 3 - adding a script.

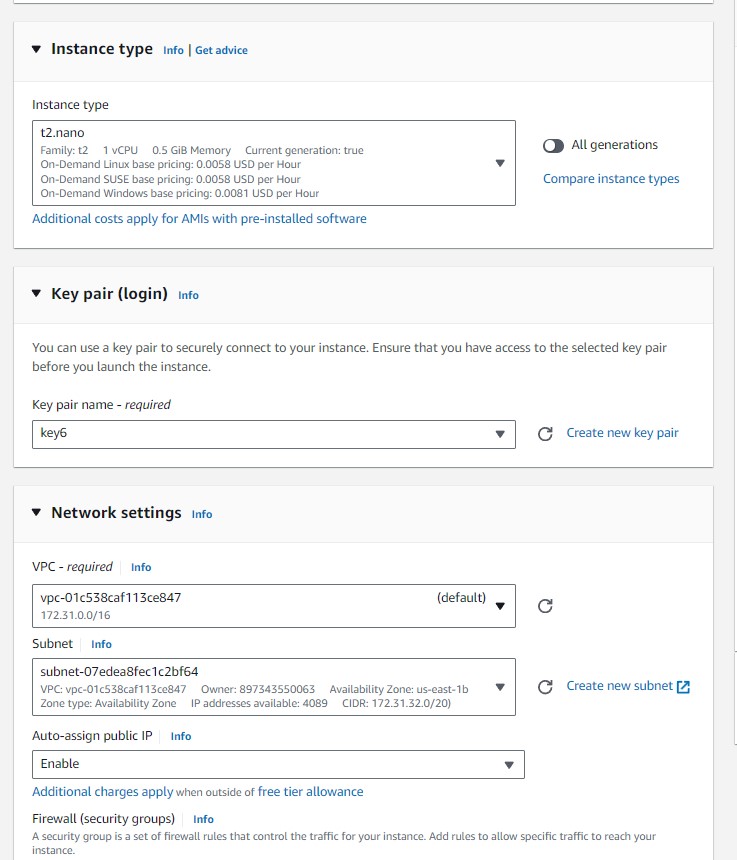


Figure Launch Instance with key pair and Network settings.

**Auto-scaling implementation and demonstration of scaling activity based on CloudWatch alarm.**

**A screenshot of a computer

Description automatically generated**

Figure Before Target Group setup.

A screenshot of a computer

Description automatically generated

Figure Load balancing Target Group.

**A screenshot of a computer

Description automatically generated**

Figure Load Balancer created successfully.

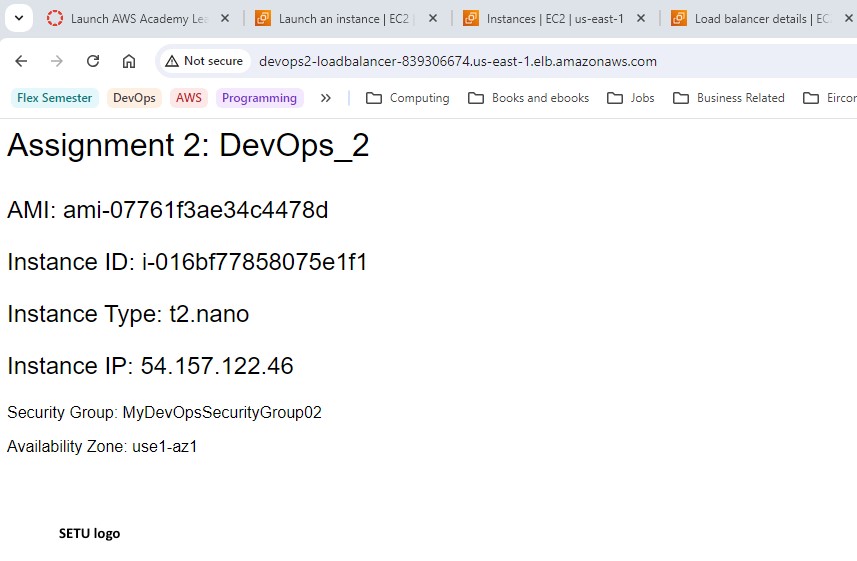


Figure Load Balancer details in browser.

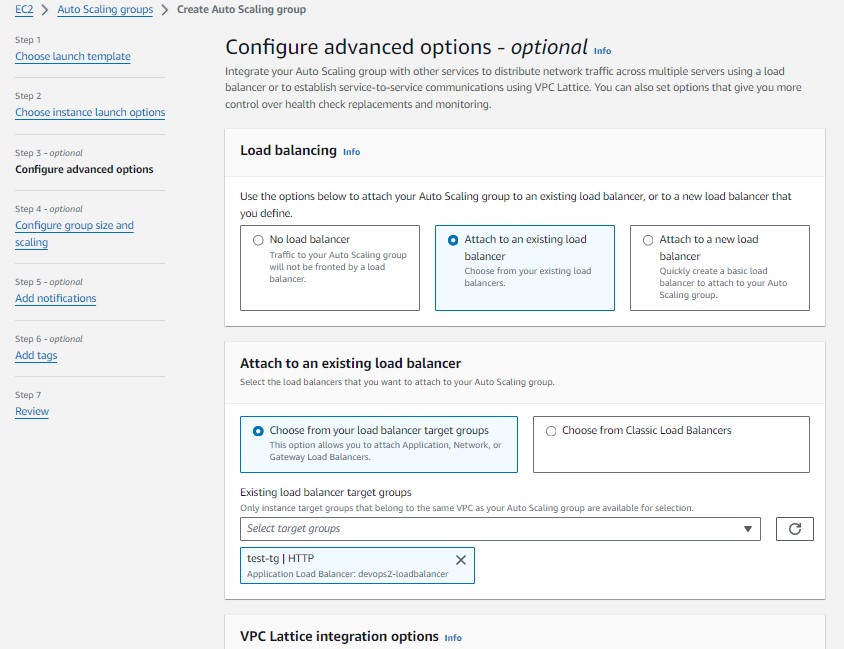
****

Figure Creating an Auto Scaling group.

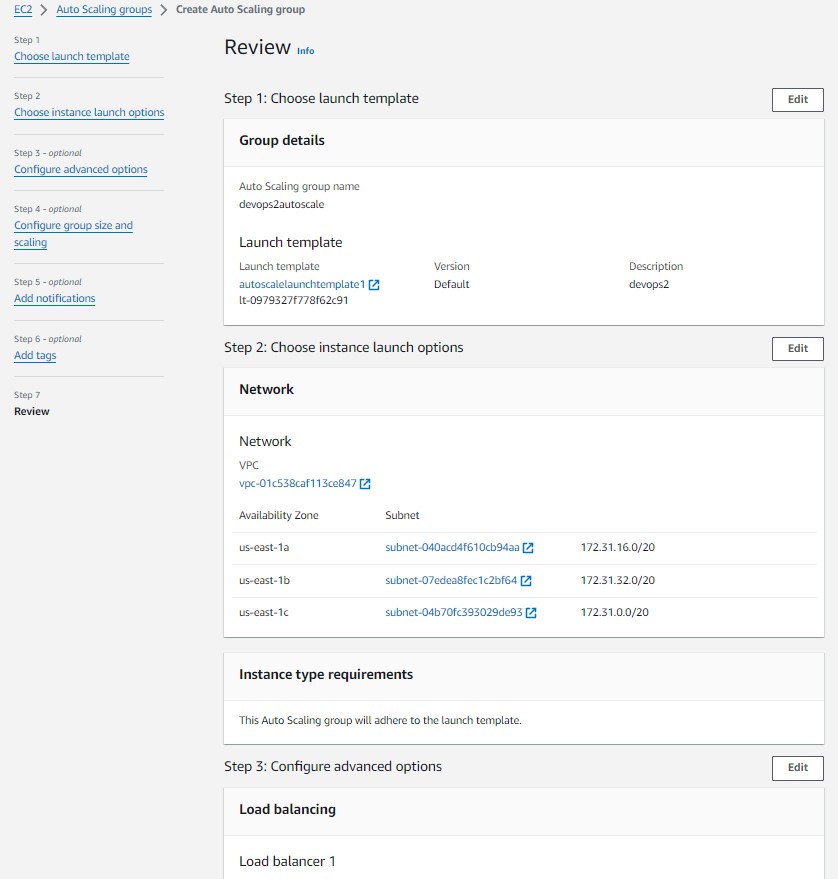
****

Figure Auto Scaling - Choosing a Launch Template and Network.

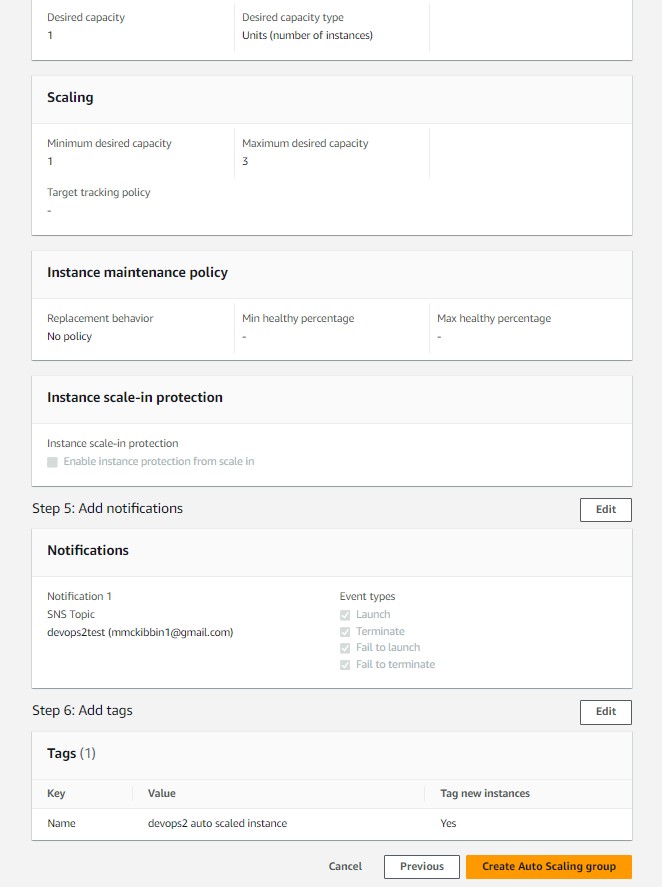
****

Figure Auto Scaling -Maximum instances, SNS Notifications, and Tags.

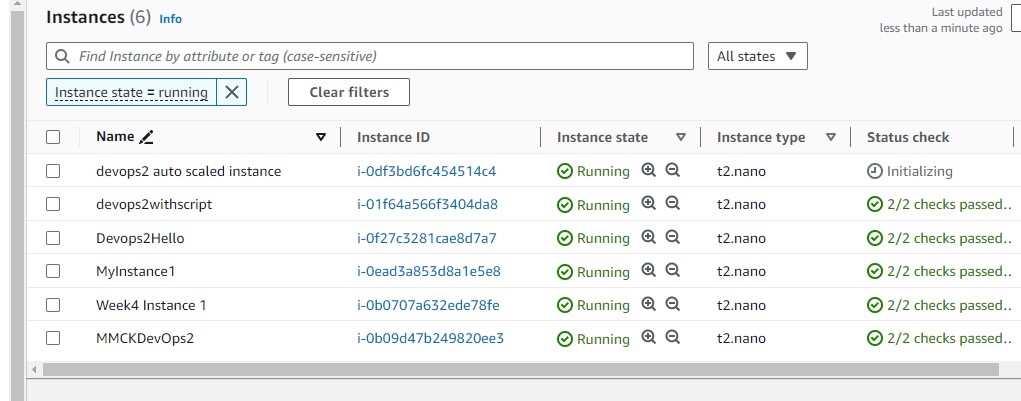


Figure Auto Scaling - First Auto Scaled Instance.

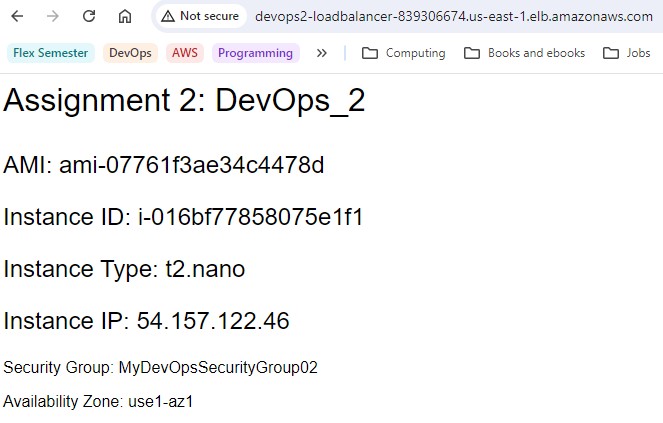


Figure Auto Scaling - First Instance Running.

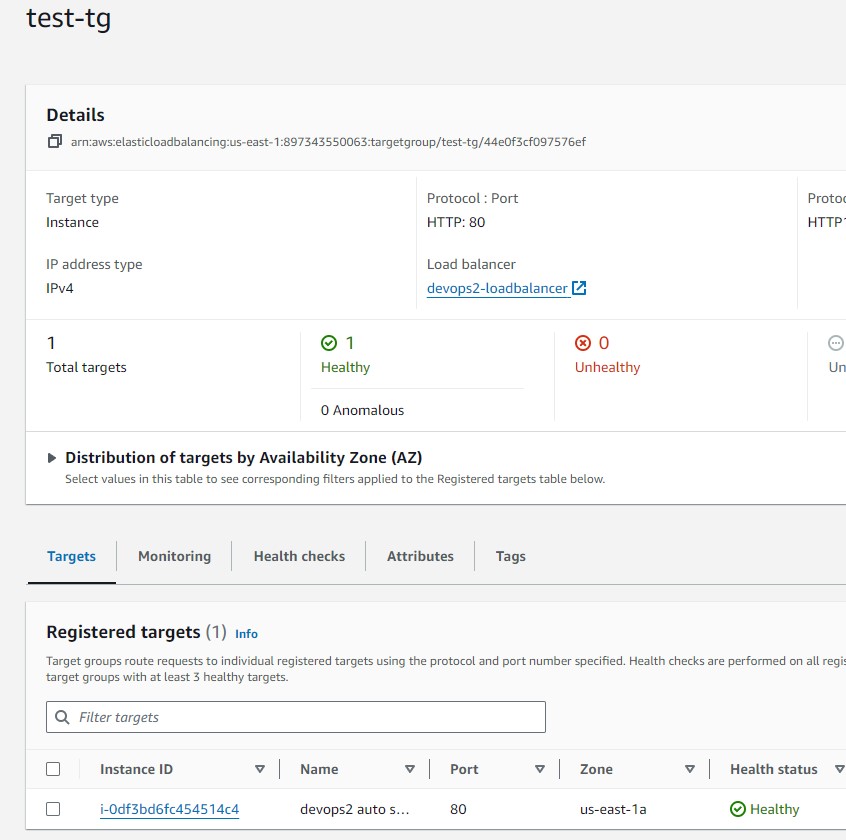


Figure Auto Scaling - Targets registered.

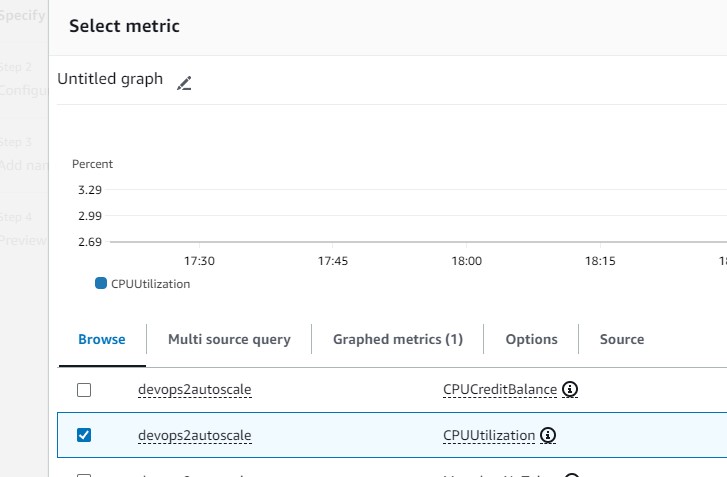


Figure Auto Scaling - Cloudwatch Metrics.

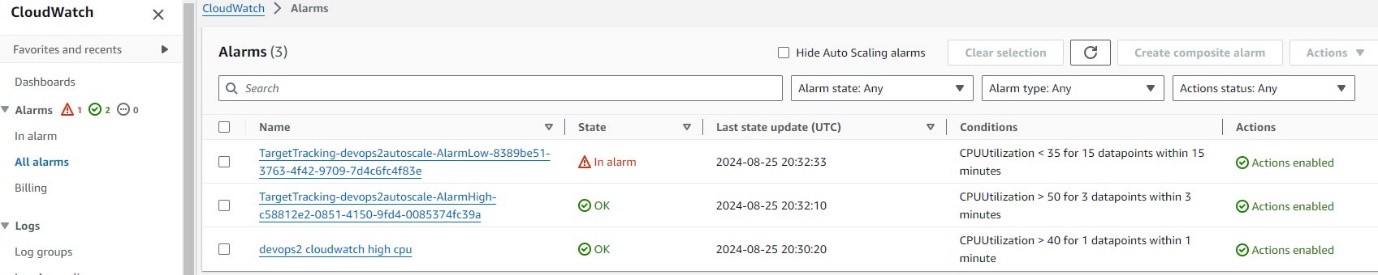


Figure Auto Scaling - Cloudwatch Alarms in progress.

### Testing: Test traffic and load balancing demonstration.

Generation of test traffic to the load balancer.

Using an infinite loop over SSH.

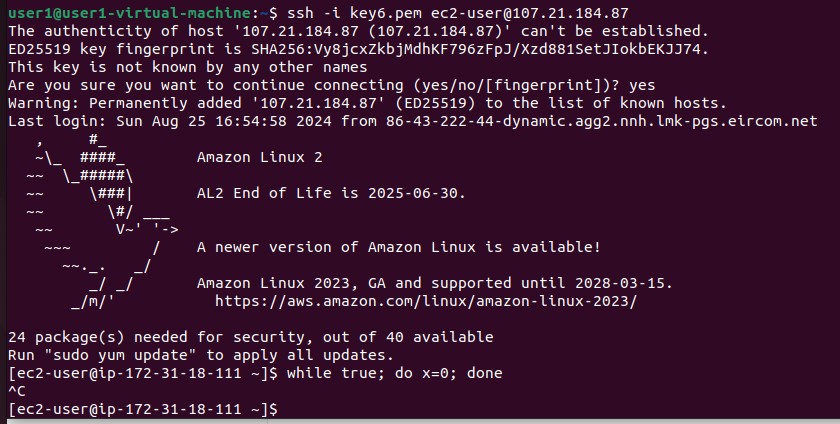


Figure Auto Scaling - Using an infinite loop to create traffic.

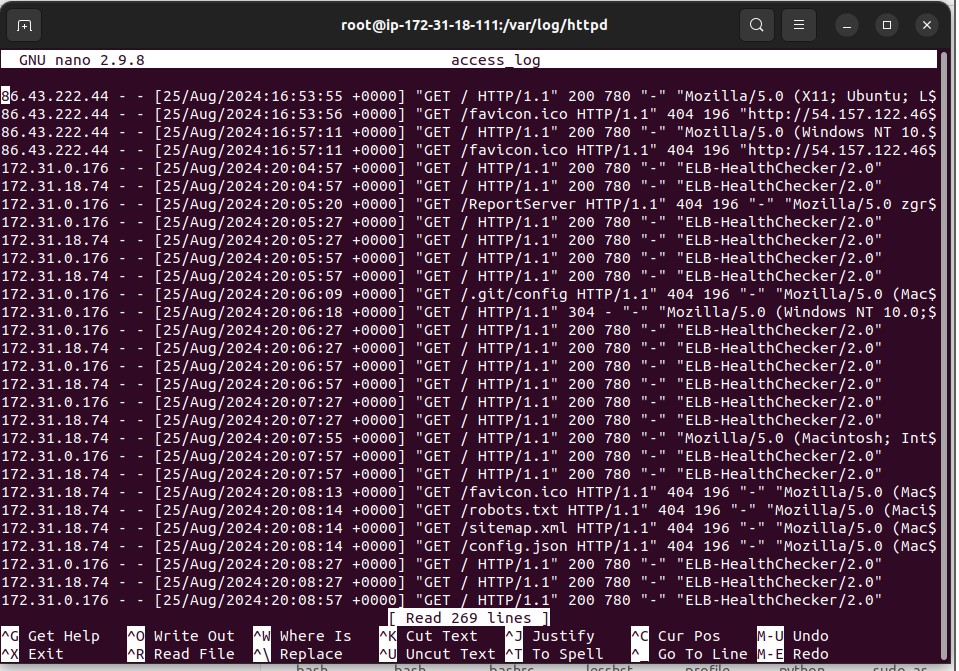


Figure Curl command generates 100 requests.

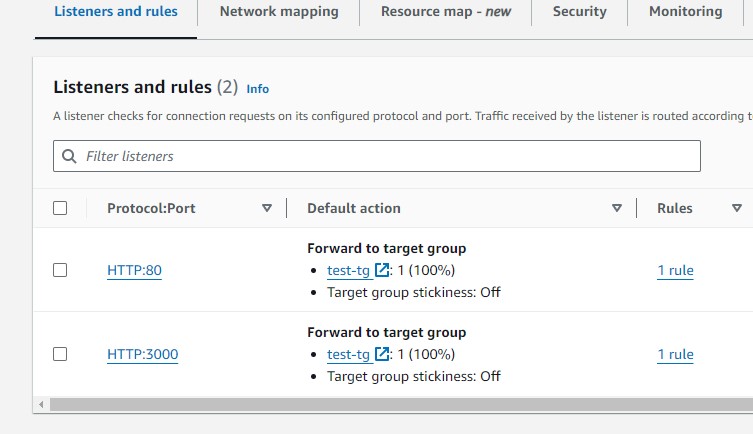


Figure Adding Listeners on port 80 and port 3000.



Figure Auto Scaled instances running.

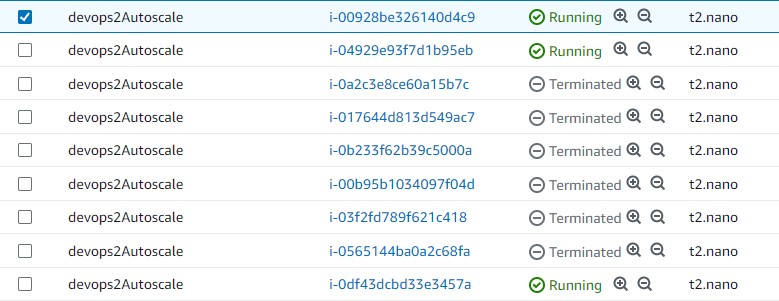


Figure Auto Scaled instances - Running and Terminated.

### Monitoring: Own script.

I used my own python program, devops\_2.py, adapted from assignment 1, to launch an instance and set up Cloudwatch alarms. I also used it to load and run further scripts, monitor.sh, and memv2.sh, to monitor and report on the custom metrics on the servers and push these to CloudWatch.

### APPENDIX A: Automated instance setup.

Including launching and configuring instance based on AMI.

Setting Key Pair and Security group.

Create webpage with content and display in a browser window.

Get instance details.

Install and run monitoring scripts – see Appendices B, and C.

Install Node.js, copy app.js, and run app in a browser window.

Set up Cloudwatch Alarms

import boto3

import time

import webbrowser

import random

import string

import urllib

import os

from cgitb import html

from ipaddress import ip\_address

from os import system

from urllib import response

from botocore.exceptions import ClientError

import json

# Launch New EC2 Instance

# Configure instance settings

# Set up EC2 website

# Download and install Apache web server

# Create index.html and add content

# Get meta data

# Get image

# copy index.html to local drive

ec2 = boto3.resource("ec2")

try:

    print("\nCreating new EC2 instance\n")

    new\_ec2\_instance = ec2.create\_instances(

        # Amazon Linux2 AMI (check current available AMIs before assignment submission)

        ImageId="ami-0ced9049444b57252",

        # How many instances to launch. Min and Max

        MinCount=1,

        MaxCount=3,

        # Instance type (t2.nano)

        InstanceType="t2.nano",

        # Tag the instance

        TagSpecifications=[

            {

                "ResourceType": "instance",

                "Tags": [

                    {"Key": "Name", "Value": "devops2Autoscale"},

                    {"Key": "Owner", "Value": "mmckibbin"},

                ],

            },

        ],

        # MyDevOpsSecurityGroup01 Security group ID

        SecurityGroupIds=["sg-08506a9bf41ce7f35"],

        KeyName="key6",

        UserData="""#!/bin/bash

            yum install httpd -y

            systemctl enable httpd

            systemctl start httpd

            echo "Content-type: text/html"

            echo '<html>' > index.html

            echo '<head>' >> index.html

            echo '<title>DevOps Assignment 1</title>' >> index.html

            echo '</head>' >> index.html

            echo '<body>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif; font-size:200%;">Assignment 2: DevOps\_2</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif; font-size:150%;">AMI: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/ami-id) >> index.html

            echo '</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif; font-size:150%;">Instance ID: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/instance-id) >> index.html

            echo '</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif; font-size:150%;">Instance Type: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/instance-type) >> index.html

            echo '</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif; font-size:150%;">Instance IP: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/public-ipv4) >> index.html

            echo '</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif;">Security Group: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/security-groups) >> index.html

            echo '</p>' >> index.html

            echo '<p style="font-family:Helvetica, sans-serif;">Availability Zone: ' >> index.html

            echo $(curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id ) >> index.html

            echo '</p>' >> index.html

            echo '<img src="http://devops.witdemo.net/logo.jpg"> logo.jpg ' >> index.html

            echo '</body>' >> index.html

            echo '</html>' >> index.html

            cp index.html /var/www/html/index.html

        """,

    )

except Exception as e:

    print("Error! \nThe EC2 creation process has encountered an error.\n")

    print(e)

    errorfile = open("error.log", "w")

    errorfile.write(str(e))

    errorfile.close()

    print("See error.log for details.")

else:

    # Print instance ID, type, & state

    print(

        "\nNew EC2 instance created successfully!"

        + "\n[ID: "

        + new\_ec2\_instance[0].id

        + "]"

        + "\n[Type: "

        + new\_ec2\_instance[0].instance\_type

        + "]"

        +

        #'\n[Region: ' + new\_ec2\_instance[0].region['Name'] + ']' +

        "\n[Current state: "

        + new\_ec2\_instance[0].state["Name"]

        + "]"

    )

    # Check instance state every 5 seconds.

    print("\nWaiting for instance to run...")

    while new\_ec2\_instance[0].state["Name"] != "running":

        time.sleep(5)

        new\_ec2\_instance[0].reload()

    # Print instance state & public ip address

    print(

        "\n\n[Current state: "

        + new\_ec2\_instance[0].state["Name"]

        + "]\n"

        + "[Public IP: "

        + new\_ec2\_instance[0].public\_ip\_address

        + "]\n"

    )

    # Wait x seconds for webserver to initialise

    print("\n\nAllowing time for web server to initialise...")

    time.sleep(60)

    print("\nOpening webpage at: " + new\_ec2\_instance[0].public\_ip\_address)

    print("\n\n")

    ip\_address = new\_ec2\_instance[

        0

    ].public\_ip\_address  # Set variable to instance public IP

    webbrowser.open(

        "http://" + new\_ec2\_instance[0].public\_ip\_address

    )  # Open web browser to instance public IP

    # 6. Monitoring

    # Out of sequence numerically as connection timed out when running later on, after s3 setup (Sections 4 & 5)

try:

    time.sleep(30)  # wait a bit...

    # set keypair permissions for ssh access

    print("\nSet keypair permission")

    system("chmod 400 key6.pem")

    print("\nDone.")

    print("\n")

    # copy monitoring scripts to instance, run them.

    print("\nCopying monitor.sh to ec2 instance")

    system(

        f"scp -o StrictHostKeyChecking=no -i key6.pem monitor.sh ec2-user@{ip\_address}:."

    )

    print("\nDone.")

    print("\n")

    print("\nSet permissions on monitor.sh")

    system(f"ssh -i key6.pem ec2-user@{ip\_address} 'chmod 700 monitor.sh'")

    print("\nDone.")

    print("\n")

    print("\nRun monitor.sh (on ec2 instance)")

    system(f"ssh -i key6.pem ec2-user@{ip\_address} './monitor.sh'")

    print("\nend of monitoring script")

    print("\n")

    print("\nCopying memv2.sh to ec2 instance")

    system(

        f"scp -o StrictHostKeyChecking=no -i key6.pem memv2.sh ec2-user@{ip\_address}:."

    )

    print("\nDone.")

    print("\n")

    print("\nSet permissions on memv2.sh")

    system(f"ssh -i key6.pem ec2-user@{ip\_address} 'chmod 700 memv2.sh'")

    print("\nDone.")

    print("\n")

    print("\nRun memv2.sh (on ec2 instance)")

    system(f"ssh -i key6.pem ec2-user@{ip\_address} './memv2.sh'")

    print("\nend of memv2 script")

    print("\n")

    # list files in instance

    print("\nList files in instance" + new\_ec2\_instance[0].id + "...")

    system(f"ssh -i key6.pem ec2-user@{ip\_address} 'ls -l'")

    print("\nDone.")

    print("\n")

    # # copy install-node.sh and run it

    # print("\nCopying install-node.sh to ec2 instance")

    # system(

    #     f"scp -o StrictHostKeyChecking=no -i key6.pem install-node.sh ec2-user@{ip\_address}:."

    # )

    # print("\nSet permissions on install-node.sh")

    # system(f"ssh -i key6.pem ec2-user@{ip\_address} 'chmod 700 install-node.sh'")

    # print("\nRun install-node.sh")

    # system(f"ssh -i key6.pem ec2-user@{ip\_address} './install-node.sh'")

    # print("\nend of install-node.sh script")

    # print("\n")

#     # install Node.js

#     print("\nInstalling Node.js...")

#     system("curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash")

#     system("source ~/.nvm/nvm.sh")

#     system("nvm install 16")

#     system("node -v")

#     print("\nDone.")

except Exception as e:

    print(e)

# Upload the app.js file to ec2 instance

file\_path = "app.js"

print("\nUploading " + file\_path + " to ec2 instance...")

system(

    f"scp -o StrictHostKeyChecking=no -i key6.pem {file\_path} ec2-user@{ip\_address}:."

)

print("\nDone.")

print("now connect via SSH and install node, then run app")

time.sleep(10)

# # run app.js

# print("\nRunning app.js...")

# system("node app.js")

# print("\nDone.")

# # open browser window to ec2 instance

# print("\nOpening browser window to ec2 instance...")

# print("\nOpening webpage at: " + new\_ec2\_instance[0].public\_ip\_address)

# print("\n\n")

# ip\_address = new\_ec2\_instance[

#     0

# ].public\_ip\_address  # Set variable to instance public IP

# webbrowser.open(

#     "http://" + new\_ec2\_instance[0].public\_ip\_address +":3000"

# )  # Open web browser to instance public IP

# # install Node.js

# print("\nInstalling Node.js...")

# system("curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash -")

# system("source ~/.nvm/nvm.sh")

# system("nvm install 16")

# #export nvm dir to bashrc

# system("export NVM\_DIR=$HOME/.nvm")

# system("[ -s \"$NVM\_DIR/nvm.sh\" ] && \. \"$NVM\_DIR/nvm.sh\"")

# system("[ -s \"$NVM\_DIR/bash\_completion\" ] && \. \"$NVM\_DIR/bash\_completion\"")

# system("nvm use 16")

# print("\ninstalling npm")

# system("npm install -g npm@latest")

# # get node version number

# print("\nNode version:")

# system("node -v")

# print("\nDone.")

# # Upload the app.js file

# file\_path = "app.js"

# if os.path.exists(file\_path):

#     try:

#         s3\_client.upload\_file(

#             file\_path, bucket\_name, "app.js", ExtraArgs={"ContentType": "text/javascript"}

#         )

#         print(f"File {file\_path} uploaded as app.js.")

#     except ClientError as e:

#         print(f"Error uploading {file\_path}: {e}")

#         exit(1)

# else:

#     print(f"File {file\_path} does not exist.")

#     exit(1)

# # run app.js

# print("\nRunning app.js...")

# system("node app.js")

#CloudWatch alarms setup

print("Setting up CloudWatch alarms...")

cloudwatch = boto3.client("cloudwatch")

# CPU utilization greater than 50%

try:

    cloudwatch.put\_metric\_alarm(

        AlarmName="HighCPUUtilization",

        ComparisonOperator="GreaterThanThreshold",

        EvaluationPeriods=1,

        MetricName="CPUUtilization",

        Namespace="AWS/EC2",

        Period=60,

        Statistic="Average",

        Threshold=50.0,

        ActionsEnabled=True,

        AlarmActions=[

            "arn:aws:automate:us-east-1:ec2:reboot",

        ],

        AlarmDescription="Alarm if server CPU utilization exceeds 50%",

        Dimensions=[

            {"Name": "InstanceId", "Value": new\_ec2\_instance[0].id},

        ],

        Unit="Percent",

    )

    print("High CPU utilization alarm created.")

except ClientError as e:

    print(f"Error creating high CPU utilization alarm: {e}")

# CPU utilization less than 30%

try:

    cloudwatch.put\_metric\_alarm(

        AlarmName="LowCPUUtilization",

        ComparisonOperator="LessThanThreshold",

        EvaluationPeriods=1,

        MetricName="CPUUtilization",

        Namespace="AWS/EC2",

        Period=60,

        Statistic="Average",

        Threshold=30.0,

        ActionsEnabled=True,

        AlarmActions=[

            "arn:aws:automate:us-east-1:ec2:terminate",

        ],

        AlarmDescription="Alarm if server CPU utilization below 30%",

        Dimensions=[

            {"Name": "InstanceId", "Value": new\_ec2\_instance[0].id},

        ],

        Unit="Percent",

    )

    print("Low CPU utilization alarm created.")

except ClientError as e:

    print(f"Error creating low CPU utilization alarm: {e}")

try:

    # Describe alarms

    print("\nDescribing alarms...")

    response = cloudwatch.describe\_alarms()

    for alarm in response["MetricAlarms"]:

        print(f"Alarm Name: {alarm['AlarmName']}")

        print(f"Alarm Description: {alarm['AlarmDescription']}")

        print(f"Alarm State: {alarm['StateValue']}")

        print(f"Alarm Actions: {alarm['AlarmActions']}")

        print(f"Alarm Comparison: {alarm['ComparisonOperator']}")

        print(f"Evaluation Periods: {alarm['EvaluationPeriods']}")

        print(f"Metric Name: {alarm['MetricName']}")

        print(f"Namespace: {alarm['Namespace']}")

        print(f"Period: {alarm['Period']}")

        print(f"Statistic: {alarm['Statistic']}")

        print(f"Threshold: {alarm['Threshold']}")

        print(f"Unit: {alarm['Unit']}")

        print("\n")

except ClientError as e:

    print(f"Error describing alarms: {e}")

# print("\n\nexiting...")

# time.sleep(1)

exit()

### APPENDIX B: monitor.sh - Instance info.

#!/usr/bin/bash

# Michael MCKibbin 20092733

# Adapted from course provided script.

#

INSTANCE\_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)

MEMORYUSAGE=$(free -m | awk 'NR==2{printf "%.2f%%", $3\*100/$2 }')

PROCESSES=$(expr $(ps -A | grep -c .) - 1)

HTTPD\_STATUS=$(systemctl status $1 | awk 'NR == 3')

HTTPD\_UPTIME=$(uptime | awk '{print $3,$4}' | cut -d, -f1)

HTTPD\_PROCESSES=$(ps -A | grep -c httpd)

APACHE\_PROCESSES=$(ps -A | grep -c apache)

APACHE\_PROCESSES\_NoGREP=$(ps -ef | grep -v grep | grep -c apache)

HTTP\_PORT=$(netstat -tuln | grep -c ":80")

HTTPS\_PORT=$(netstat -tuln | grep -c ":443")

AVAILABILITY\_ZONE=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone)

AMI\_ID=$(curl -s http://169.254.169.254/latest/meta-data/ami-id)

SECURITY\_GROUP=$(curl -s http://169.254.169.254/latest/meta-data/security-groups)

IPV4=$(curl -s http://169.254.169.254/latest/meta-data/public-ipv4)

SSH\_PORT=$(netstat -tuln | grep -c ":22")

echo "Instance ID: $INSTANCE\_ID"

echo "Memory utilisation: $MEMORYUSAGE"

if [ $HTTPD\_PROCESSES -ge 1 ]

then

    echo "Web server is running"

else

    echo "Web server is NOT running"

fi

echo "No of HTTPD processes: $PROCESSES"

echo "HTTPD server status: $HTTPD\_STATUS"

echo "HTTPD server uptime: $HTTPD\_UPTIME"

echo "No of Apache processes: $APACHE\_PROCESSES"

echo "No of Apache processes (No grep): $APACHE\_PROCESSES\_NoGREP"

echo "No of SSH connections: $SSH\_PORT"

echo "No of HTTP connections: $HTTP\_PORT"

echo "No of HTTPS connections: $HTTPS\_PORT"

echo "Availability zone: $AVAILABILITY\_ZONE"

echo "AMI ID: $AMI\_ID"

echo "Security group: $SECURITY\_GROUP"

echo "Public IPv4: $IPV4"

### APPENDIX C: memv2.sh - Cloudwatch.

#!/bin/bash

TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`

INSTANCE\_ID=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/instance-id)

#INSTANCE\_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)

USEDMEMORY=$(free -m | awk 'NR==2{printf "%.2f\t", $3\*100/$2 }')

TCP\_CONN=$(netstat -an | wc -l)

TCP\_CONN\_PORT\_80=$(netstat -an | grep 80 | wc -l)

IO\_WAIT=$(iostat | awk 'NR==4 {print $5}')

aws cloudwatch put-metric-data --metric-name memory-usage --dimensions Instance=$INSTANCE\_ID --namespace "Custom" --value $USEDMEMORY

aws cloudwatch put-metric-data --metric-name Tcp\_connections --dimensions Instance=$INSTANCE\_ID --namespace "Custom" --value $TCP\_CONN

aws cloudwatch put-metric-data --metric-name TCP\_connection\_on\_port\_80 --dimensions Instance=$INSTANCE\_ID --namespace "Custom" --value $TCP\_CONN\_PORT\_80

aws cloudwatch put-metric-data --metric-name IO\_WAIT --dimensions Instance=$INSTANCE\_ID --namespace "Custom" --value $IO\_WAIT