SWE-455: Cloud Application Engineering

Homework 01:

Deploying a simple cloud-based application

Name: AYED ALQAHTANI

ID: 202023400

This report outlines the deployment of a simple Node.js server on AWS EC2 instances. The server is designed to return a basic HTML response (<h1>Hello, User! </h1>) when accessed. Below, I provide an overview of the deployment process, including screenshots, challenges faced.

Deployment steps

Server code

We are going to make a simple NodeJS server that listens to http requests and shows (hello users!)

First, we I created a simple NodeJS server with simple endpoint

index.js > ...

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/html');
    res.end('<h1>Hello, User!</h1>');
});

const port = 80;
server.listen(port, () => {
    console.log(`Server running at port:${port}/`);
});
```

Then I created a git repository to access it later when we created the virtual machine.

Virtual machine code

package.json script.sh M

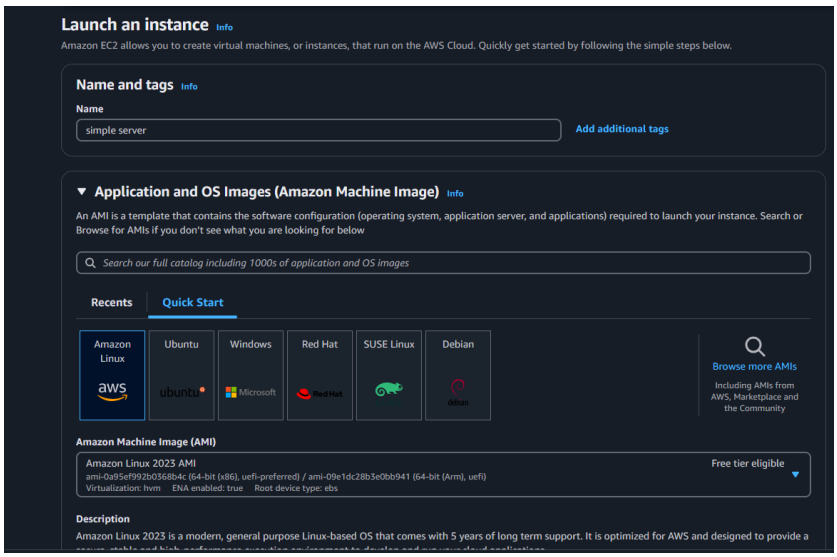
script.sh

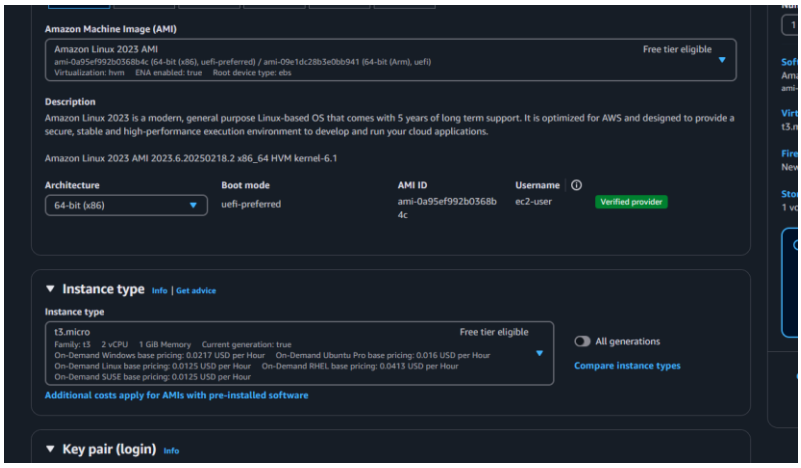
```
#!/bin/bash
# Update the system
yum update -y

# Install Node.js and npm
curl -sL https://rpm.nodesource.com/setup_16.x | bash -
yum install -y nodejs

# Install Git
yum install -y git

# Clone the Git repository
git clone https://github.com/ayed87/simple_server.git /home/ec2-user/myapp

# Navigate to the app directory
cd /home/ec2-user/myapp

# Install dependencies
npm install

# Start the Node.js server
node index.js &
```

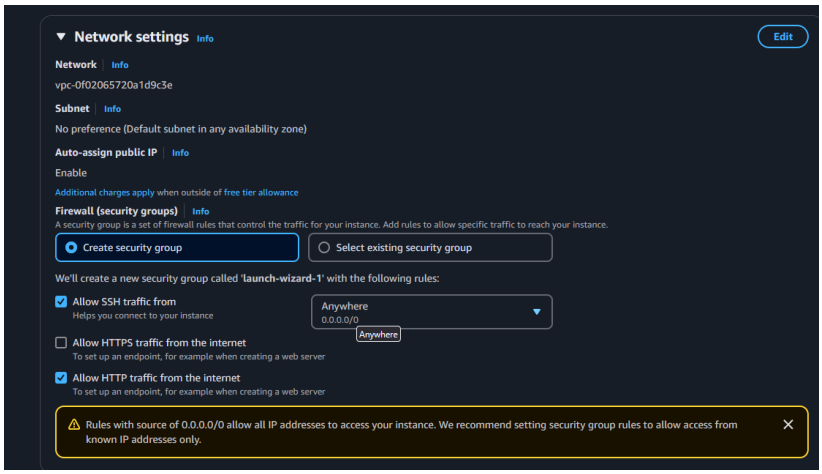
This code will be run when we create our virtual machine.

Create AWS EC2 virtual machine steps

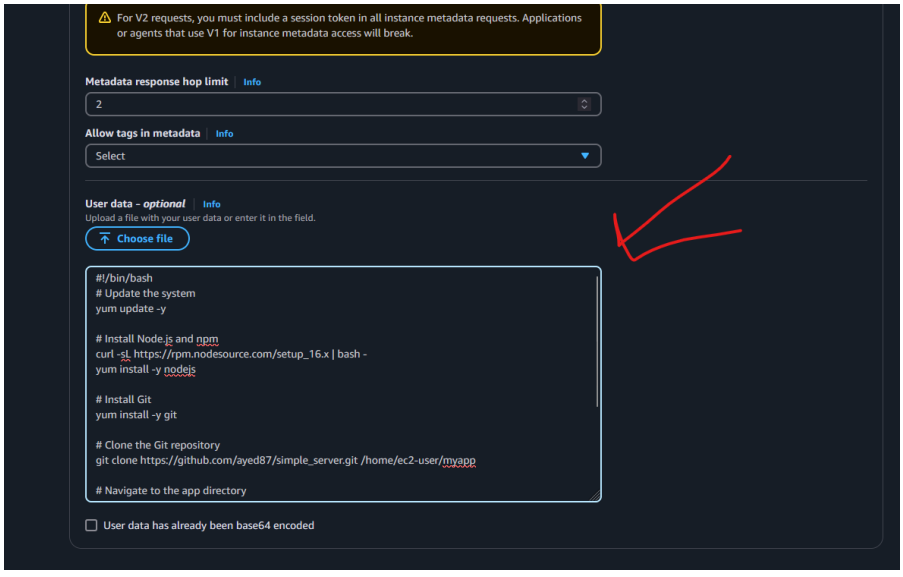
Configuring EC2 using AWS image



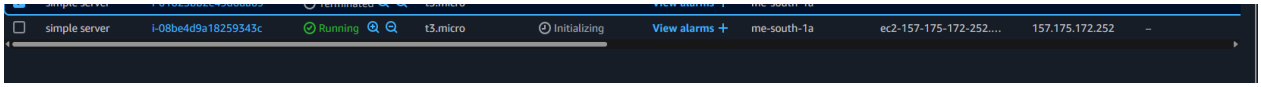




Here we should allow traffic from HTTP (80)



Here we add the script that we wrote before

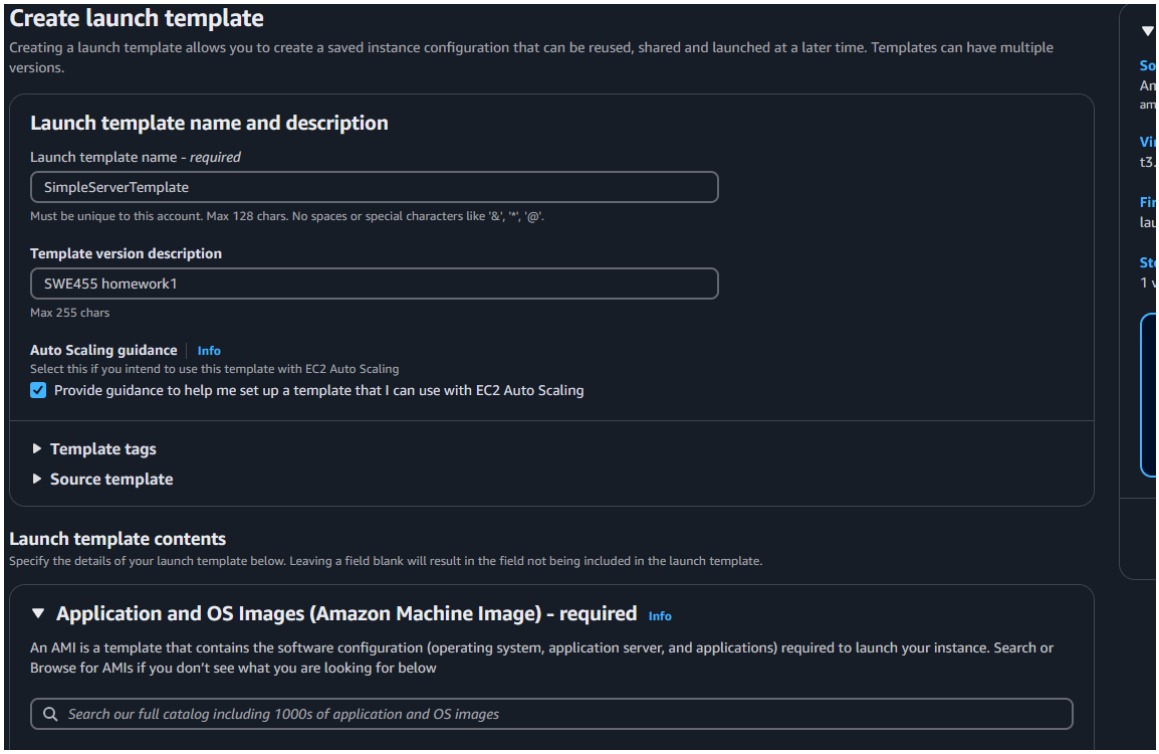


Now the server is running, and we can access through internet using its public IP address

Scalability

In order to make the EC2 scalable, we should create an autoscaling group

Configuring a template for our autoscaling group:



Choose launch template or configuration Info

Specify a launch template that contains settings common to all EC2 instances that are launched by this Auto Scaling group. If you currently use launch configurations, you might consider migrating to launch templates.

Name

Auto Scaling group name
Enter a name to identify the group.

example

Must be unique to this account in the current Region and no more than 255 characters.

Launch template Info

Switch to launch configuration

Launch template
Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.

SimpleServerTemplate

Create a launch template

Version

Default (1)

Create a launch template version

Description SWE455 homework1	**Launch template** SimpleServerTemplate lt-0942e5a88c9391ec4	**Instance type** t3.micro
AMI ID ami-0a95ef992b0368b4c	**Security groups** -	**Request Spot Instances** No
Key pair name -	**Security group IDs** sg-0ad9e7061aab6752c	

Additional details

Choose instance launch options Info

Choose the VPC network environment that your instances are launched into, and customize the instance types and purchase options.

Instance type requirements Info

Override launch template

You can keep the same instance attributes or instance type from your launch template, or you can choose to override the launch template by specifying different instance attributes or manually adding instance types.

Launch template SimpleServerTemplate lt-0942e5a88c9391ec4	**Version** Default	**Description** SWE455 homework1
Instance type t3.micro		

Network Info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.

vpc-0f02065720a1d9c3e
172.31.0.0/16 Default

Create a VPC

Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.

Select Availability Zones and subnets

me-south-1a | subnet-03af311432a792a0e
172.31.0.0/20 Default

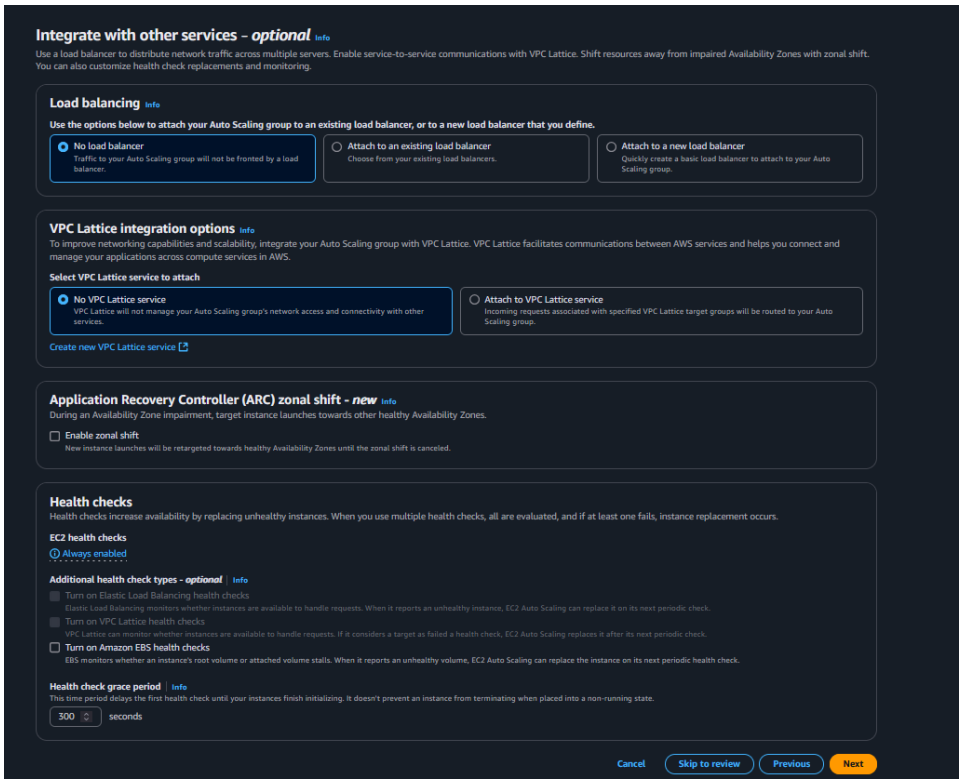
me-south-1b | subnet-01793a38dade31e3f
172.31.16.0/20 Default

me-south-1c | subnet-0d6334e4d73efd4f4
172.31.32.0/20 Default

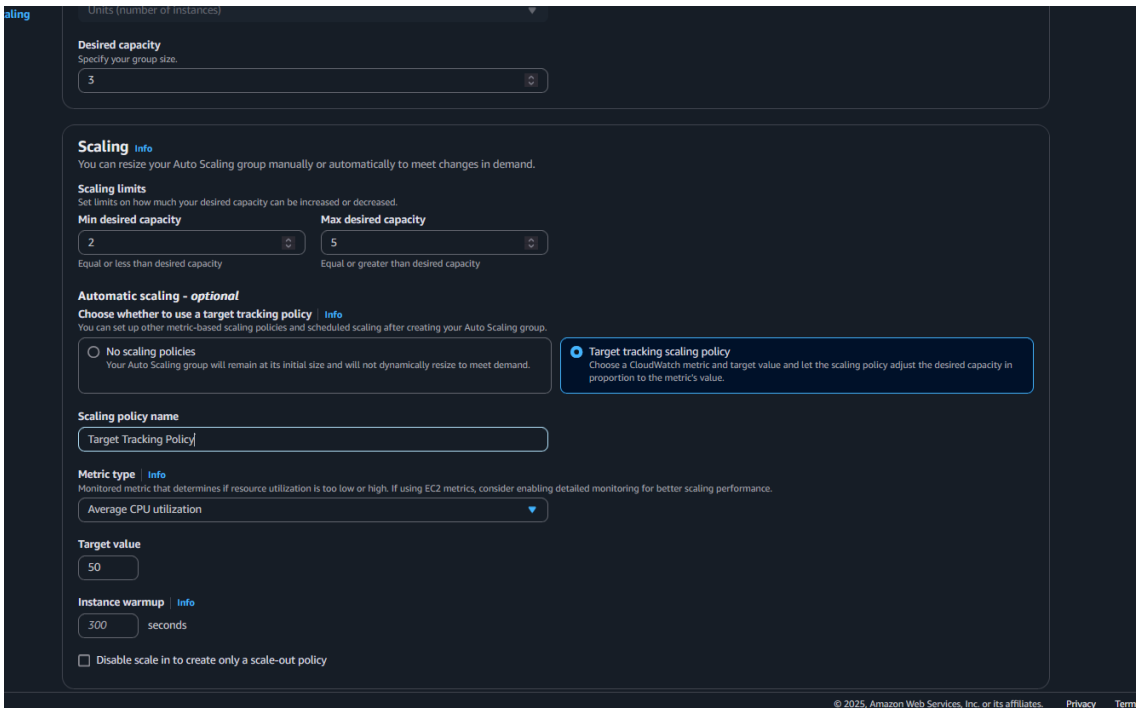
Create a subnet

Availability Zone distribution - *new*
Auto Scaling automatically balances instances across Availability Zones. If launch failures occur in a zone, select a strategy.

- (●) **Balanced best effort**
  If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.
- ( ) **Balanced only**
  If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve balanced distribution.

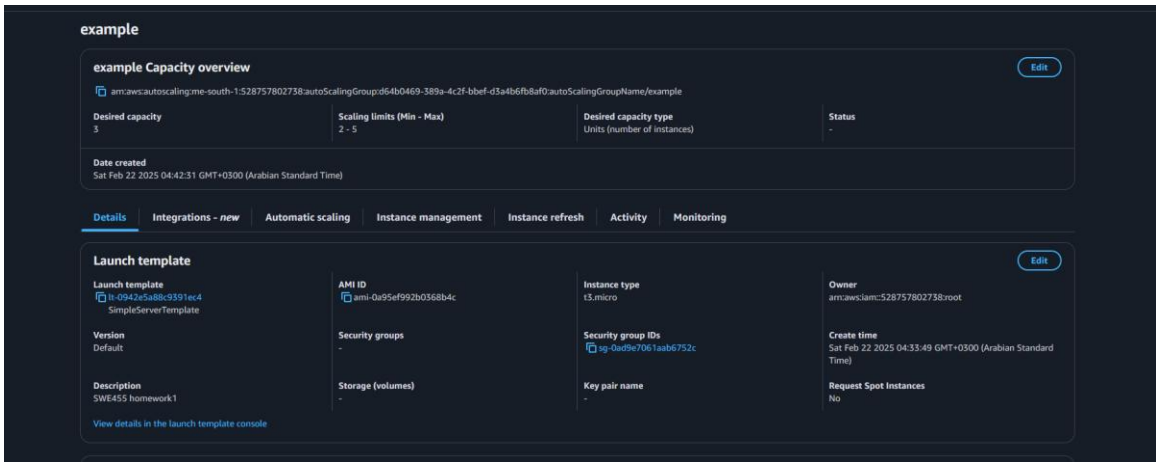


For simplicity we will configure only auto scaling groups without load balancer

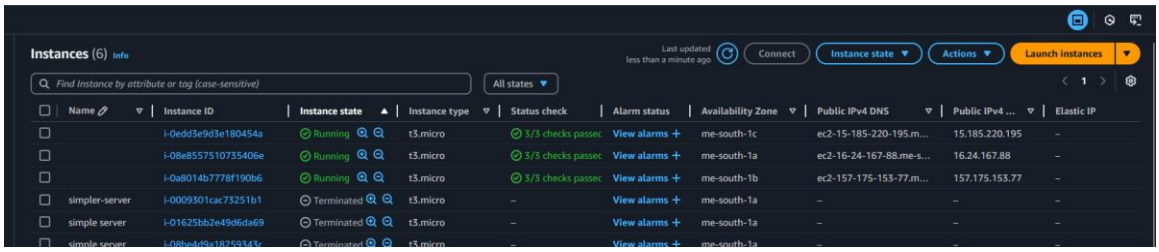


Here we define the minimum to 2 and the maximum to 5. And the desired (default) is 3

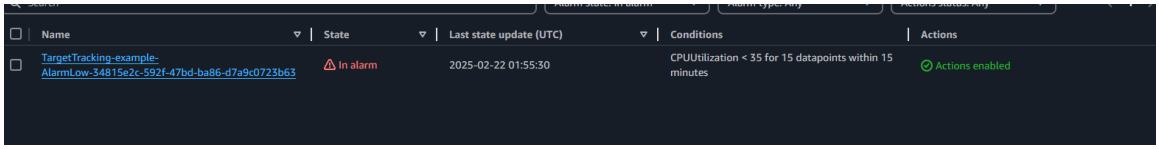
The template is created

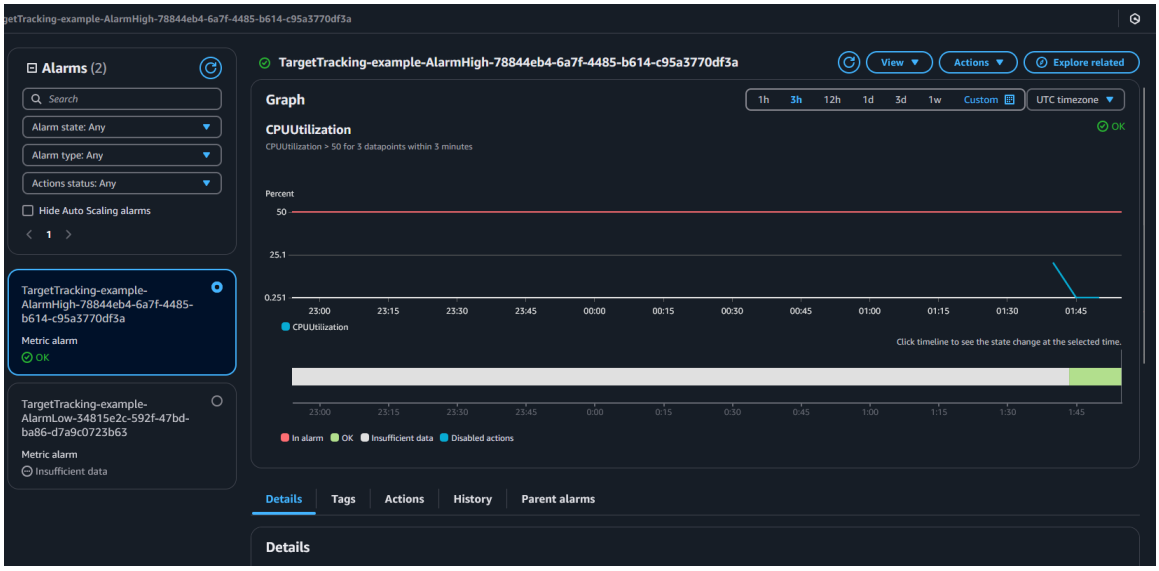


3 EC2s are running 3 which are the desired that we put

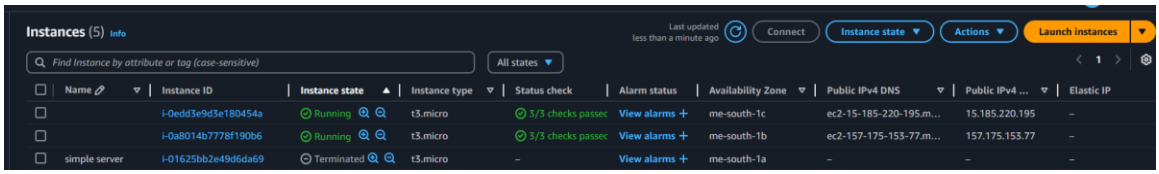


Testing scalability by CloudWatch

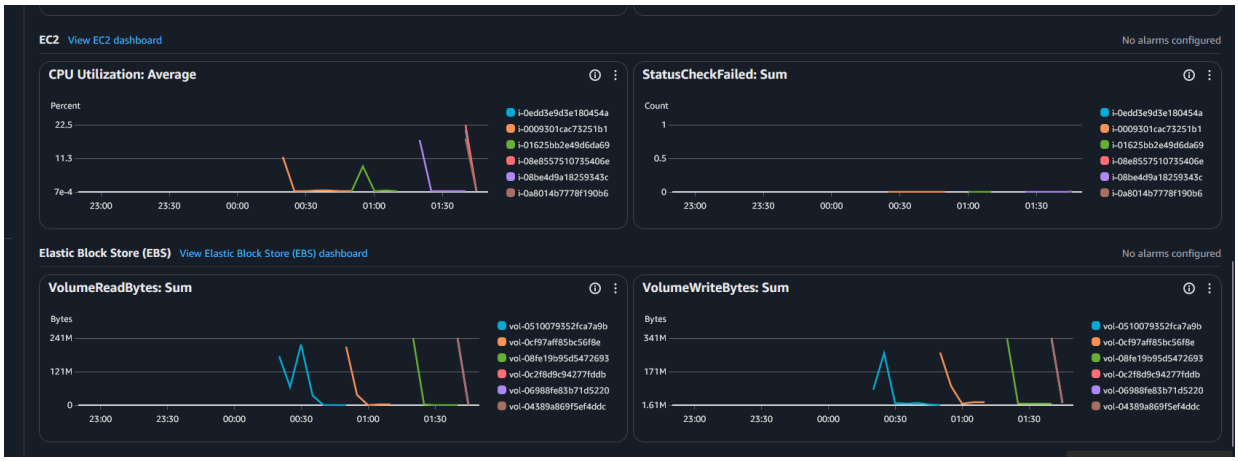


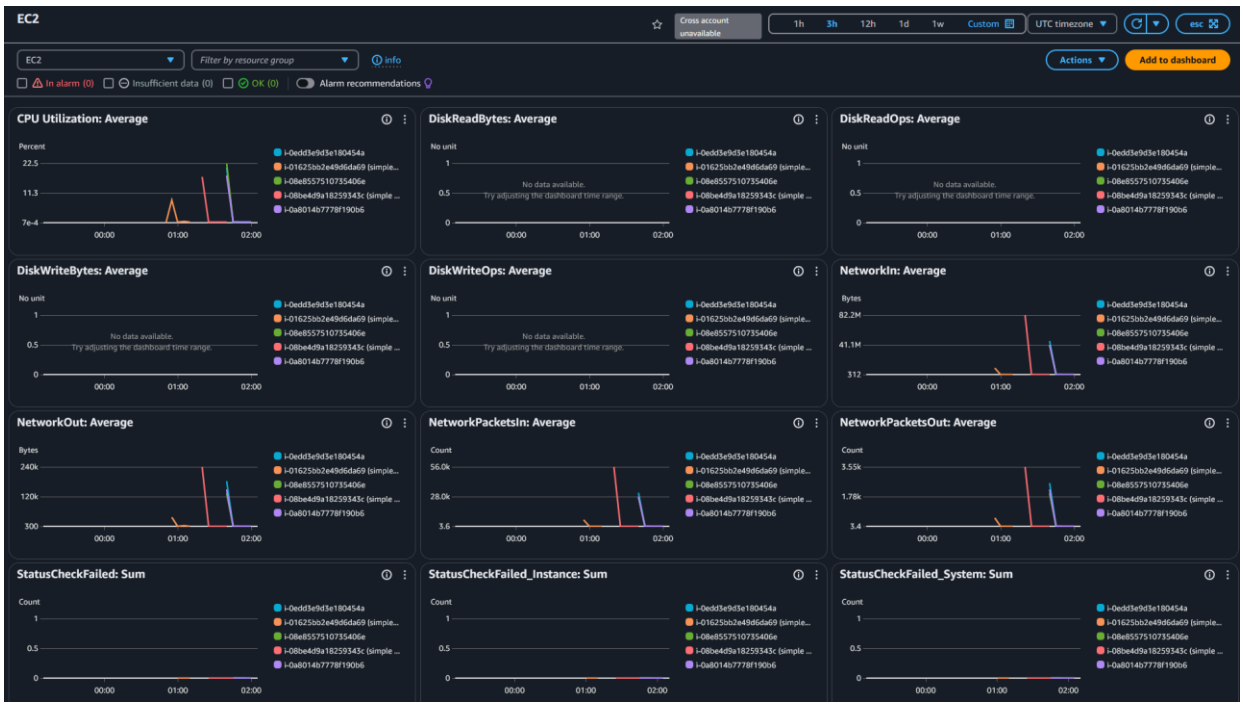


From the screenshots above the cloud watch catch that CPU usage is low. Hence it reduced the number of virtual machines to 2.



Other monitoring mastics in cloud watch dashboard





Challenges

Security Group Configuration:

Initially, the server was inaccessible due to the security group inbound role. It did not allow inbound traffic on port 80. This was resolved by updating the security group rules to accept traffic from port 80.

Port is not correctly configured in the server code:

Initially, the server code was listening to port 3000 which is a default port when developing the NodeJS server. I fixed this issue by changing it to 80.