COS20019 – CLOUD COMPUTING ARCHITECTURE

Name: Nguyen Linh Dan

Student ID: 103488557

ACA MODULE 9 CHALLENGE LAB

CREATING A SCALABLE AND HIGHLY AVAILABLE ENVIRONMENT FOR THE CAFE

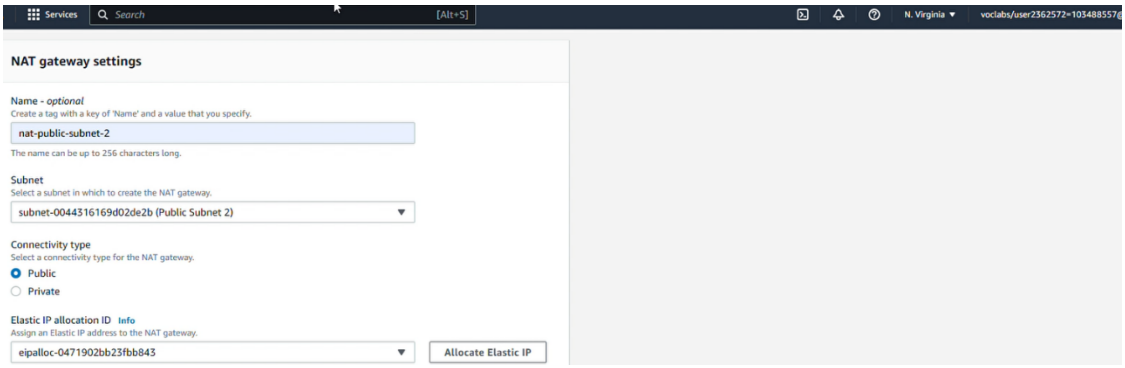
Task 1: Inspecting your environment

In this lab, I only go through the VPC settings to see how the VPC and subnets are configured. There is no configuration required in this part. I inspected that the initial infrastructure includes:

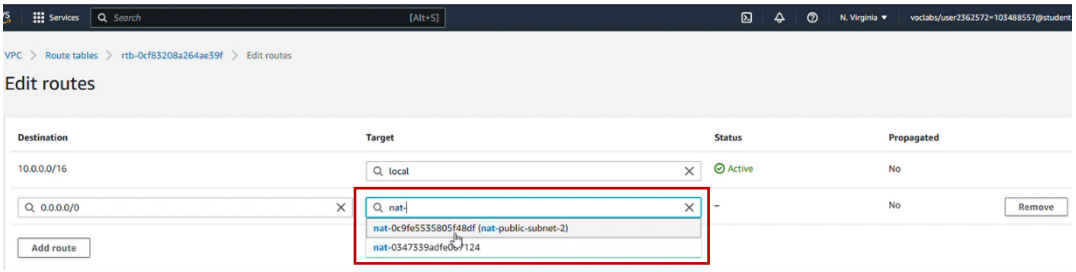
- 1 VPC with 2 Availability Zones
- 6 subnets (3 subnets/AZ)
- A security group called CafeSG is used for the following steps.
- The Route tables of all subnets were created, but they still need further configuration.

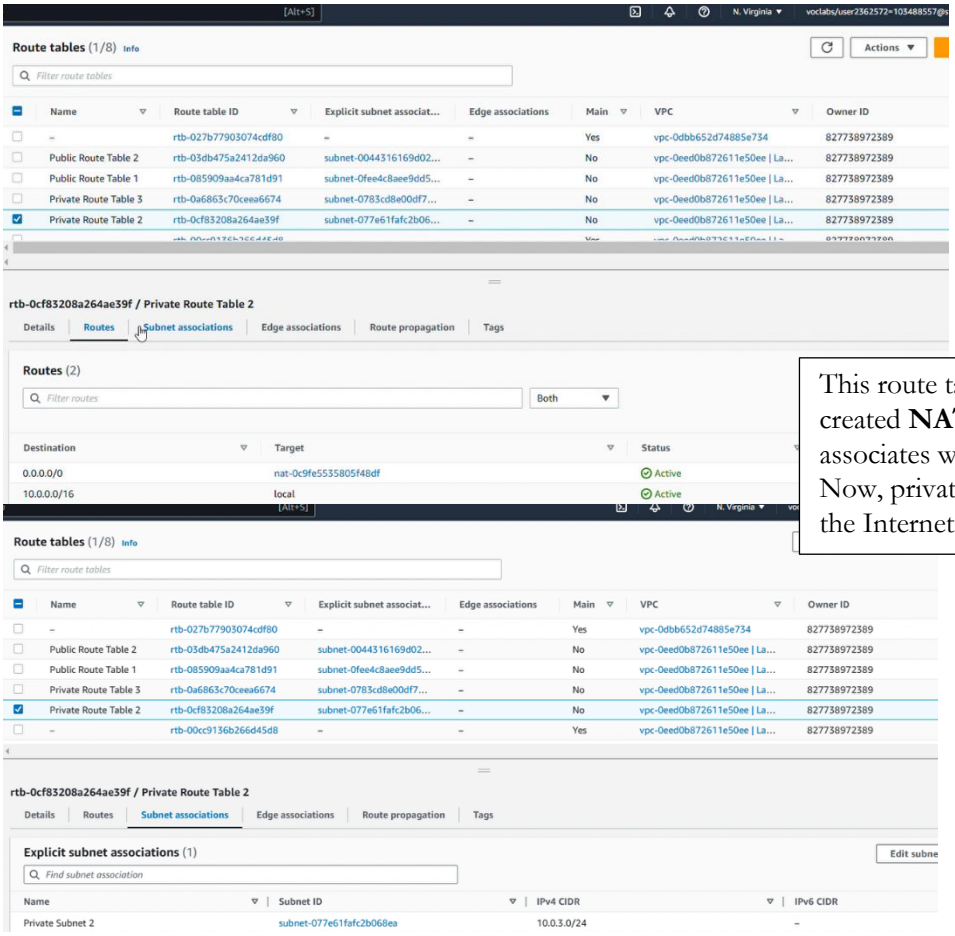
Task 2: Creating a NAT gateway for the second Availability Zone

First, I created a **NAT gateway** in the Public subnet 2. This NAT gateway would allow the scaling instances in the private subnets to access the Internet.



After creating the NAT gateway, I need to create a **Route table** to add the NAT gateway and associate it with the private subnet 2. Luckily, the Private subnet 2 Route table was prepared by the unit convenor, so I only needed to edit it without creating it.

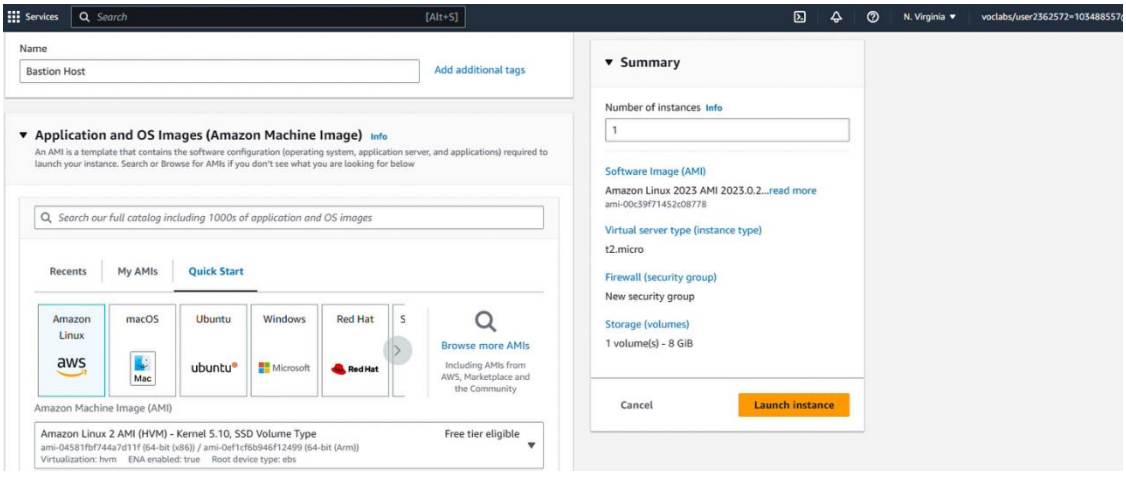


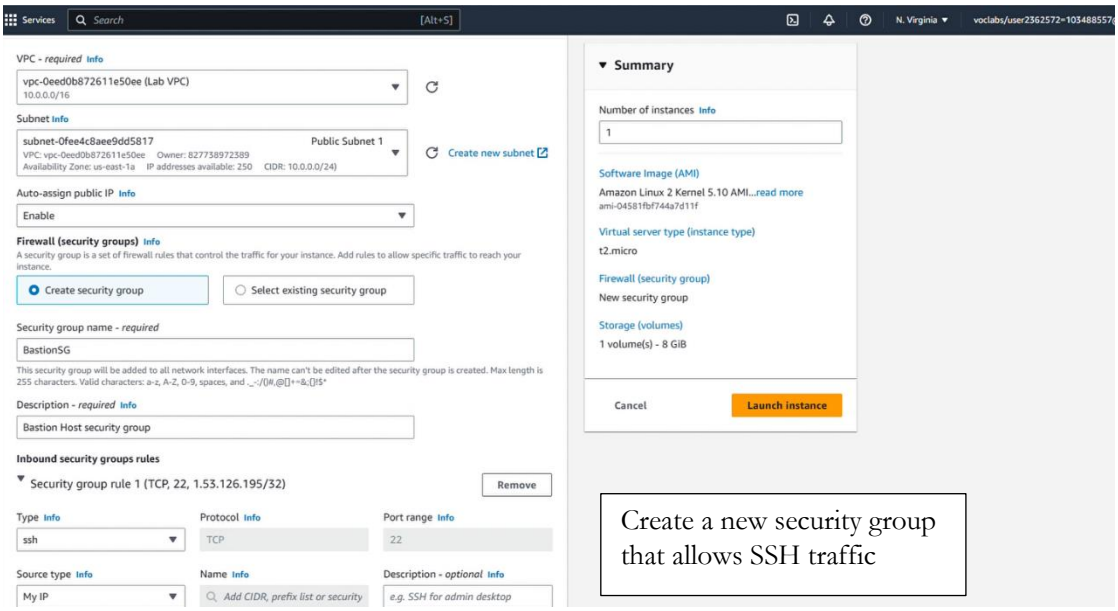


This route table routes to the created **NAT gateway**, and associates with **Private subnet 2**. Now, private subnet 2 can access the Internet.

Task 3: Creating a bastion host instance in a public subnet

In this lab, we need an EC2 instance called **Bastion Host** to ssh into the scaling instances and increase its CPU load. This will help to test if Auto Scaling can automatically add more instances to handle the high CPU load. The instance configurations are basic, we should only pay attention to its **Security group**. The inbound rule of its security group allows **SSH**, which is necessary to ssh into the instance by PuTTY step 8.

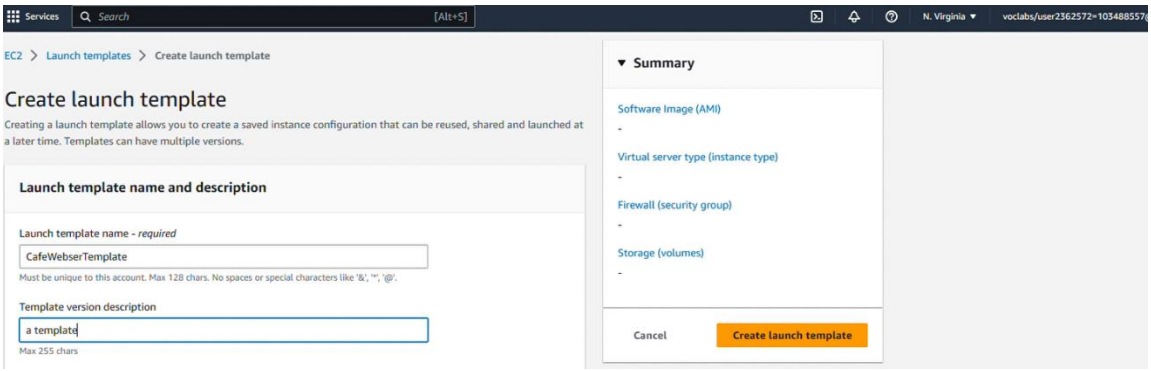


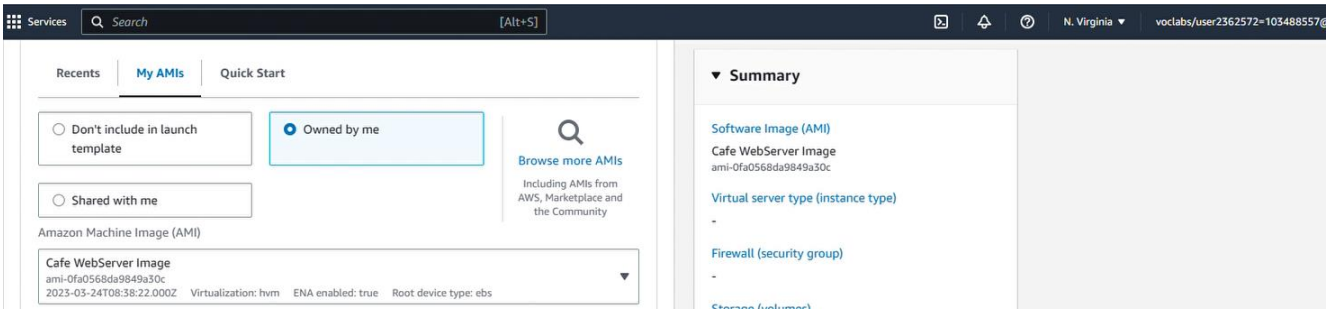


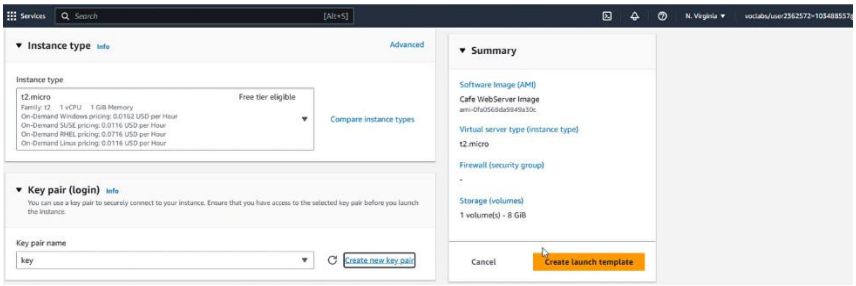
Create a new security group that allows SSH traffic

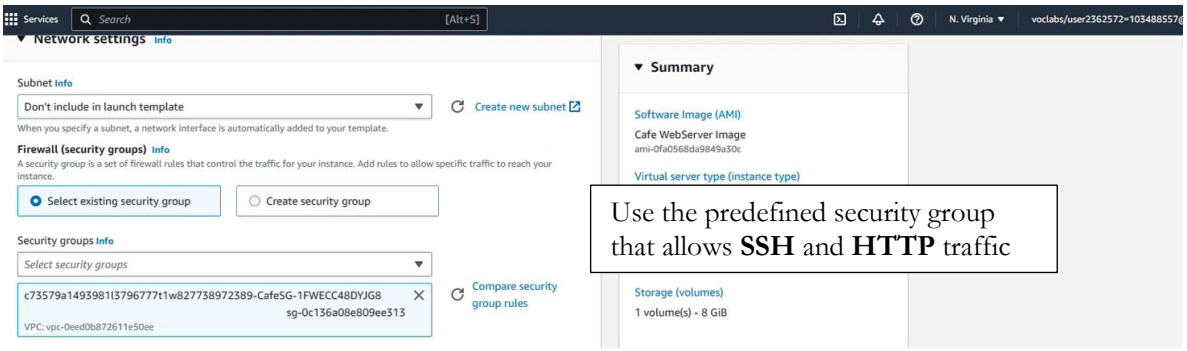
Task 4: Creating a launch template

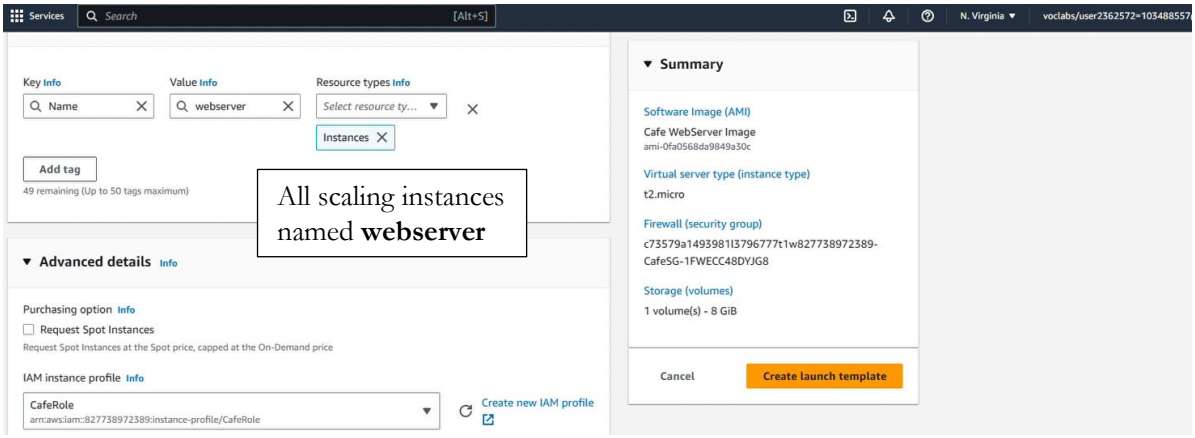
In this lab, I created a **Launch template** to prepare for the Auto Scaling Group. It was created from the AMI of the **CafeWebAppServer** instance, and it works quite similarly to the **Launch Configurations** I did in the previous labs. Both will determine how the additional instances generated by the Auto Scaling Group work.









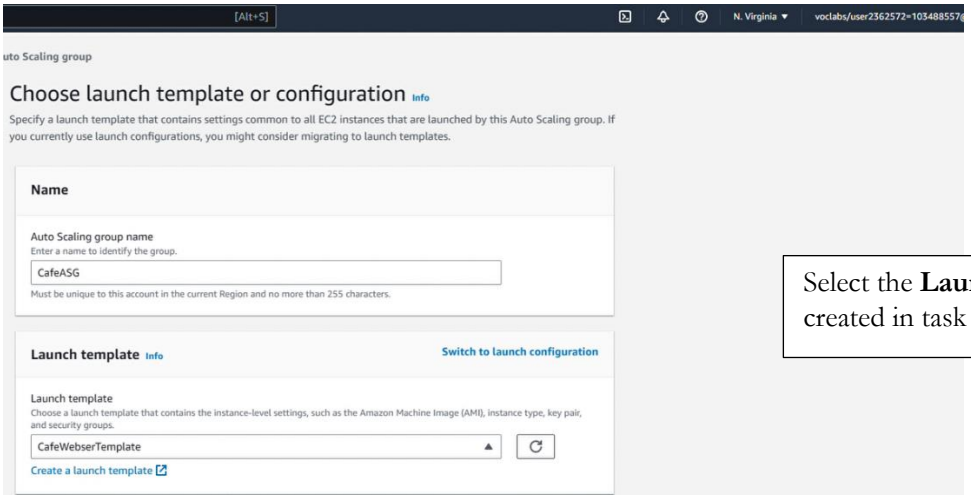


Task 5: Creating an Auto Scaling group

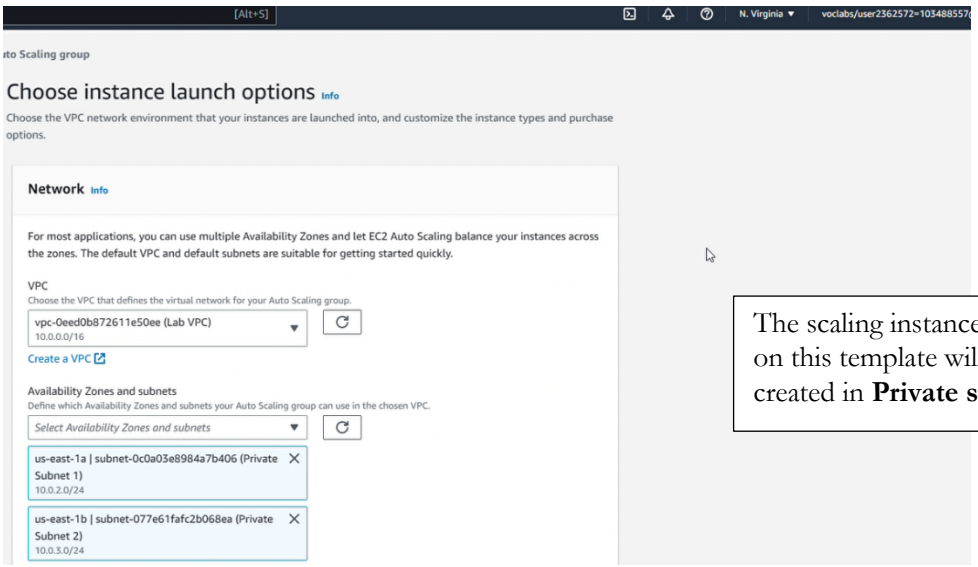
Now, everything is ready for the Auto Scaling Group. It will hold the responsibility to automate the process of checking the current state of the system and add more instances when needed.

Important points:

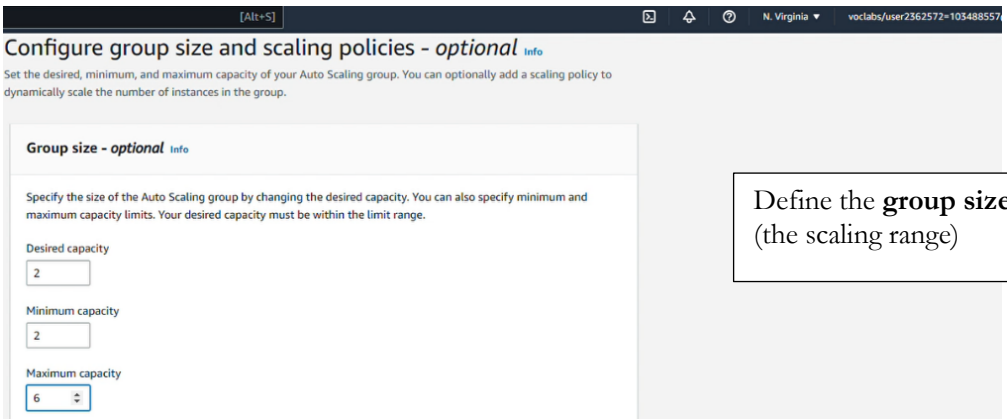
- The scaling instances will be launched in 2 Private subnets
- The group size is:
  - Desired capacity: 2
  - Minimum capacity: 2
  - Maximum capacity: 6
- A target tracking policy is included



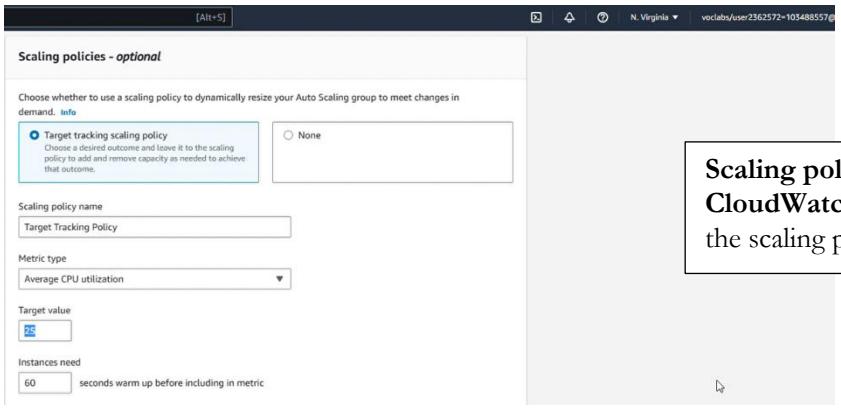
Select the **Launch template** created in task 4



The scaling instances based on this template will be created in **Private subnets**

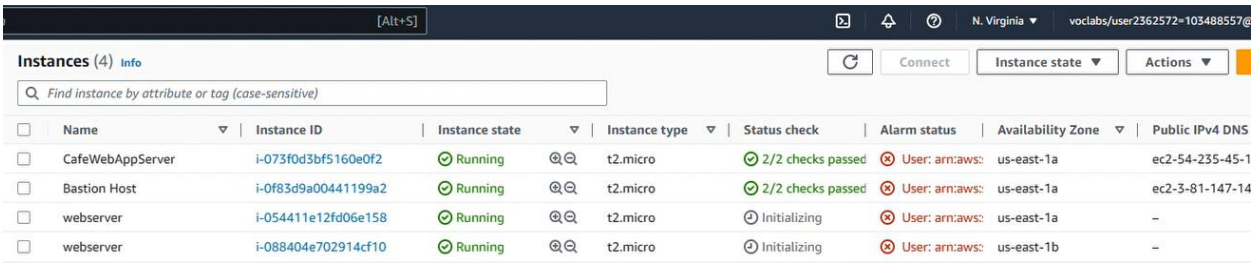


Define the **group size** (the scaling range)



Scaling policy is the metric for the **CloudWatch** alarms that invoke the scaling process.

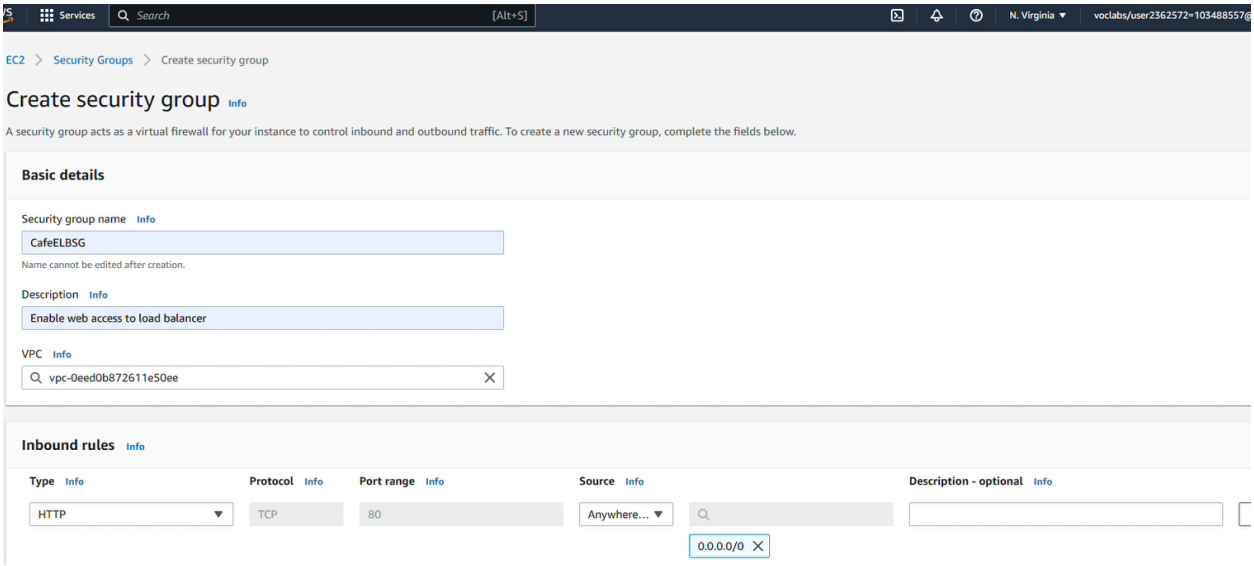
Result: 2 new instances were added.



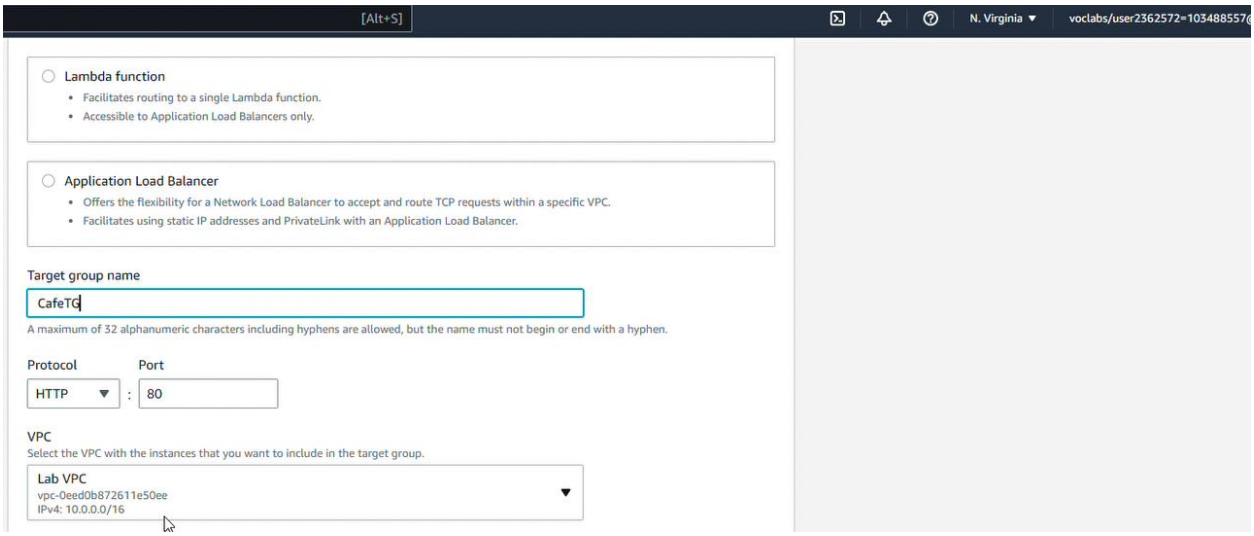
Task 6: Creating a load balancer

Before creating a Load Balancer, I need to create a **Security Group** and a **Target Group** to use on the Load Balancer configuration page.

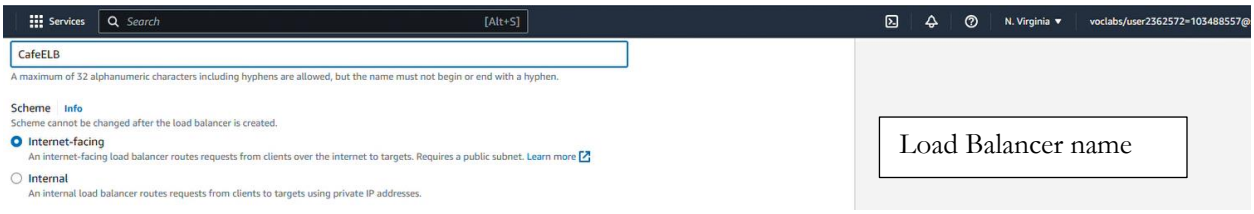
Security group: The Load Balancer security group is named **ELBSG**, and it allows **HTTP** access from anywhere.

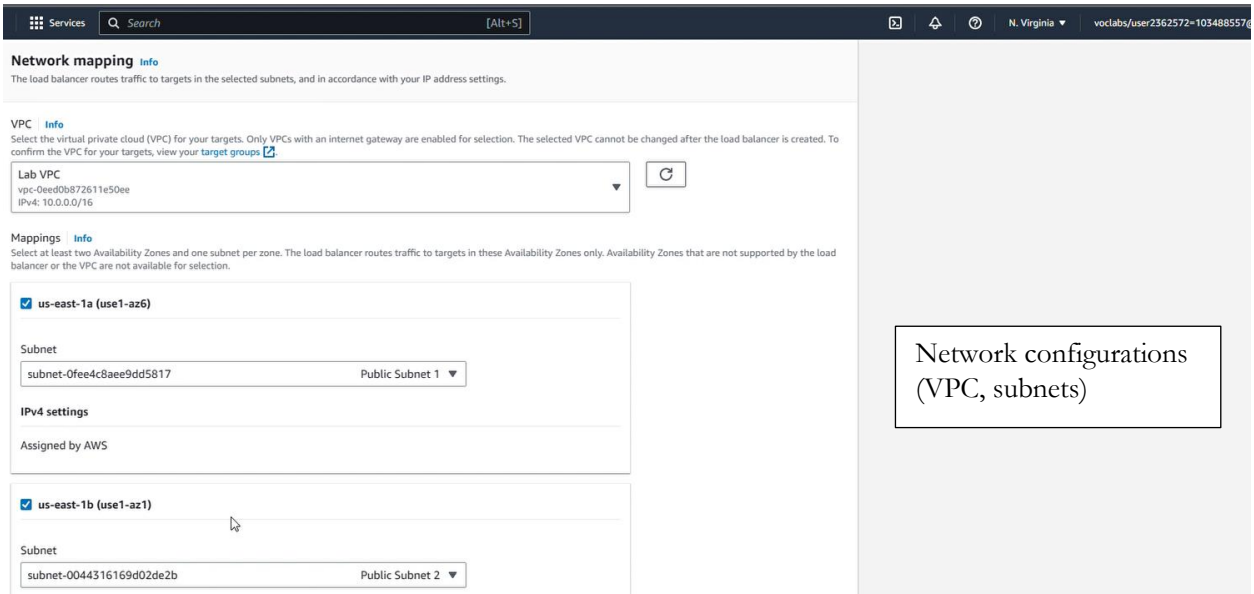


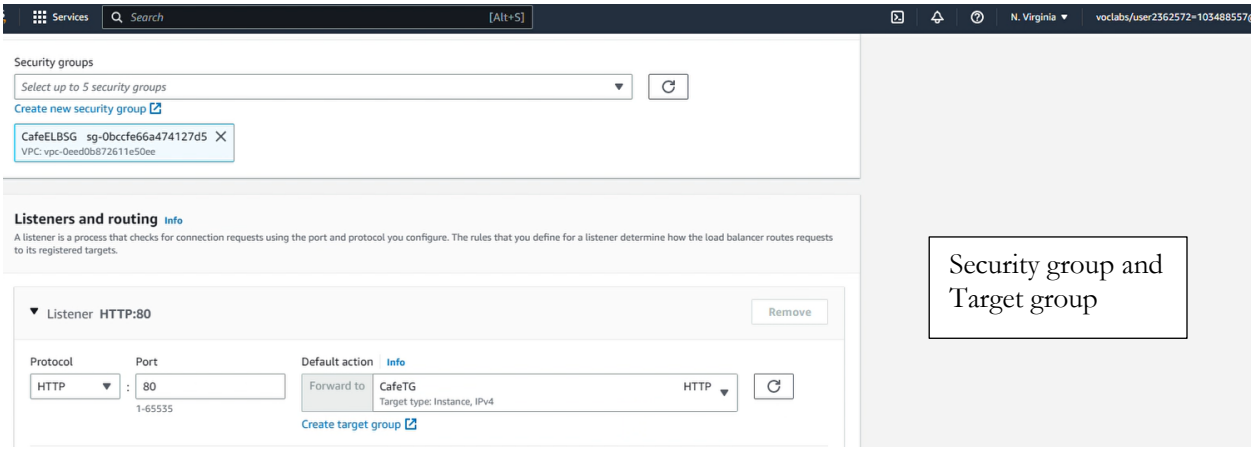
Target Group: I only created a Target group with a name and attached it to my VPC. I did not register targets for it yet because it would have the targets when it had been attached to the Load Balancer and the Auto Scaling Group.



Load Balancer: used 2 public subnets and used the **ELBSG** Security Group as well as the **CafeTG** Target Group specified above. This balancer was also attached to my VPC.

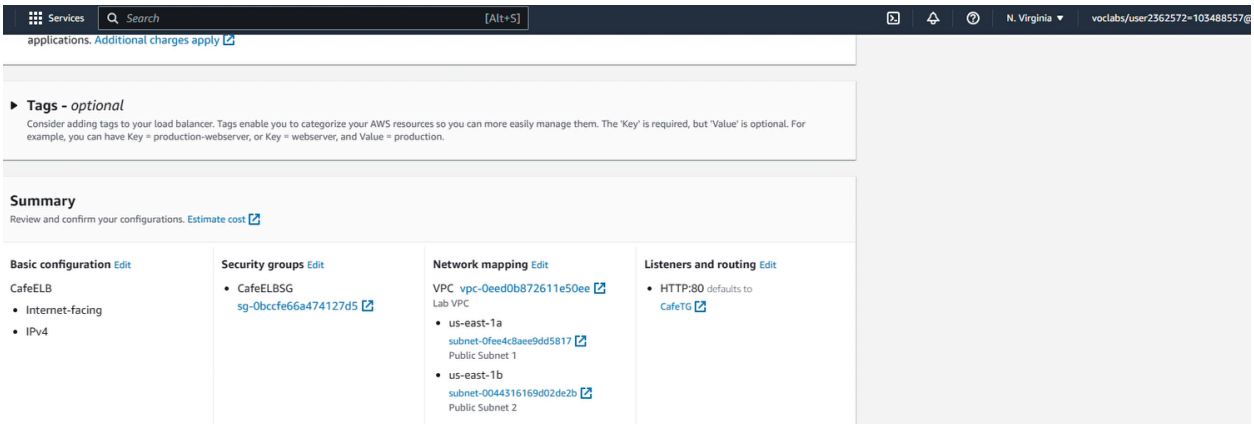




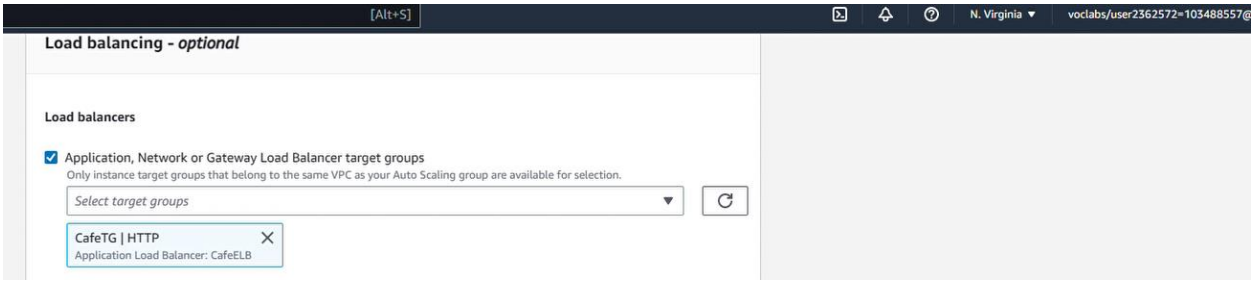


Security group and Target group

The configurations summary

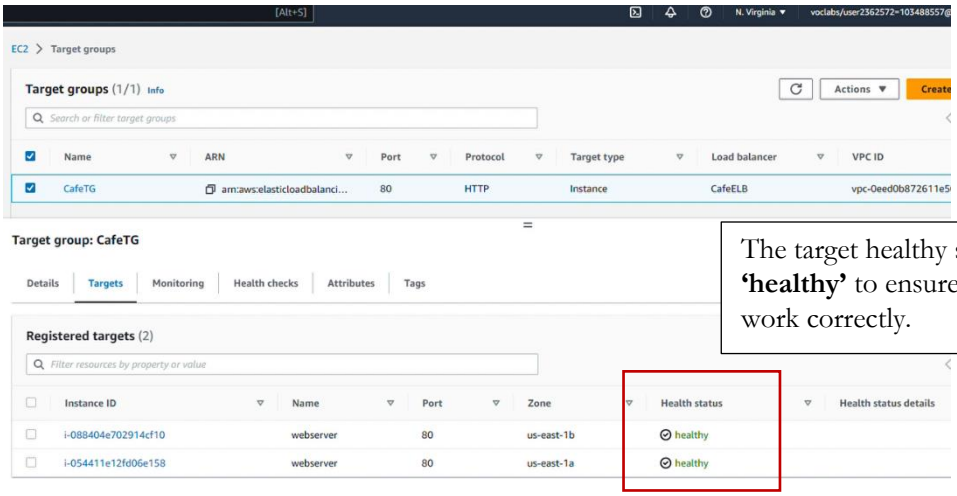


I waited for a few minutes until the Load Balancer status changed to **'Active'**. That means it was ready to be attached to an Auto Scaling Group. I returned to the **Auto Scaling Group** → Select the group → **Edit** → Select **CafeELB** as the load balancer.



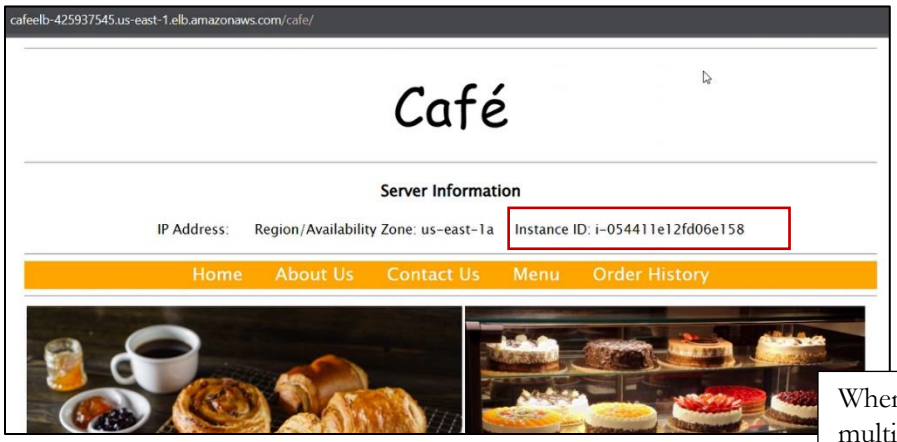
Task 7: Testing the web application

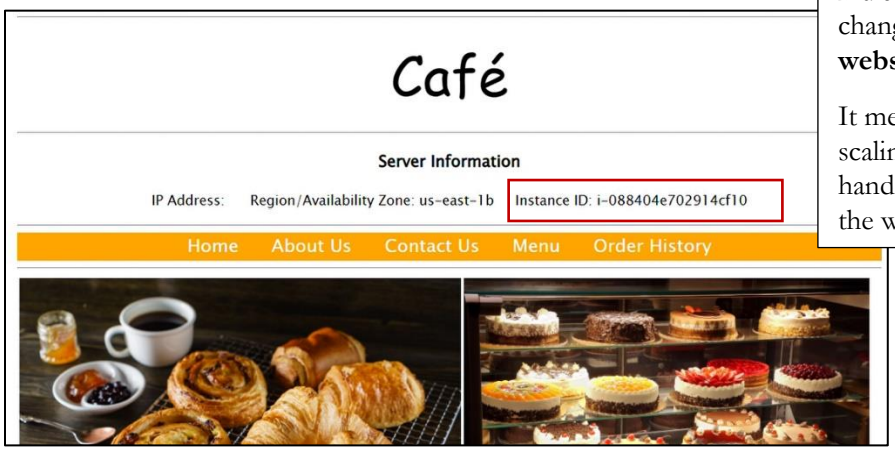
I tested all my configurations by accessing the web application through the **Load Balancer DNS Name** instead of using the EC2 Public IP address. All targets in the Target Group must be **'healthy'** before accessing the web.



The target healthy status must be **'healthy'** to ensure that it can work correctly.

Testing result





When I refreshed the website multiple times, the Instance ID changed between two **webserver** instances.

It means the **webserver** scaling instances alternately handle the access to increase the web **availability**.

Task 8: Testing automatic scaling under load

In this task, I increased the load on the web server to see whether the web app can scale out automatically.

I ssh into the Bastion Host, then I ssh into a random web server. I need to use the Bastion Host because all scaling instances are stored in the private subnets. After accessing the web server instance, I run the provided commands on Canvas to increase the CPU load.

Result

The app scaled out by adding many instances: