IT 334 - DevOps Engineering on AWS cloud

PROJECT DOCUMENTATION

Prepared by:
Ajla Korman
Kerim Šabić

Jun, 2024

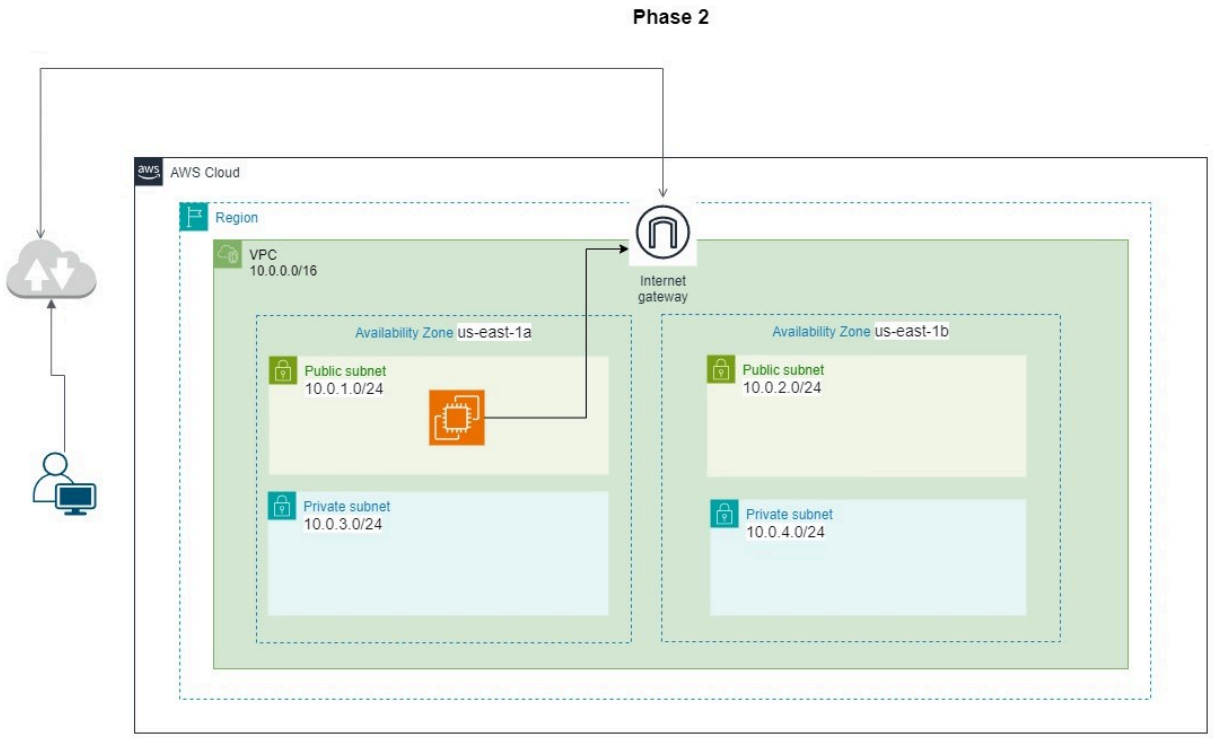
In this document you can find the solution for building a Highly Available, Scalable Web Application.

The problem is following:

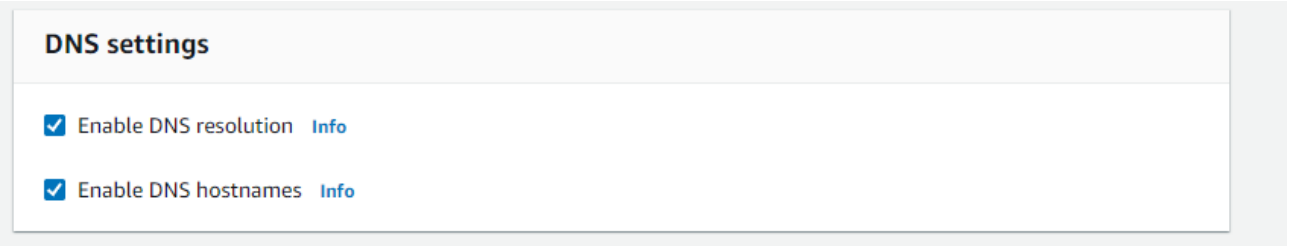
Example University is preparing for the new school year.
The admissions department has received complaints that their web application for student records is slow or not available during the peak admissions period because of the high number of inquiries.
You are a cloud engineer. Your manager has asked you to create a proof of concept (POC) to host the web application in the AWS Cloud.
Your manager would like you to design and implement a new hosting architecture that will improve the experience for users of the web application.
You're responsible for building the infrastructure to host the student records web application in the cloud. Your challenge is to plan, design, build, and deploy the web application to the AWS Cloud in a way that is consistent with best practices of the AWS Well-Architected Framework.
During the peak admissions period, the application must support thousands of users, and be highly available, scalable, load balanced, secure, and high performing.

Phase 2 - Creating a basic functional web application

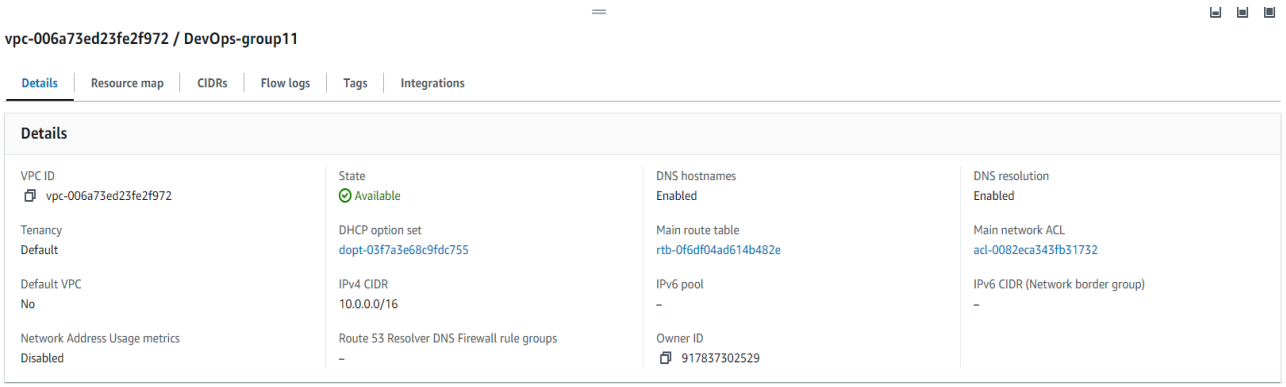
In this phase we will create a basic architecture for our web application.
The diagram below shows the structure for this phase:



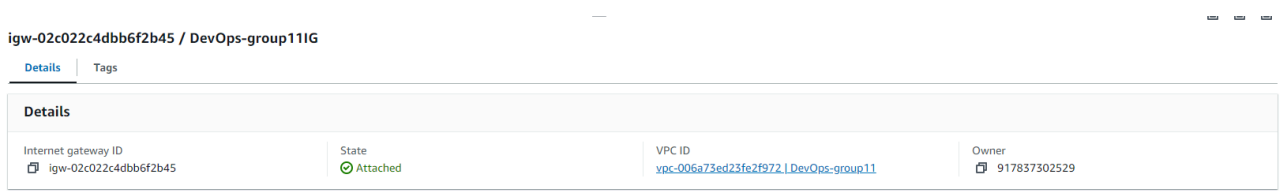
The first step is creating a VPC, Virtual Private Cloud.
Go to the VPC service, and click Create VPC.
In the name tag section put the name of the VPC, in our case it will be called “DevOps-group11”.
In the IPV4 CIDR enter the CIDR block size which is in our case “10.0.0.0/16”, this determines the number of IP addresses available for your resources and subnets.
Click on the Create VPC button and your VPC should be created.
Then select the created VPC, go to actions and click edit VPC settings. Make sure you enable dns resolutions and hostnames like this:



At the end your configuration should look like this:

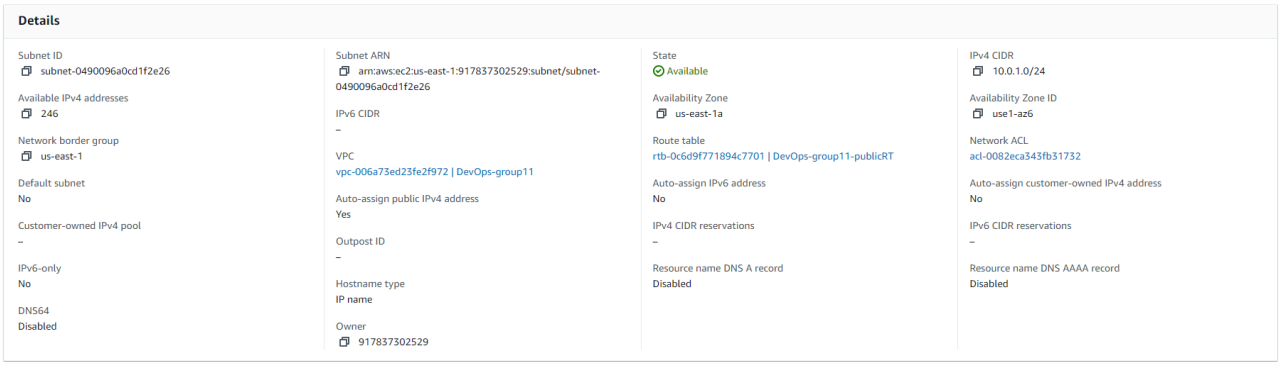


Next up we will create an Internet Gateway that is used by the resources in the VPC to communicate with the internet.
In the menu on the left select Internet gateways and click Create Internet gateway. Put the name of the internet gateway, in our case it will be DevOps-group11IG and click create internet gateway. Then choose actions and attach the Internet gateway to the VPC we have created.

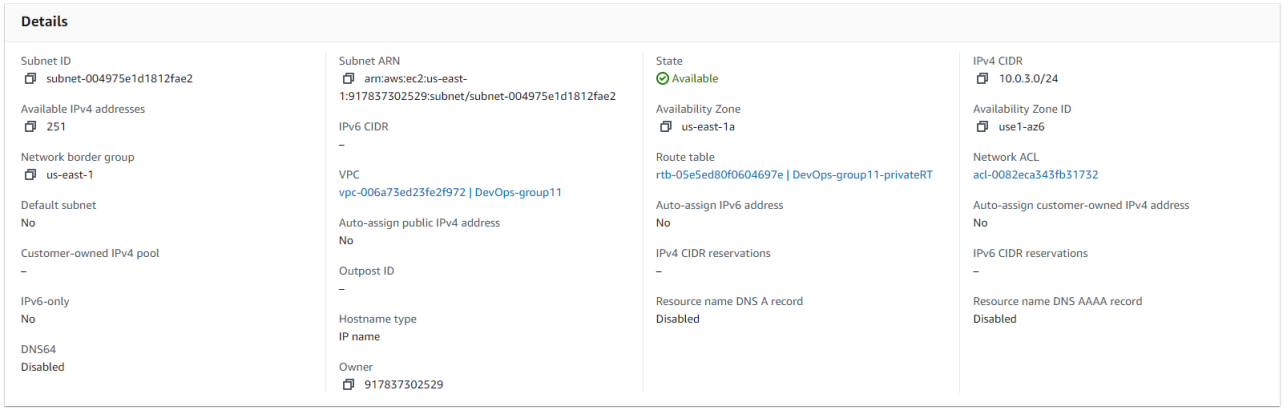


Next we will create 2 subnets in one availability zone and again 2 subnets in the second availability zone to make our application highly available. In each availability zone we will have one private and one public subnet.
Go to the subnest in the left menu and click Create subnets.
Select the VPC we have created and enter the IPV4CIDR address range and create the subnet.

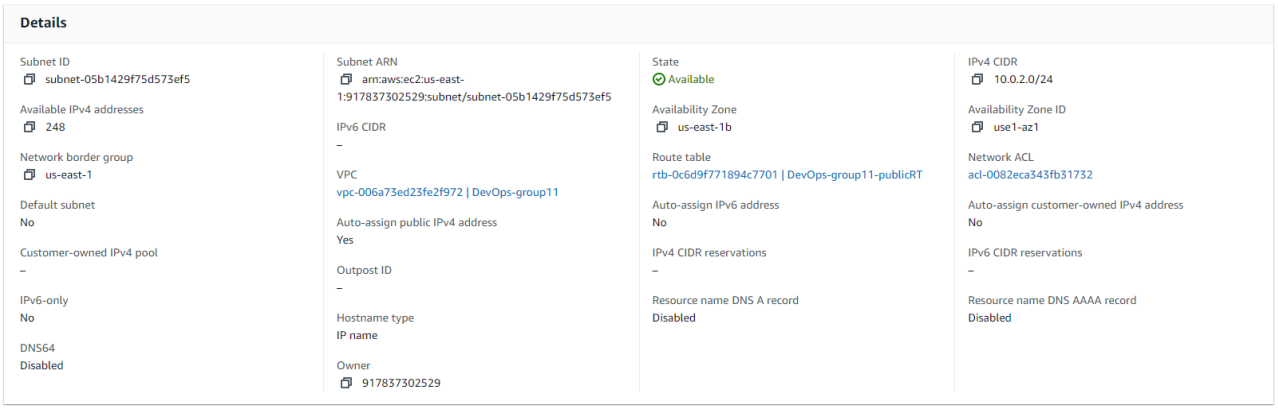
The first public subnet in the first availability zone should look like this:



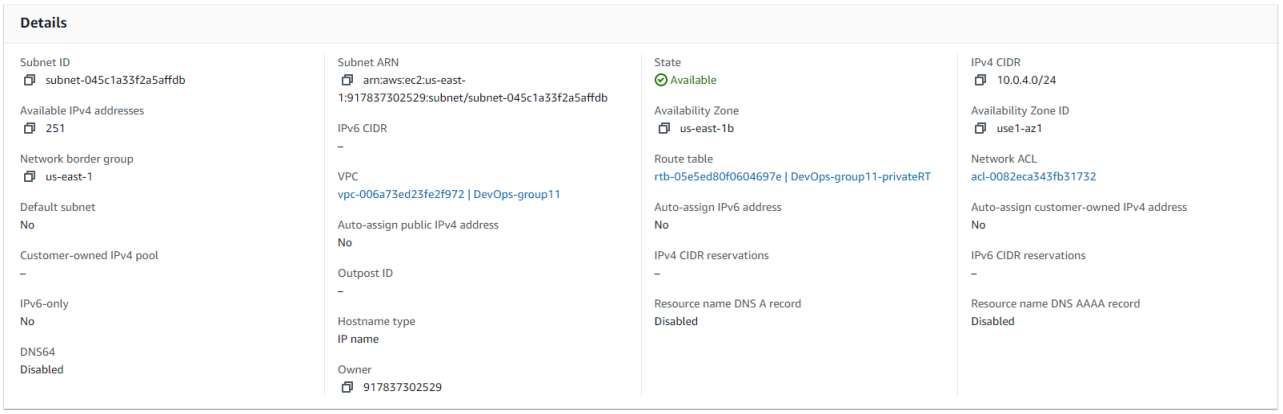
The first private subnet in the first availability zone should look like this:



The first public subnet in the second availability zone should look like this:



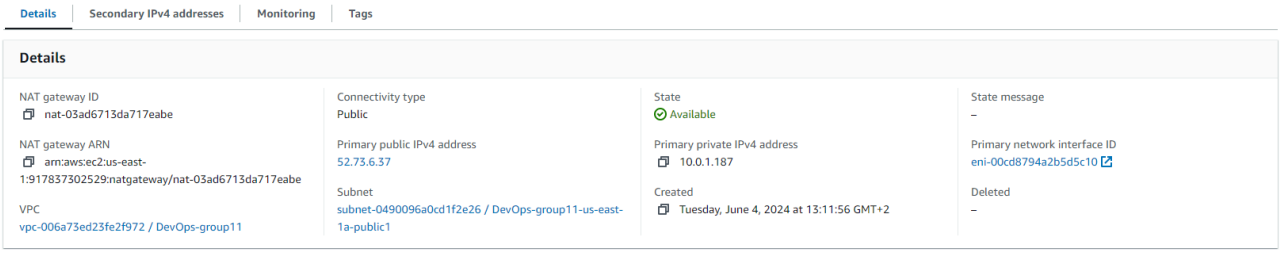
The first private subnet in the second availability zone should look like this:



Make sure you enable auto-assign public IPV4 address on the public subnets in each availability zone.

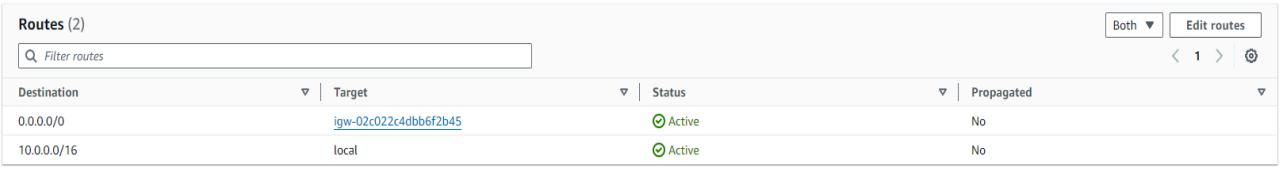
After this we will create NAT Gateway. You can use a NAT gateway so that instances in a private subnet can connect to services outside your VPC but external services cannot initiate a connection with those instances.

Click on the create NAT Gateway and create a nat gateway so it looks like this:



Next we will create private and public route tables.
Click on Create route table, put the name of the public route table and select the VPC we created. We will create routes to route the internet traffic to the internet gateway we created and request within our VPC to be routed locally.

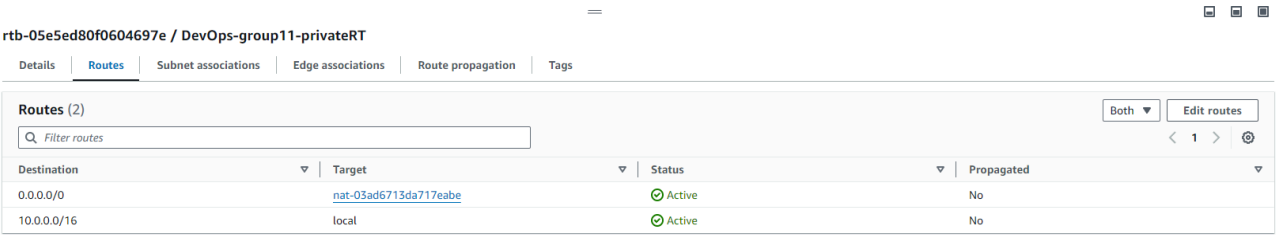
When created click on routes, edit routes and add the following routes:



Click on the subnet associations and associate to the both public subnets we created.

Next we will create a private route table.
Click on Create route table, put the name of the private route table and select the VPC we created. We will create routes to route the internet traffic to the NAT gateway we created and request within our VPC to be routed locally.

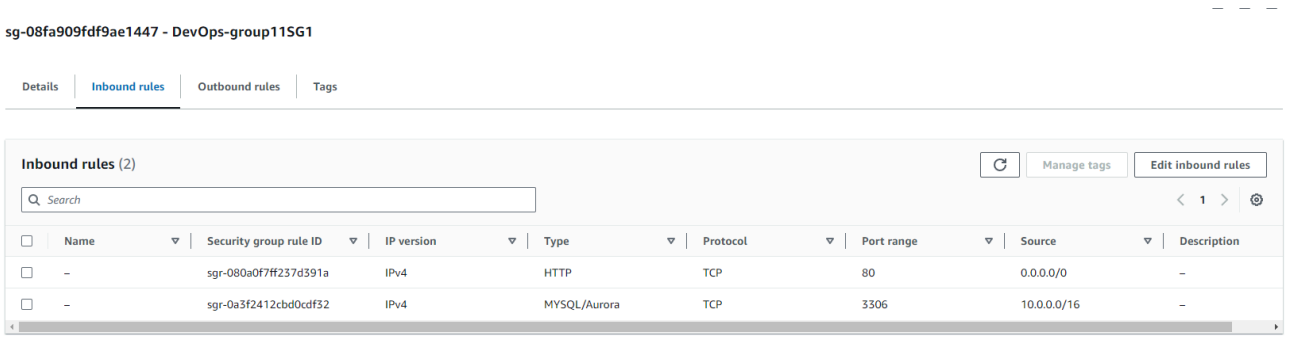
When created click on routes, edit routes and add the following routes:

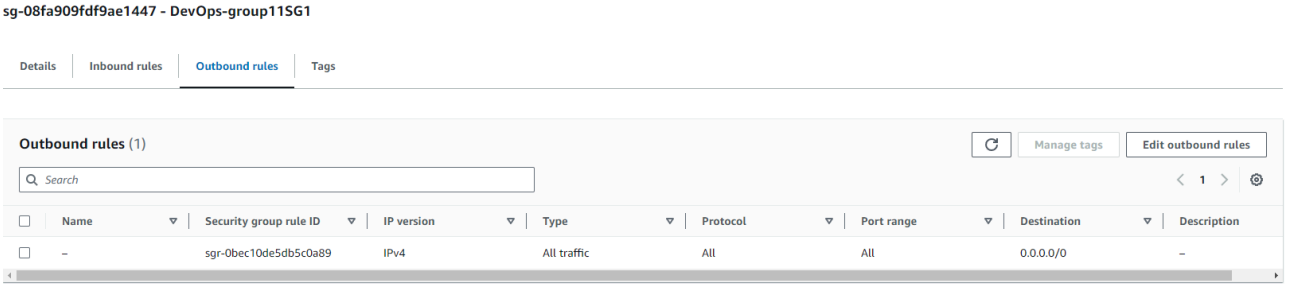


Click on the subnet associations and associate the route table with both private subnets we created.

Next we will create an EC2 instance to host our web application.
Find the EC2 service and click on the Create Instance button.
Put the name of the instance, for the application on the os image select Ubuntu, instance type should be t2.micro, for keypair chose vockey
In the network settings click edit. Select our VPC, put the insane in the first public subnet we created, enable auto assign public ip
Click create security group, here we will create security group for our ec2 instance

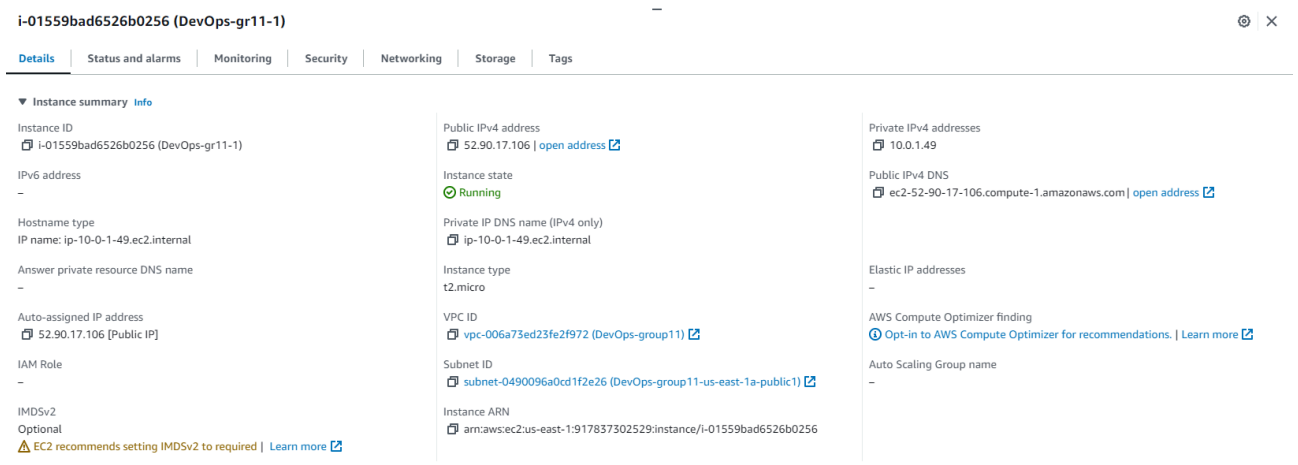
Enter the security group name and add the rules so at the end the security group look like this:



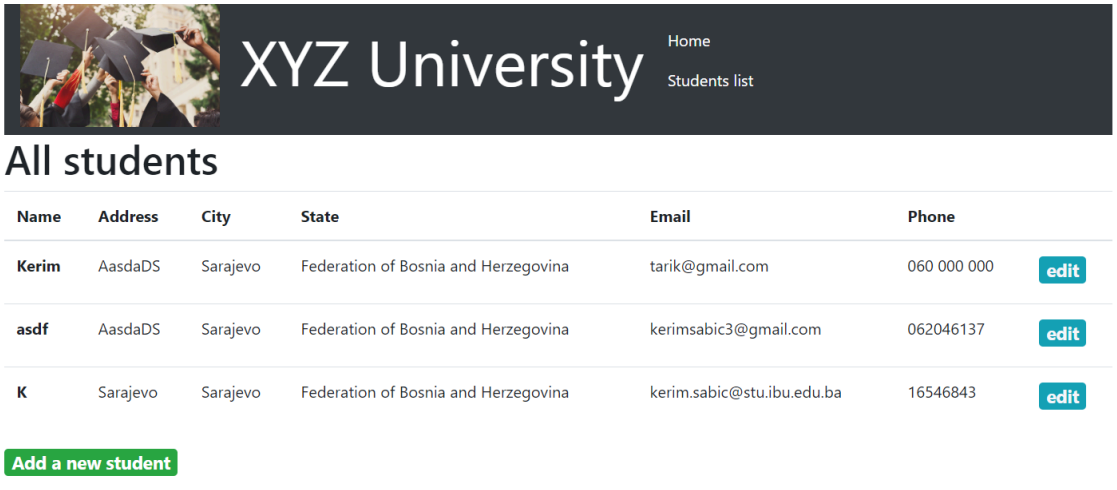


After creating the security group select it in the EC2 creation menu.
Scroll down to the advanced details, expand it and paste the user data that is given in the UserdataScript-phase2

Our instance should be running now and we should be able to access our application by our public IPV4 DNS.



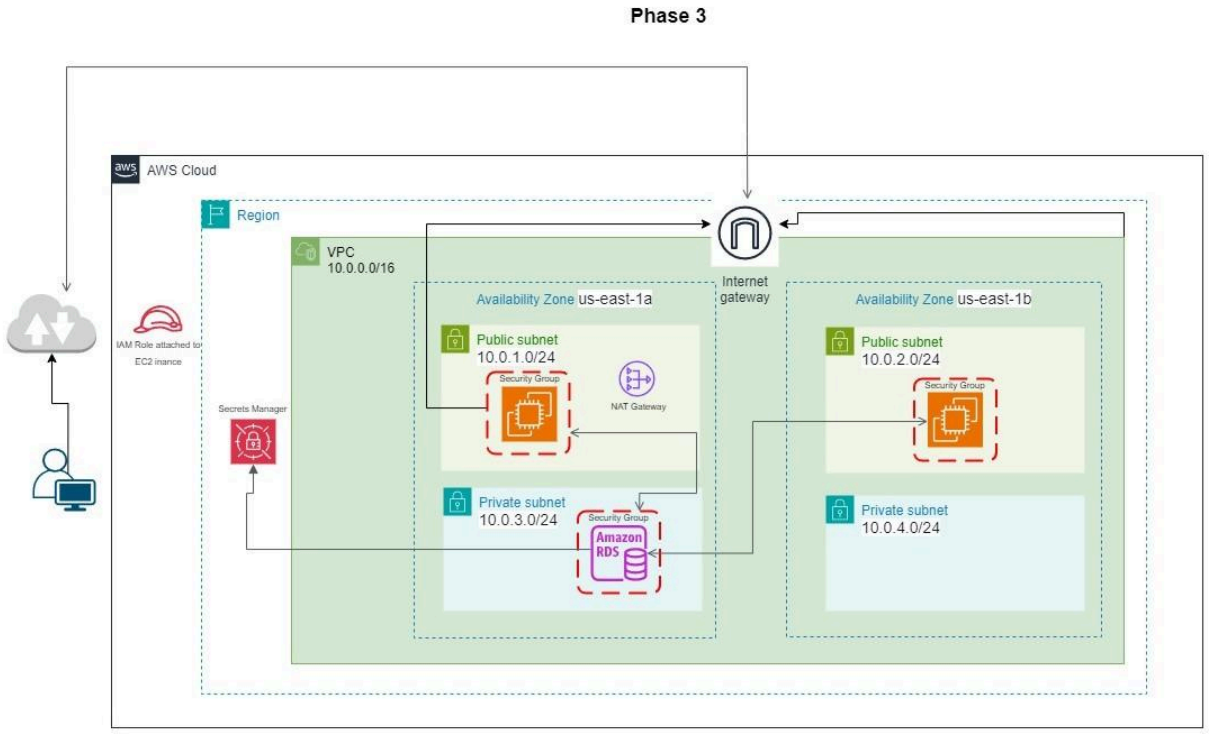
We can now perform all CRUD operations on our application:



Phase 3 - Decoupling the application components

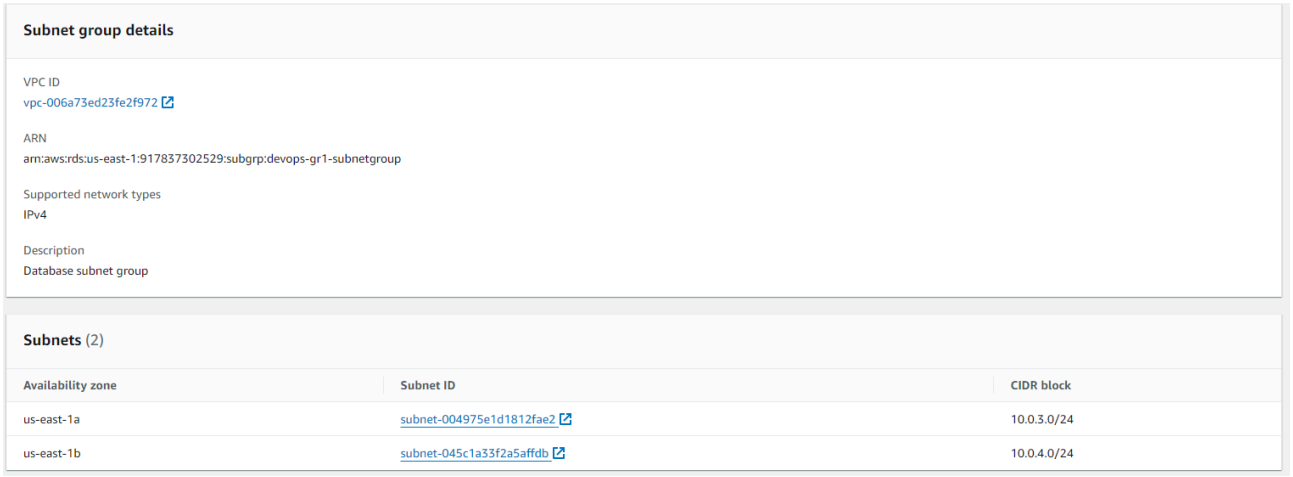
In this phase we will create a RDS MySQL database that will run on the private subnet in the same availability zone as our application. We will transfer the data from the database running on the EC2 instance to our RDS database. With this approach we can create new instances of our application and connect it to the rds database instead of saving data to that instance. This will help to make our application highly available.

The diagram below shows the structure for this phase:



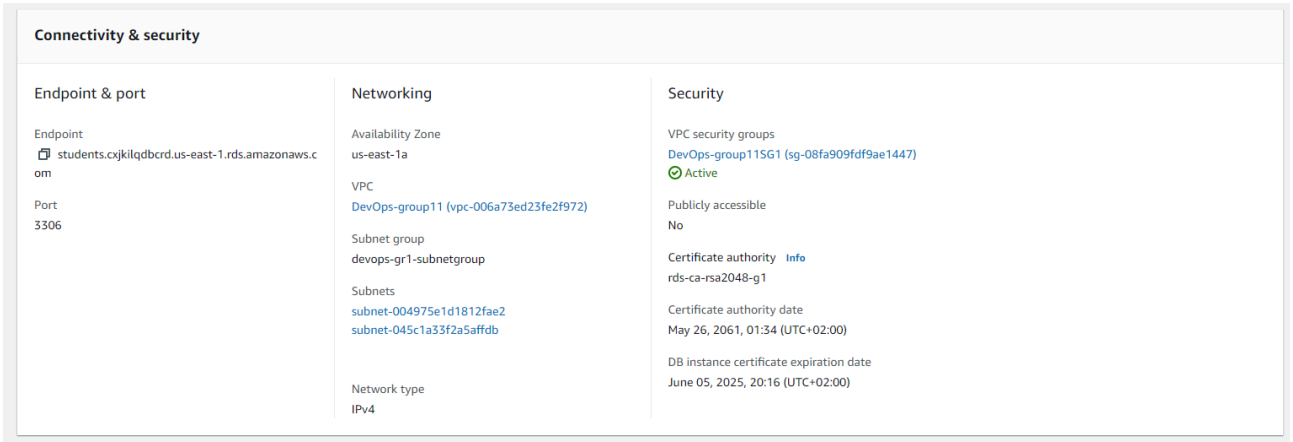
First create subnet groups for the database.

Go to the rds service, and click create subnet groups. When creating associate 2 private subnets we have created.



Go to the RDS service in the AWS console and create a new database instance.
For the engine type choose MySql, engine version should be the latest stable one, for templates use Production, select single db instance.
For the database instance id put Students and, for the master username put nodeapp, click self managed and add a password for the database.
DB instance class should be db.t3.micro, add your VPC and subnet group we created previously.

Created database should look like this:



Open Cloud9 service and create a new environment. New environment should be created with a type of new EC2 instance, use t3.micro for platform use linux for platform, for connection use ssh, use created VPC and put in the wanted subnet, in our case first public one in the first availability zone.
After creating instance, open the cloud9 environment.
Now we need to create a secret manager for storing our database secret. In the terminal paste the following command:

**aws secretsmanager create-secret **
**--name Mydbsecret **
**--description "Database secret for web app" **

--secret-string "{\"user\":\"nodeapp\",\"password\":\"ADD_PASSWORD\",\"host\":\"students.cxjkilqdbcrd.us-east-1.rds.amazonaws.com\",\"db\":\"STUDENTS\"}"

This will create a new secret with AWS Secrets manager which you can see when you go to this service, select the Mydbsecret and click on the button to retrieve secret value.

Next up we will drop the database that sits on the ec2 instance we first created into the data.sql file. We can do this by pasting the following script:

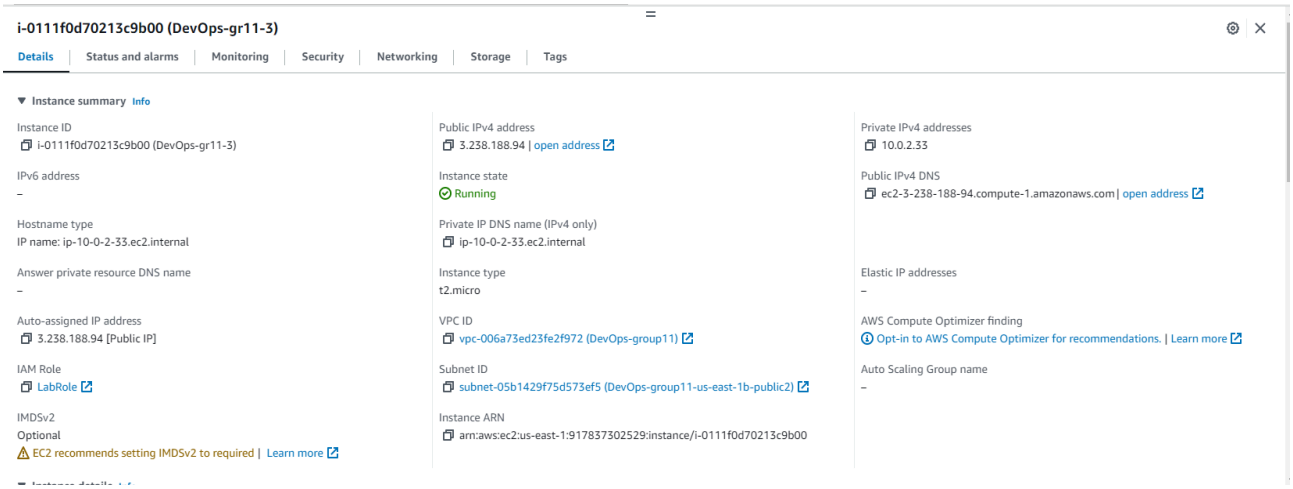
mysqldump -h <enter private ip of your ec2 insance> -u nodeapp -p --databases STUDENTS > data.sql

New file should be created called data.sql, we will now import this data into our database we created with rds.
Paste the following script:

mysql -h <here put the endpoint of the rds database you created> -u nodeapp -p STUDENTS < data.sql

Now create a new EC2 instance following the same guidelines as for the first, just in this one in the network section put it inside the second public subnet in other availability zone and also for the user data paste the user data from the UserdataScript-phase3.
Also attach the user role to the instance. User role is LabRole.

At the end EC2 should look like this:

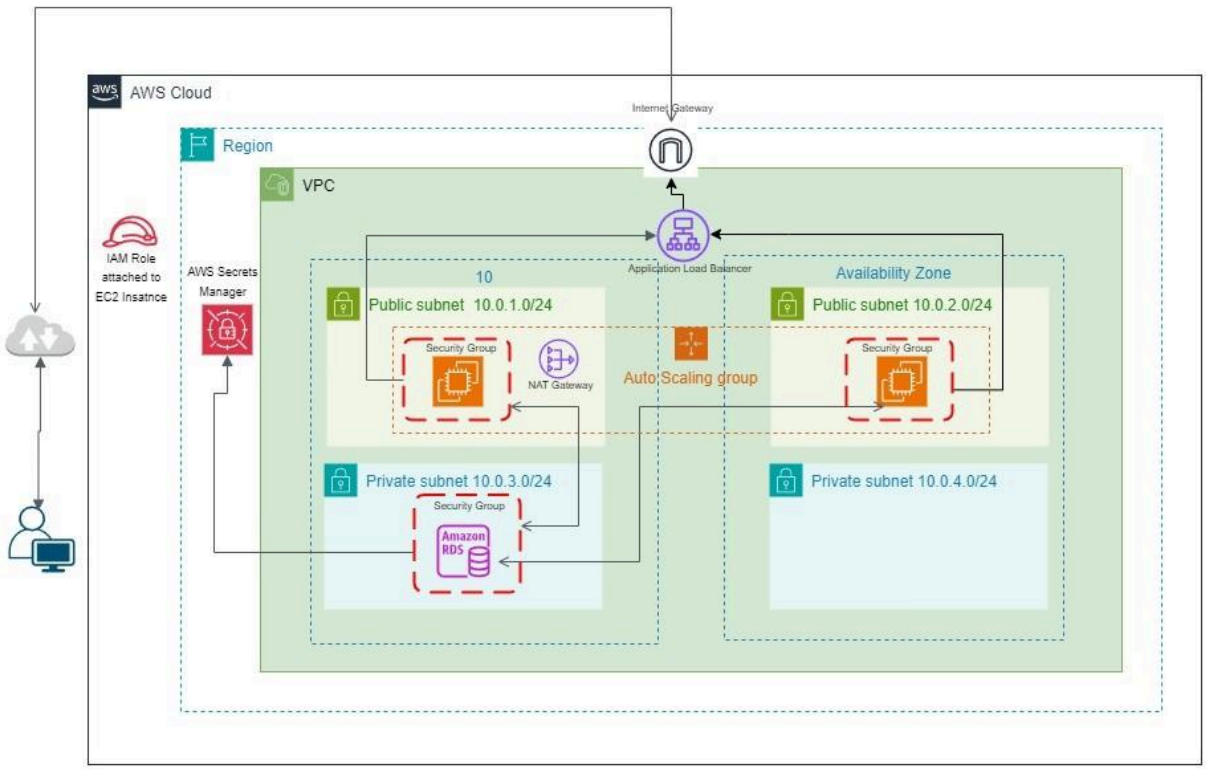


Now the database should be connected to the ec2 instances and they query data from the RDS endpoint.

Phase 4 - Implementing high availability and scalability

In this phase we create fully functional, highly available and scalable applications by creating load balancer, laid balancer security group, target group, amazon machine image, launch template and auto scaling group.

The diagram below shows the final structure:

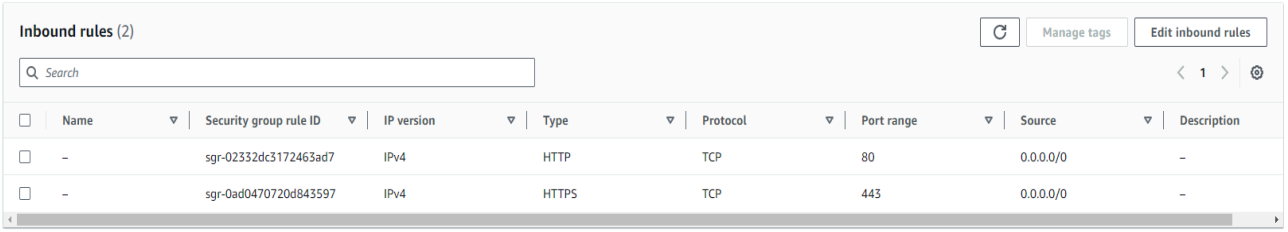


The first step is creating a load balancer.

Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones.

It monitors the health of its registered targets, and routes traffic only to the healthy targets.
We will create an application load balancer.
Click on the create load balancer, then choose application load balancer, put the name of the load balancer, select internet facing.
In the networks mapping section, select your VPC, the one we created, then select 2 availability zones where you will select 2 public subnets, one in each availability zone.
For the security group select create new security group. Here we will create a new security group for our load balancer.

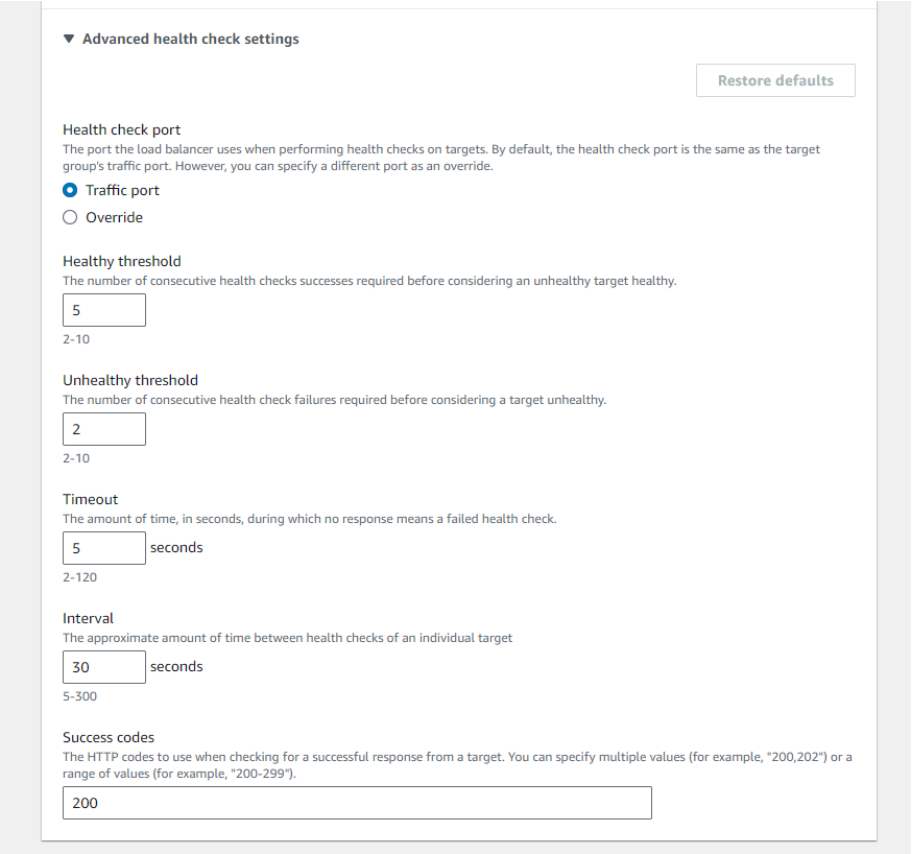
Put the name of the security group, select our VPC and add inbound rules like this:



This will allows HTTP and HTTPS traffic to our load balancer.
After creating this security group, return to the load balancer configuration and for the security group select the newly created security group.

Then in the listening and routing section click on the create target group.

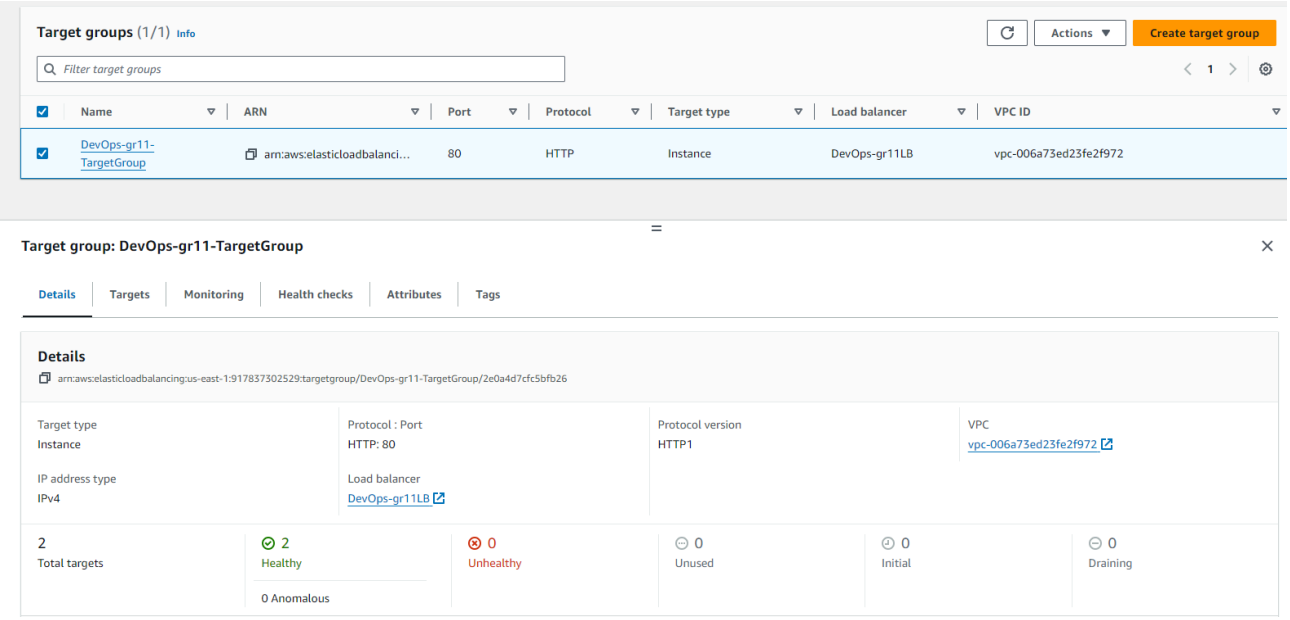
In new window select instances, put the name of the target group, select our VPC and in the advanced details configure following:



Target groups route requests to individual registered targets, such as EC2 instances, using the protocol and port number that you specify.

In the register targets select the EC2 instance that you want to be registered as a target.

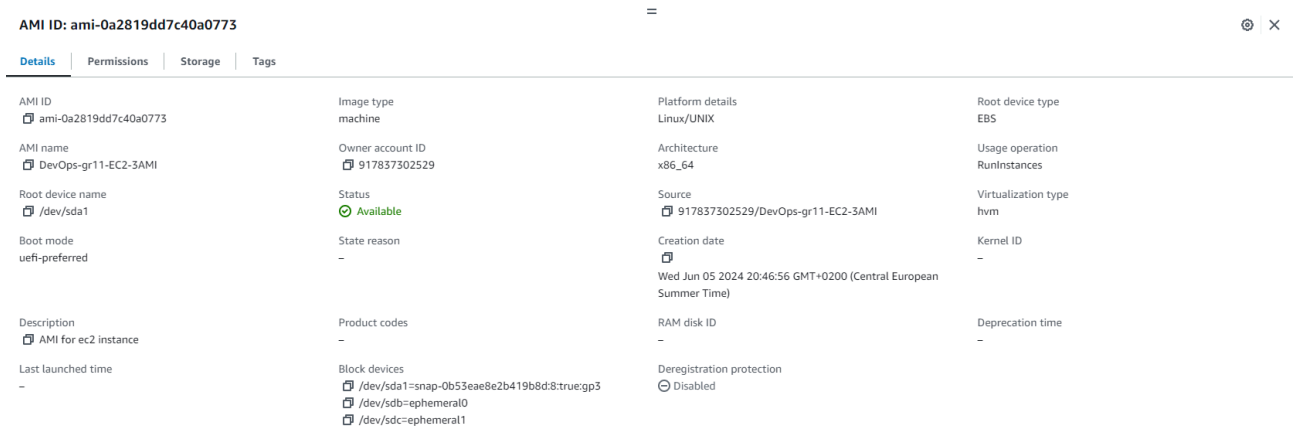
Target group should look like this:



Now return to the creation of the load balancer and in the target group select the newly created target group.
Scroll down to the end and create a load balancer.

Next up we will create an Amazon machine image (AMI) from the EC2 instance.
An Amazon Machine Image (AMI) is a supported and maintained image provided by AWS that provides the information required to launch an instance.
Go to the EC2 instances, select the second created instance and in the actions menu select image and templates and the select create image.
Put the name and description of the image and create AMI

Once created your AMI should look like this:

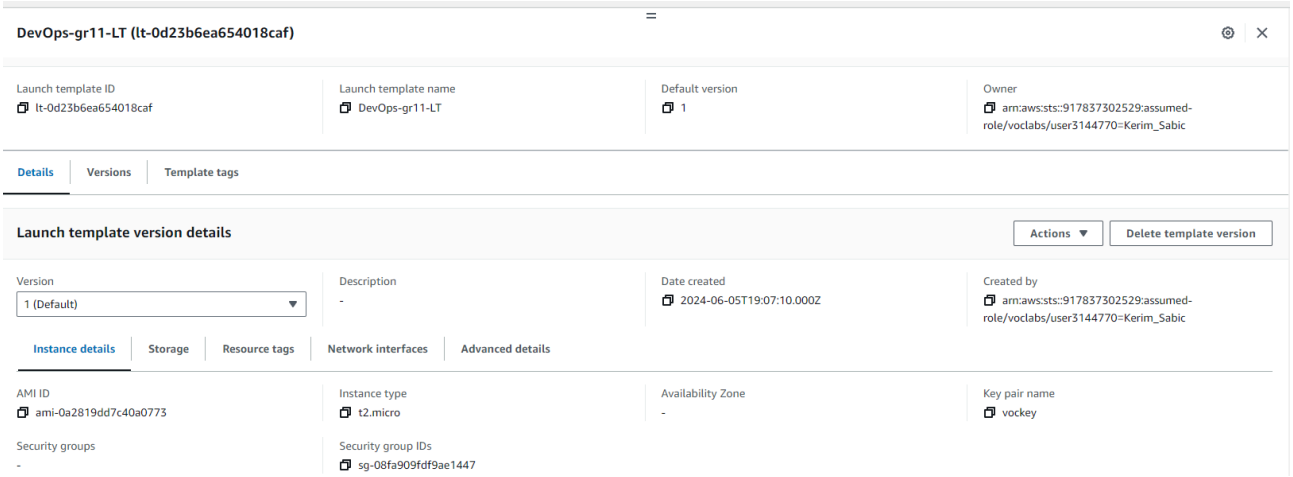


Now when we create AMI, we will create a Launch template.

You can use a launch template to store instance launch parameters so that you do not have to specify them every time you launch an instance. For example, you can create a launch template with the AMI ID, instance type, and network settings that you typically use to launch instances. When you launch an instance using the Amazon EC2 console, an AWS SDK, or a command line tool, you can specify the launch template instead of entering the parameters again.

Put the name of the template, for the AMI chose my amis, and select the AMI we have created just before.
For the instance type select the same as the instance type that already exists, that is t2.micro.
Key pair name should be vockey.
Select the security group we first created, that is attached to the other EC2 instances, and for user data paste the user data provided in UserdataScript-phase3.
Create a launch template.

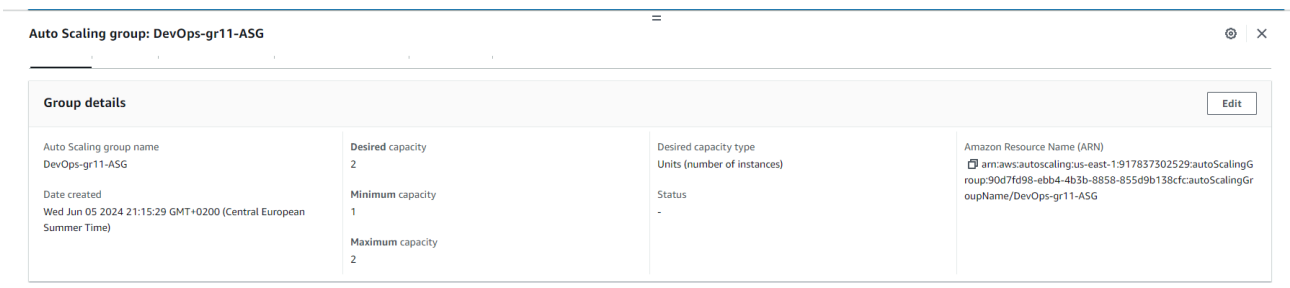
It should look like this:

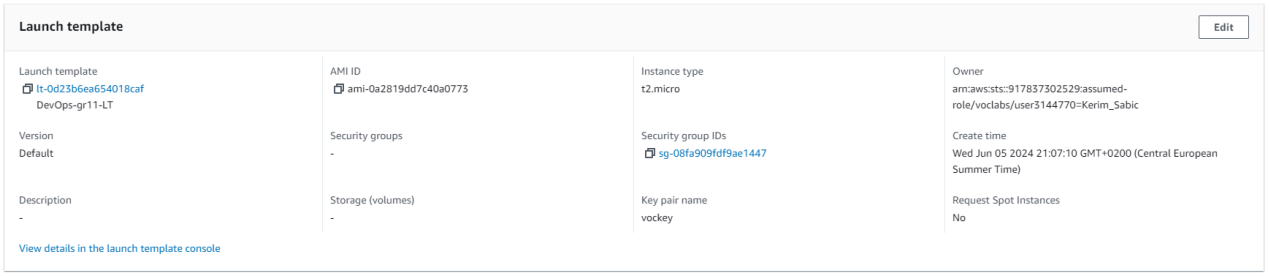


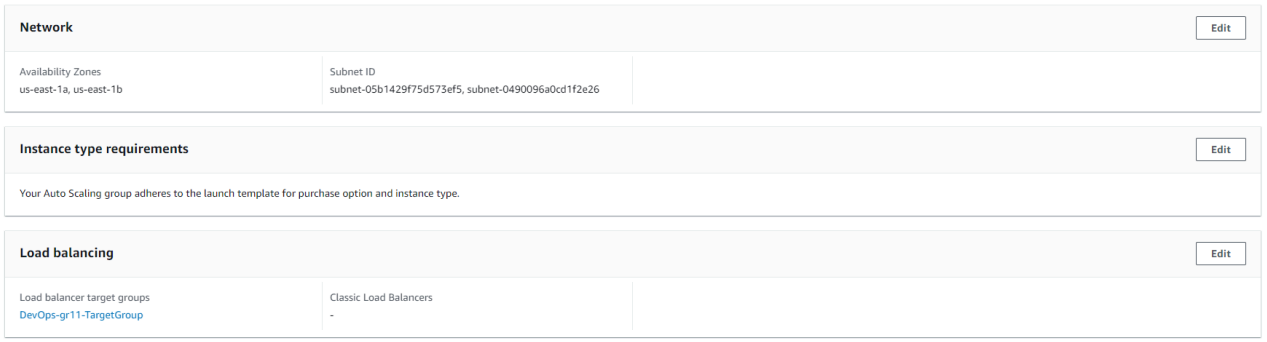
Finally create an auto scaling group.

An Auto Scaling group contains a collection of EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management.
Both maintaining the number of instances in an Auto Scaling group and automatic scaling are the core functionality of the Amazon EC2 Auto Scaling service.
The size of an Auto Scaling group depends on the number of instances that you set as the desired capacity.
You can adjust its size to meet demand, either manually or by using automatic scaling.
An Auto Scaling group starts by launching enough instances to meet its desired capacity.
It maintains this number of instances by performing periodic health checks on the instances in the group.
The Auto Scaling group continues to maintain a fixed number of instances even if an instance becomes unhealthy. If an instance becomes unhealthy, the group terminates the unhealthy instance and launches another instance to replace it.

Click on the Create auto scaling group. Put the name and for the launch template select the newly created template.
Configure the VPC to be the VPC we created and choose both of the public subnets we created.
For the load balancer select Attach to an existing load balancer and then choose the load balancer we created same with the target group.
For the health check period select 90 seconds
Next, for the desired capacity select 2, minimum 1 and maximum 2.
Don’t select scaling policies and finally add tags.
Add key name and value add the name you want your insurance to have.
Finally create the auto scaling group.







We have successfully create highly available and scalable web application.

We can now access our web application through the application load balancer DNS name, and the load balancer will automatically distribute the requests to appropriate EC2 instances.