**Deploy a 3 Tier Architecture On AWS using RDS Aurora Read/Write replica**

Objectives:

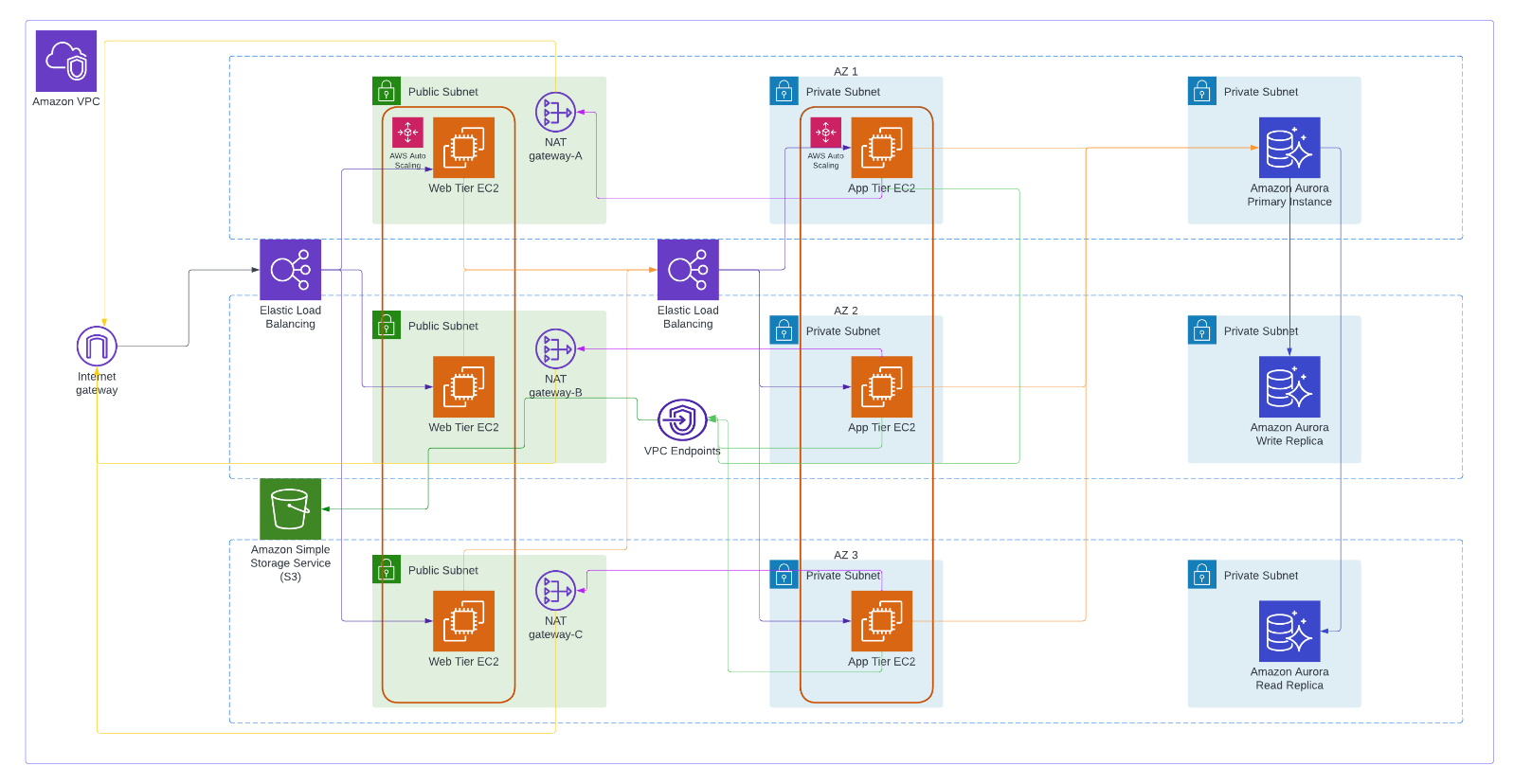
* Highly scalable, highly available, & fault-tolerant.
* Using custom VPC instead of default ones.
* Secure & apply best practices of IAM & security.

Summary:

In this project, we are going to be building a 3 Tier Web Application. In this project, the goal is to create a highly resilient website which can quickly auto-scale with respect to the incoming traffic & Use the best security practices for Access control to various resources. The web tier will be public facing EC2 instances to which the public will get access to with the help of External ALB. From the Web tier, the traffic will be routed internally to App tier Private Instances using Internal Load balancer.

We will be using IAM roles for EC2 to use S3 & SSM. Then we will be creating various network resources in one click in the VPC instance & create 5 Security groups for External Facing Load balancer, Web tier EC2, Internal Facing Load balancer, App tier Private instances & DB. Aurora RDS DB will be created with Multi-AZ read/write replica module for high scalability & availability of the Database.

Architecture:



Services Used:

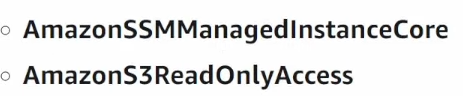
1. IAM Role
2. S3
3. 3 NAT Gateways
4. 3 Elastic IPs
5. S3 Endpoint
6. 9 Subnets
7. IGW
8. 1 Public 6 Private Route Tables
9. RDS
10. 5 SGs
11. EC2 (5-10)
12. ALB - 2
13. ASG - 2
14. TG - 2
15. Template -2
16. AMI - 2
17. Snapshots Auto Created

Steps:

**Part 1: Setup**

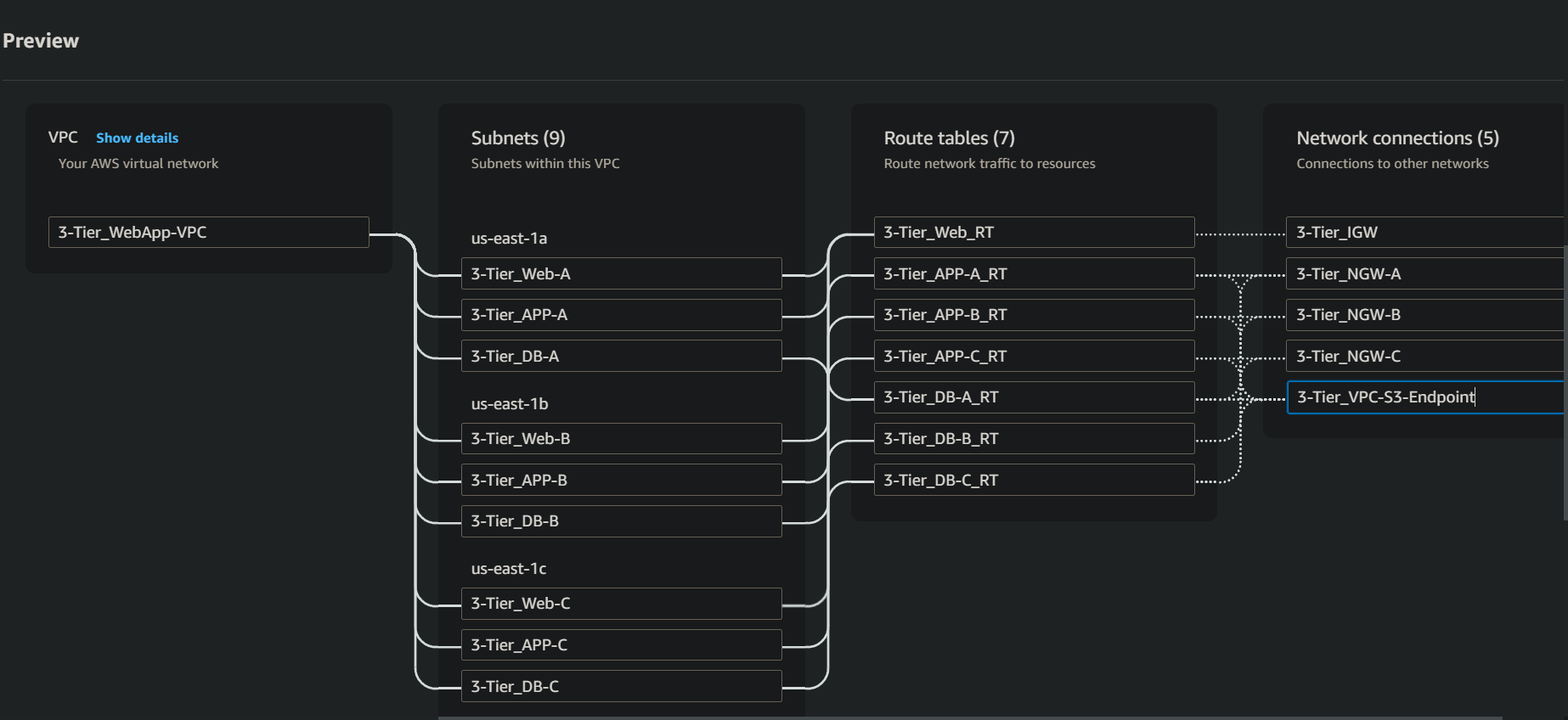
Step 1: Create a S3 bucket with all default settings. Use this S3 bucket to deploy code to the server.

Step 2: Create IAM Role for EC2 instance. Use following permissions



**Part 2: Networking & Security**

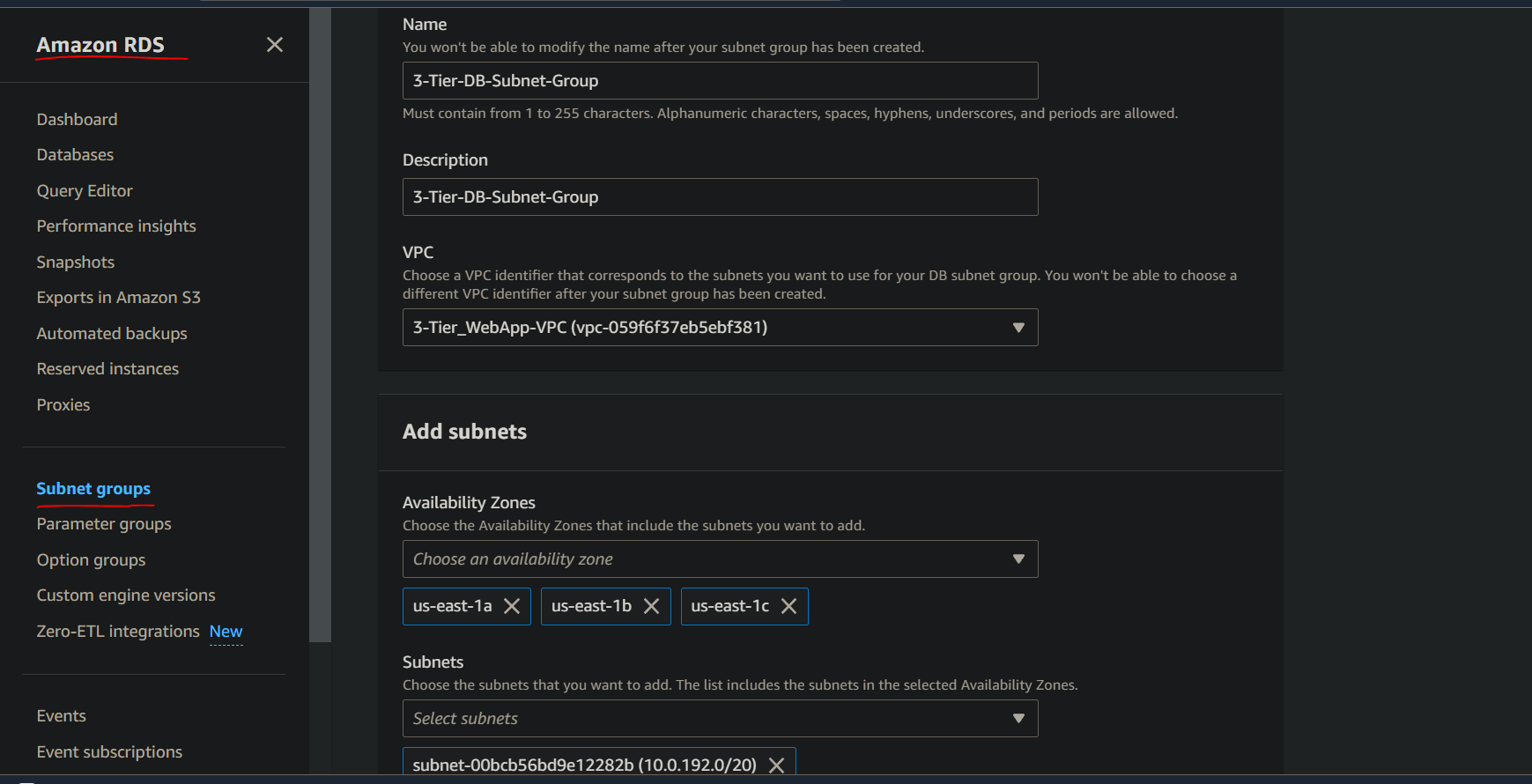
Step 1: Create following resources in VPC & more.



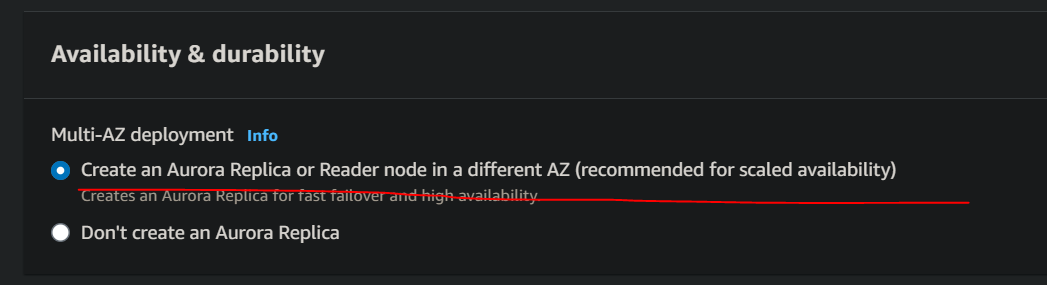
Step 2: Create 5 Security groups for External Facing Load balancer, For Web tier EC2, for Internal Facing Load balancer, For Private instances (use port 4000 for private instance SG) & for DB. Use SG of previously created as source for security group 2, 3, 4 & 5.

**Part 3: Database Deployment**

Step 1: Create DB subnet group



Step 2: Create DB (AuroraSQL). Use this option



**Part 4: App Tier Instance Deployment**

Step 1: Create App tier EC2 instance. Use appropriate VPCs, subnets & SGs. Use IAM role created.

Connect to the instance & run following commands for creating mysql.

sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm

sudo dnf install mysql80-community-release-el9-1.noarch.rpm -y

sudo dnf install mysql-community-server -y

sudo systemctl start mysqld

Step 2: Copy DB writer instance endpoint & use it to run “mysql -h YourEndpoint -u YourUsername -p”. Mysql will be open connected to that instance. Now create a database, table, & some values.

Step 3: Copy dbconfig file from my github repo – code>apptier>dbconfig . Fill the details & upload it to S3 bucket. Also copy app tier folder there.

Run the commands:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash

source ~/.bashrc

nvm install 16

nvm use 16

npm install -g pm2

cd ~/

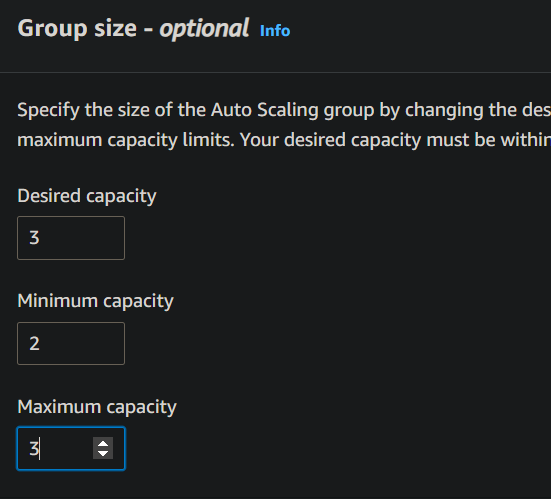
aws s3 cp s3://BUCKET\_NAME/app-tier/ app-tier --recursive

**Step 4: Internal Load Balancing and Auto Scaling**

Step 1: Create image of App Server 1.

Step 2: Create TG. Then create ALB (both for App-Tier). Then create launch template using the image.

Step 3: Create ASG for App-Tier. Use these values.



**Step 5: Web Tier Instance Deployment**

Step 1: Edit internal load balancer dns into nginx config file (code folder) & then deploy nginx file & web-tier folder into s3 bucket.

Step 2: Create Web server instance.

Connect to it & run following commands:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash

source ~/.bashrc

nvm install 16

nvm use 16

cd ~/

aws s3 cp s3://3tier-codedeploy-bucket/web-tier/ web-tier --recursive

cd ~/web-tier

npm install

npm run build

sudo amazon-linux-extras install nginx1 -y

cd ~/

aws s3 cp s3://BUCKET\_NAME/web-tier/ web-tier --recursive

sudo rm nginx.conf

sudo aws s3 cp s3://3tier-codedeploy-bucket/nginx.conf .

sudo service nginx restart

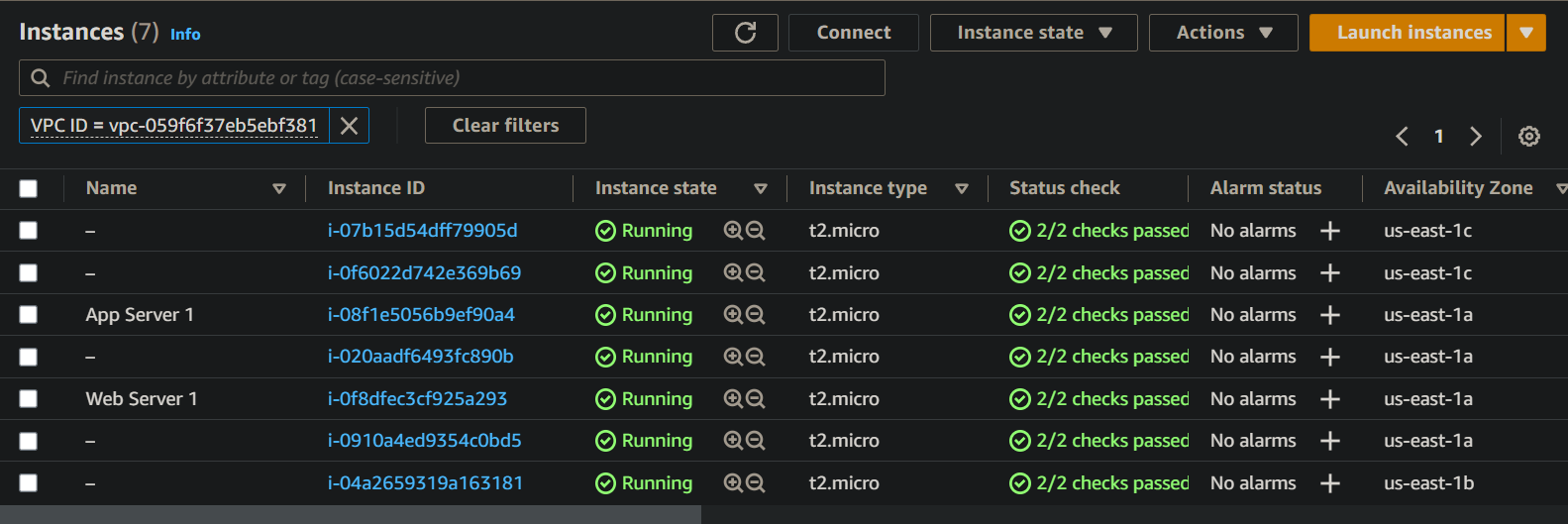
chmod -R 755 /home/ec2-user

sudo chkconfig nginx on

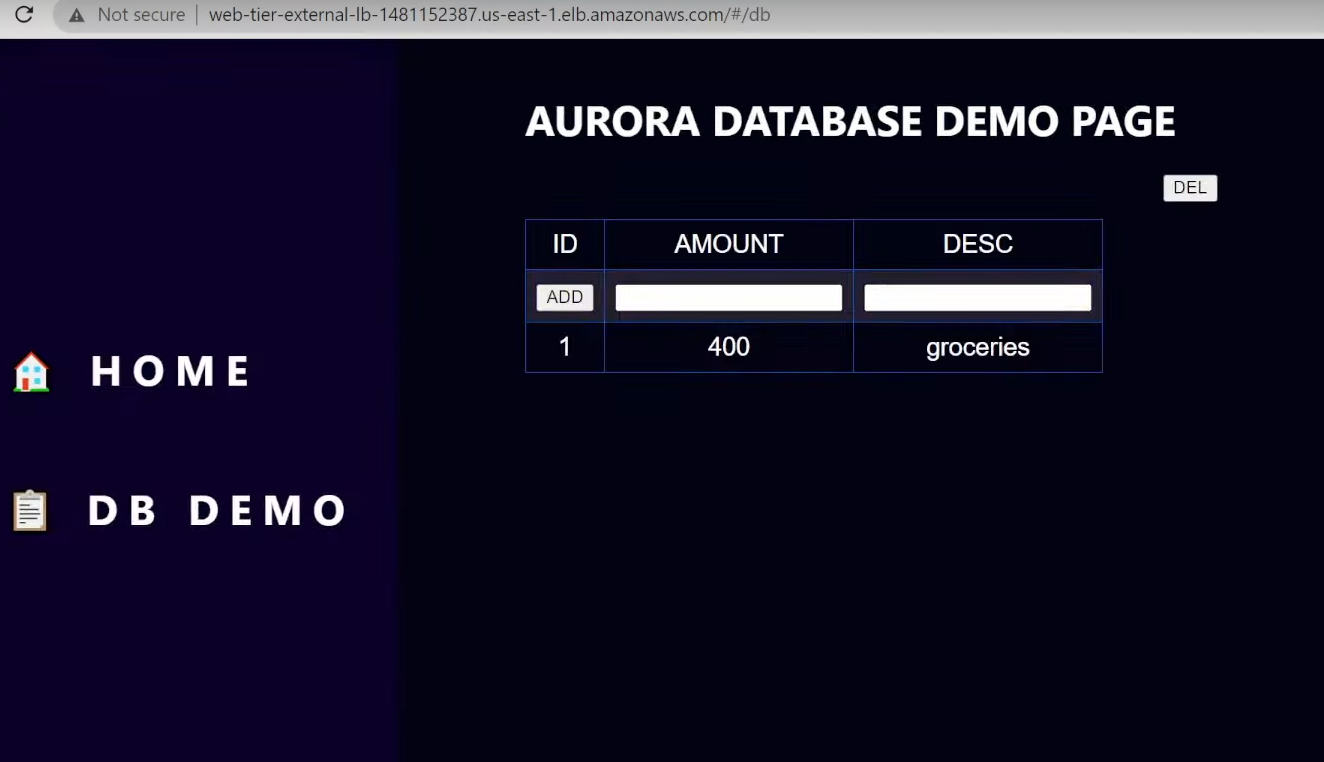
Step 3: Create Image of Web Server. Create TG, ALB, Launch template & ASG for Web Server.

Step 4: Now use Public ALB DNS to use website.

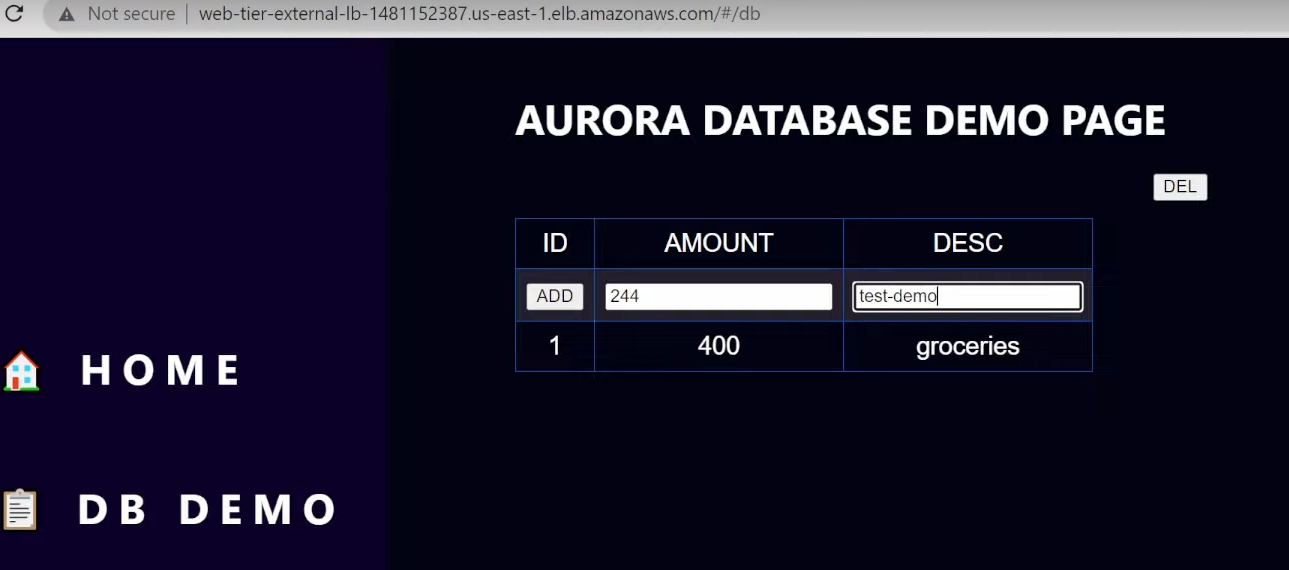
Instances spun up as a result of all the ASGs at work



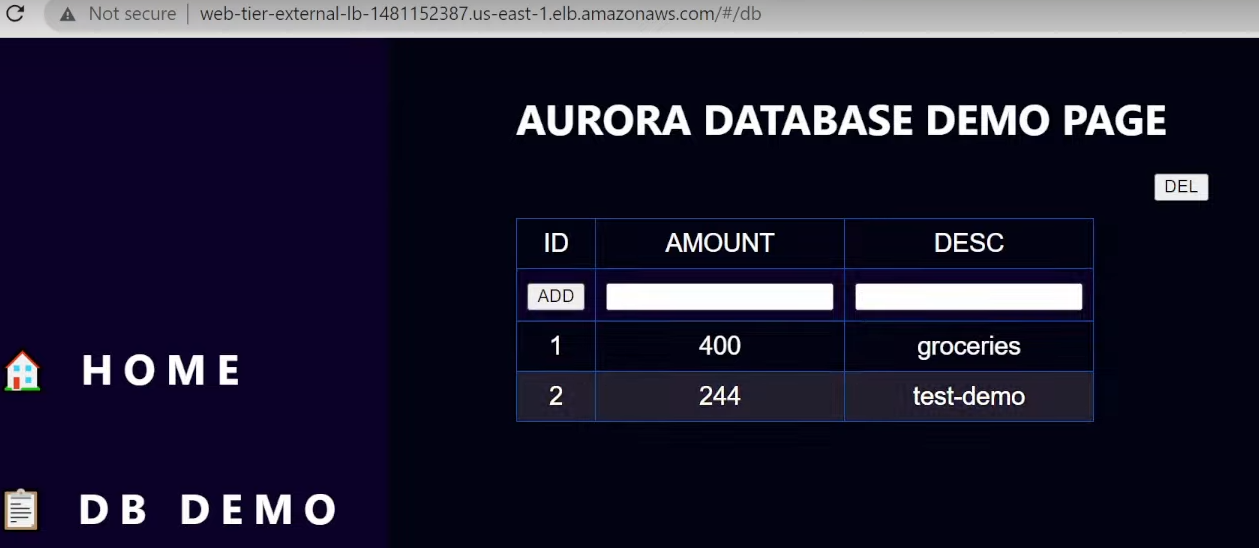
**Outcome:**



Now lets add some values



Values Added



Resources to Cleanup:

