

# **Three-Tier Architecture Deployment on AWS**

## **Index**

### **Part 0: Setup**

#### **Objectives:**

1. Download Code from GitHub Repository
2. IAM EC2 Instance Role Creation
3. S3 Bucket Creation

### **Part 1: Networking and Security**

#### **Objectives:**

1. Create VPC
2. Configure Subnets
3. Setup Route Tables
4. Attach Internet Gateway
5. Setup NAT Gateway
6. Configure Security Groups

### **Part 2: Database Deployment**

#### **Objectives:**

1. Deploy Database Layer
2. Configure Subnet Groups
3. Implement Multi-AZ Database Configuration

## **Part 3: App Tier Instance Deployment**

### **Objectives:**

1. Create App Tier Instance
2. Configure Software Stack for Node.js Application
3. Setup Database Schema
4. Test Database Connectivity

## **Part 4: Internal Load Balancing and Auto Scaling**

### **Objectives:**

1. Create AMI of App Tier Instance
2. Create a Launch Template
3. Configure Auto Scaling
4. Deploy Internal Load Balancer

## **Part 5: Web Tier Instance**

### **Deployment Objectives:**

1. Update NGINX Configuration Files
2. Create a Web Tier Instance
3. Configure Software Stack for NGINX and React.js Website

## **Part 6: External Load Balancer and**

### **Auto Scaling Objectives:**

1. Create AMI of Web Tier Instance
2. Create a Launch Template
3. Configure Auto Scaling
4. Deploy External Load Balancer

## Introduction

This documentation outlines the steps to deploy a three-tier architecture on Amazon Web Services (AWS). The architecture consists of separate layers for presentation, application, and data, ensuring scalability, flexibility, and security.

### **Technologies, Services, and Tools Used:**

- GitHub
- AWS Cloud (Amazon Web Services)
- EC2 (Elastic Compute Cloud)
- Security Groups
- AWS RDS (Relational Database Service)
- Route Table
- Internet Gateway
- NAT Gateway
- Nginx
- Elastic Load Balancer
- Auto Scaling Group

# Part 0: Setup

## Step-by-Step Guide:

### 1. Download Code from GitHub Repository:

- Clone or download the project code from the GitHub repository provided.

```
git clone https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.git
```

### 2. IAM EC2 Instance Role Creation:

- Create an IAM role for EC2 instances with necessary permissions for accessing AWS services.
- Attach 2 policy –
  - AmazonSSMManagedInstanceCore**
  - AmazonS3ReadOnlyAccess**

The screenshot shows the AWS IAM console with the 'EC2InstanceRoleForWorkshop' role selected. The 'Summary' tab displays basic information: Creation date (May 18, 2024, 12:44 (UTC+05:30)), Last activity (27 minutes ago), ARN (arn:aws:iam::378064563647:role/EC2InstanceRoleForWorkshop), and Instance profile ARN (arn:aws:iam::378064563647:instance-profile/EC2InstanceRoleForWorkshop). The 'Permissions' tab shows three attached managed policies: AmazonS3ReadOnlyAccess, AmazonSSMManagedInstanceCore, and AmazonSSMReadOnlyAccess. The bottom of the screen shows the Windows taskbar with various pinned icons and system status.

### 3. S3 Bucket Creation:

- Create an S3 bucket and accept the rest of the default settings to create the bucket. It is used for storing assets or other resources required for the project.

The screenshot shows the AWS S3 console interface. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/theawsthetier-workshop?region=us-east-1&bucketType=general&tab=objects`. The left sidebar has a 'Buckets' section with various options like Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, and Block Public Access settings. Below that is a 'Storage Lens' section with Dashboards, Storage Lens groups, and AWS Organizations settings. A 'Feature spotlight' section is also present. The main content area shows the 'theawsthetier-workshop' bucket. The navigation bar at the top of the content area includes 'Amazon S3 > Buckets > theawsthetier-workshop'. Below it, tabs for 'Objects' (selected), Properties, Permissions, Metrics, Management, and Access Points are visible. The 'Objects' tab shows a list titled 'Objects (3)'. The table has columns for Name, Type, Last modified, and Size. The objects listed are 'app-tier/' (Folder), 'nginx.conf' (conf), and 'web-tier/' (Folder). Action buttons for Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload are available above the table. A search bar and pagination controls are also present. The bottom of the screen shows the Windows taskbar with various pinned icons and system status information.

Name	Type	Last modified	Size
app-tier/	Folder	-	-
nginx.conf	conf	May 18, 2024, 18:45:25 (UTC+05:30)	-
web-tier/	Folder	-	-

# Part 1: Networking and Security

## Step-by-Step Guide:

### 1. Create VPC:

- Set up a Virtual Private Cloud (3twebapp-demo) to isolate the network environment.
- Define CIDR block (10.0.0.0/16)

The screenshot shows the AWS VPC Details page for a VPC named 'vpc-06748081b2e900eb2 / theawsthetierworkshop'. The VPC ID is 'vpc-06748081b2e900eb2'. The State is 'Available'. The Default VPC is 'No'. The IPv4 CIDR is '10.0.0.0/16'. The Network Address Usage metrics are disabled. The DNS resolution is 'Enabled'. The Main route table is 'rtb-094255f4da233644'. The Main network ACL is 'acl-095457fd5a401952d'. The IPv6 pool is '-' and the IPv6 CIDR (Network border group) is '-'. The Owner ID is '378064363647'. Below the details, there are tabs for Resource map, CIDRs, Flow logs, Tags, and Integrations. The Resource map section shows links to VPC, Subnets (6), Route tables (4), and Network interfaces (Connect). The bottom of the screen shows the Windows taskbar with various icons and the date/time as 19-05-2024.

### 2. Configure Subnets :-

3. Created 2 public, 4 private subnets and 2 Availability Zone within the VPC including CIDR Block.
4. Private DB means (For Database)

**Subnets (7) Info**

Name	Subnet ID	State	VPC	IPv4 CIDR
private-app-subnet-az-2	subnet-04248a75c7a937847	Available	vpc-06748081b2e900eb2   the...	10.0.4.0/24
my	subnet-0f6b622664ddbc008	Available	vpc-064bb34e7c9cc78f0	172.31.0.0/20
private-db-subnet-az-1	subnet-09eff14a935d12e65	Available	vpc-06748081b2e900eb2   the...	10.0.5.0/24
private-db-subnet-az-2	subnet-0616bc4247d1da4d3	Available	vpc-06748081b2e900eb2   the...	10.0.6.0/24
public-web-subnet-az-2	subnet-0a4159526ea317f6e	Available	vpc-06748081b2e900eb2   the...	10.0.2.0/24
private-app-subnet-az-1	subnet-074c9cd72c9650bbc	Available	vpc-06748081b2e900eb2   the...	10.0.3.0/24
public-web-subnet-az-1	subnet-08cc94935569d1e6f	Available	vpc-06748081b2e900eb2   the...	10.0.1.0/24

Select a subnet

### 3 Setup Route Tables:

- Configured 2 route tables (Private-Route-Table & Public-Route-Table) to control the routing of network traffic.

**Route tables (5) Info**

Name	Route table ID	Explicit subnet assoc...	Edge associations	Main	VPC
Private-RT-AZ-2	rtb-0fe955d28023e3d5e	subnet-04248a75c7a937...	-	No	vpc-06748081b2e900eb2   t...
-	rtb-07e8dc27e12f16137	-	-	Yes	vpc-064bb34e7c9cc78f0
PRIVATE-RT-AZ-1	rtb-0f419803b8b0fd2cf	subnet-074c9cd72c9650...	-	No	vpc-06748081b2e900eb2   t...
PublicRouteTable	rtb-09d0ae4b7c6c12cc5	2 subnets	-	No	vpc-06748081b2e900eb2   t...
-	rtb-094255f4ada233644	-	-	Yes	vpc-06748081b2e900eb2   t...

Select a route table

## 4. Create and Attach Internet Gateway:

Create and Attach an Internet Gateway to the VPC for public internet access.

The screenshot shows the AWS VPC dashboard with the 'Internet gateways' section selected. The main pane displays a list of two Internet gateways:

Name	Internet gateway ID	State	VPC ID	Owner
three-tier-igw	igw-058e7034af023b856	Attached	vpc-06748081b2e900eb2   theawsthre...	378064363647
-	igw-0976b2069e79d8a7a	Attached	vpc-064bh34e7c9cc78f0	378064363647

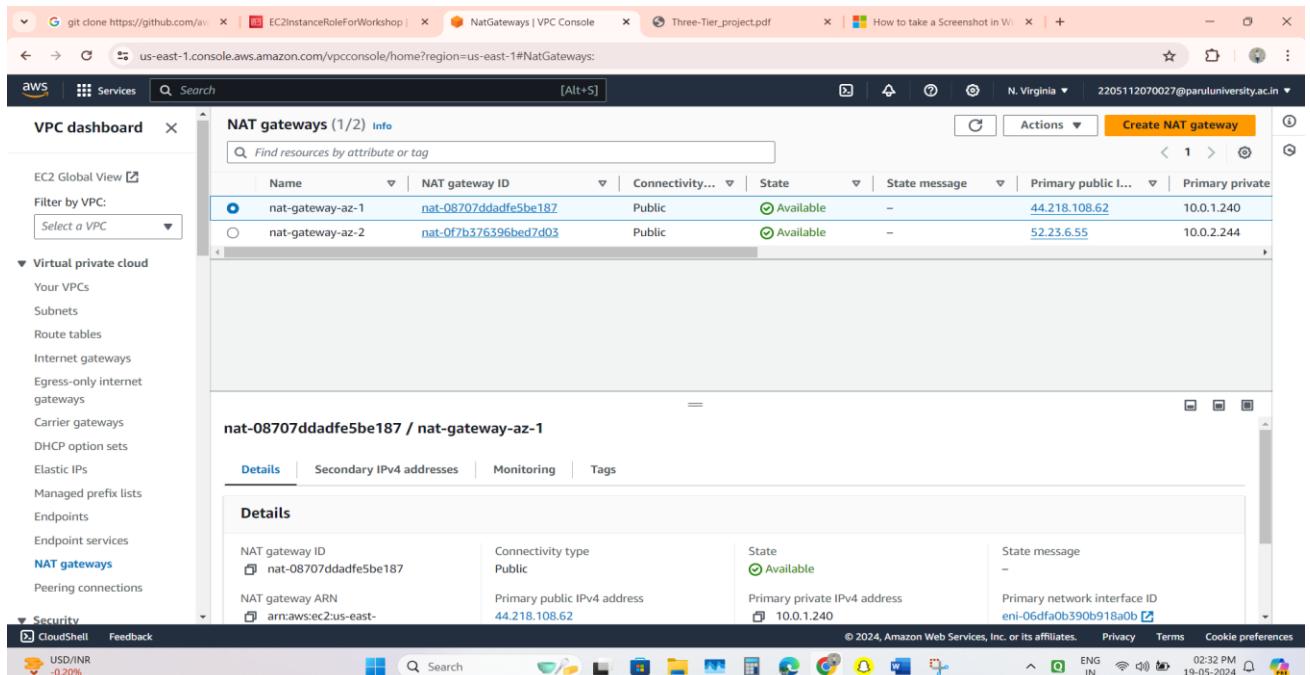
The details for the first Internet gateway (igw-058e7034af023b856) are shown in the expanded view:

Internet gateway ID	State	VPC ID	Owner
igw-058e7034af023b856	Attached	vpc-06748081b2e900eb2   theawsthre...	378064363647

The bottom of the screen shows the Windows taskbar with various pinned icons and system status information.

## 5 Setup NAT Gateway:

- Deploy a NAT Gateway for private subnet instances to access the internet.
- While creating we will going to choose **Public Subnet AZ-1** and for Second NAT Gateway **Public Subnet AZ-2** in which to create the NATGateway.
- In Both NAT Gateways we will assign an Elastic IP.



The screenshot shows the AWS VPC Dashboard with the 'NAT gateways' section selected. There are two entries in the table:

Name	NAT gateway ID	Connectivity...	State	Primary public I...	Primary private
nat-gateway-az-1	nat-08707ddadfe5be187	Public	Available	44.218.108.62	10.0.1.240
nat-gateway-az-2	nat-0f7b376396bed7d03	Public	Available	52.23.6.55	10.0.2.244

Below the table, the details for the first NAT gateway (nat-08707ddadfe5be187) are shown:

Details			
NAT gateway ID nat-08707ddadfe5be187	Connectivity type Public	State Available	State message -
NAT gateway ARN arn:aws:ec2:us-east-	Primary public IPv4 address 44.218.108.62	Primary private IPv4 address 10.0.1.240	Primary network interface ID eni-06dfa0b390b918a0b

## 6 Configure Security Groups:

- Defined 5 security groups to control inbound traffic to EC2 instances, RDS, and other resources.

The screenshot shows the AWS VPC Console interface. On the left, there is a navigation sidebar with various VPC-related options like Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, Endpoints, Endpoint services, NAT gateways, and Peering connections. Under the 'Security' section, 'Network ACLs' and 'Security groups' are listed, with 'Security groups' being the active tab. The main content area displays a table titled 'Security Groups (8)'. The table has columns for Name, Security group ID, Security group name, VPC ID, and Description. The rows list eight security groups: default, default, InternetFacingLoadBalancerSG, WebTierInstancesSG, InternalLoadBalancerSG, PrivateAppInstancesSG, launch-wizard-2, and DatabaseTierInstancesSG. Each row also includes a checkbox and an 'Actions' button. At the top right of the table, there are buttons for 'Create security group', 'Export security groups to CSV', and a search bar. The browser's address bar shows the URL: us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#SecurityGroups. The status bar at the bottom indicates the date and time as 19-05-2024 02:38 PM.

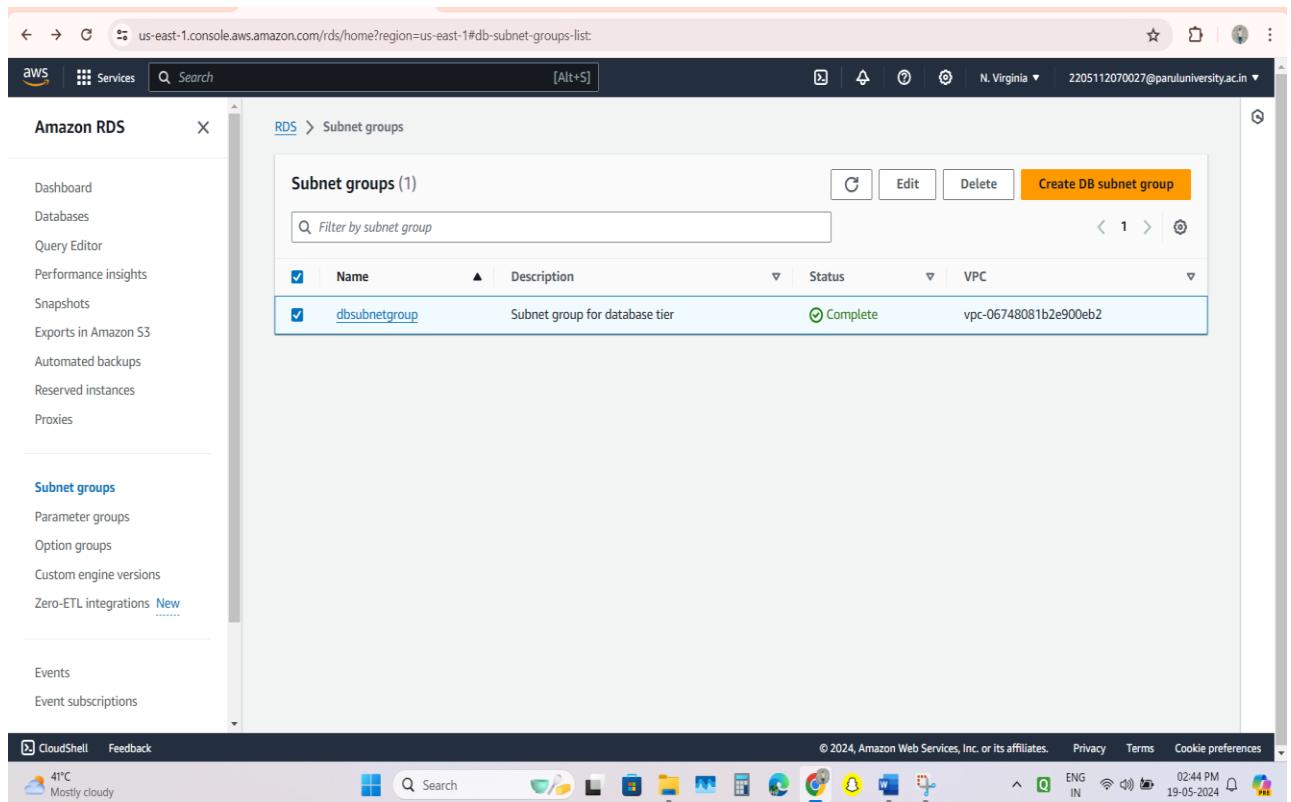
Name	Security group ID	Security group name	VPC ID	Description
-	sg-0cd5b8dadf76e43ce	default	vpc-06748081b2e900eb2	default VPC sec
-	sg-04e96a6a87641058b	default	vpc-064bb34e7c9cc78f0	default VPC sec
-	sg-09f9a0951befd0a66	InternetFacingLoadBalancerSG	vpc-06748081b2e900eb2	Security group f
-	sg-0a546a9b1ac1d4859	WebTierInstancesSG	vpc-06748081b2e900eb2	Security group f
-	sg-05031d5703dad513e	InternalLoadBalancerSG	vpc-06748081b2e900eb2	Security group f
-	sg-010fbe7eaf620c544	PrivateAppInstancesSG	vpc-06748081b2e900eb2	7Sc8F4pJN4ooN
-	sg-0b275eb24ada7a304	launch-wizard-2	vpc-064bb34e7c9cc78f0	launch-wizard-2
-	sg-023cac06924d4b9d2	DatabaseTierInstancesSG	vpc-06748081b2e900eb2	Security group f

## Part 2: Database Deployment

### Step-by-Step Guide:

#### 1. Creating Subnet Group:

- After Going into the RDS service Click Subnet Group to create one and add **2 Private-DB-Subnet (AZ-1 & AZ-2)** For The Database Tier for Added in Subnet Groups
- Make Sure to select Both Availability Zones (**1a & 1b**) in the Creation of the Subnet Group to ensure High Availability.



The screenshot shows the AWS RDS Subnet groups list page. The URL is [us-east-1.console.aws.amazon.com/rds/home?region=us-east-1#db-subnet-groups-list](https://us-east-1.console.aws.amazon.com/rds/home?region=us-east-1#db-subnet-groups-list). The page displays a table titled "Subnet groups (1)". The table has columns: Name, Description, Status, and VPC. One row is shown: "dbsubnetgroup" with the description "Subnet group for database tier", status "Complete", and VPC "vpc-06748081b2e900eb2". There is a "Create DB subnet group" button at the top right of the table. The left sidebar shows navigation links for Dashboard, Databases, Query Editor, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, Reserved instances, Proxies, and Subnet groups. The Subnet groups section is expanded, showing Parameter groups, Option groups, Custom engine versions, Zero-ETL integrations (with a "New" link), Events, and Event subscriptions. The bottom of the screen shows the Windows taskbar with various pinned icons and system status information.

Name	Description	Status	VPC
dbsubnetgroup	Subnet group for database tier	Complete	vpc-06748081b2e900eb2

## Deploying Database

1. Navigate to Databases on the left-hand side of the RDS dashboard and click Create Database.
2. We will now go through several configuration steps. Start with a **Standard created** for this **MySQL-Compatible Amazon Aurora** database. Leave the rest of the defaults in the **Engine options** as default.
3. Under the Templates section, select the Dev/Test since this is not being used for production now.
4. Under Settings Database cluster identifier, keep the default name database 1. We will keep the username as 'admin', set a password to 'welcome-db.'
5. Under the Cluster storage configuration section, we will keep Aurora Standard and keep the default option under Instance Configuration. Next, under Availability and Durability, we will keep the option to create an Aurora Replica or reader node in a different availability zone as recommended along with our VPC under Connectivity.
6. We will also choose the subnet group we created earlier and select no for public access.

Now, let us set the security group we created for the database layer. We will not make any changes under password authentication because password authentication is always on. Click Create to create the database.

The screenshot shows the AWS RDS console with the 'Databases' section selected. A database cluster named 'database-1' is listed, containing two instances: 'writer instance' and 'reader instance', both running Aurora MySQL. The cluster is located in the us-east-1 region with 2 instances, db.r6g.2xlarge, and 2 information nodes.

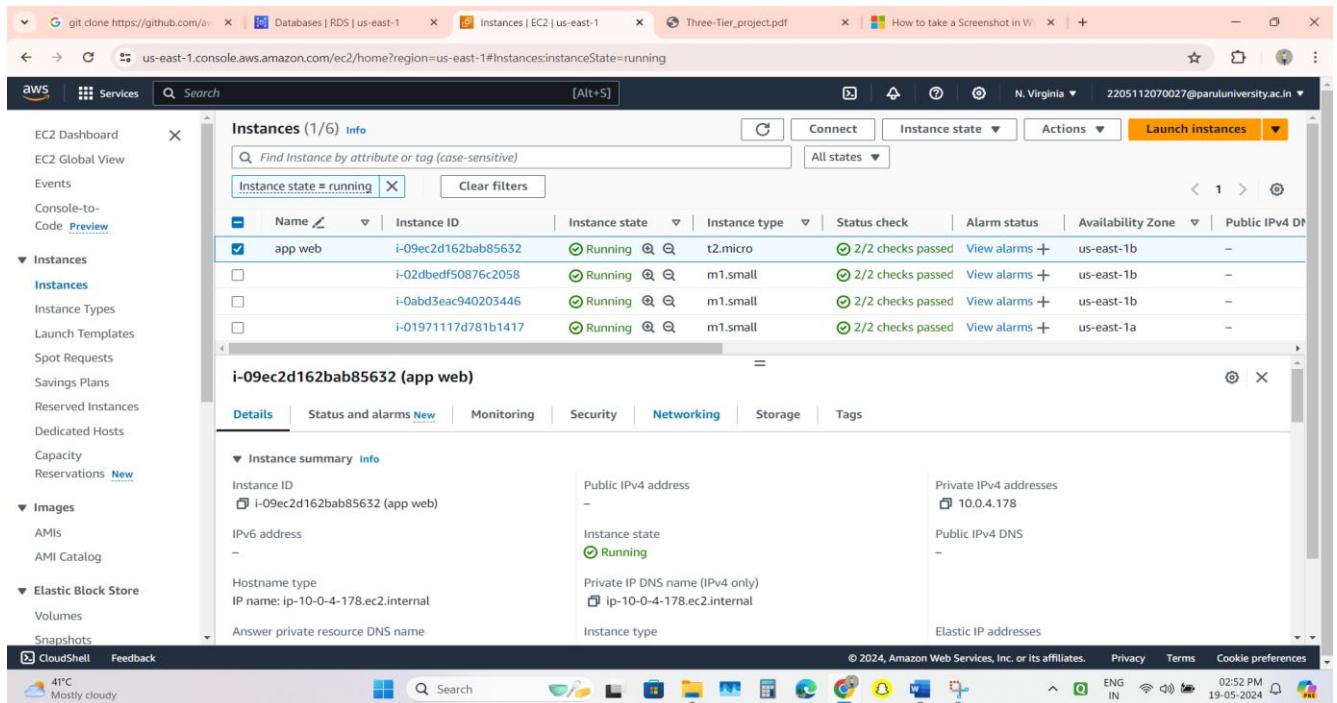
DB identifier	Status	Role	Engine	Region & AZ	Size	Recommend
database-1	Available	Regional cluster	Aurora MySQL	us-east-1	2 instances	2 Information
database-1-instance-1	Available	Writer instance	Aurora MySQL	us-east-1a	db.r6g.2xlarge	1 Information
database-1-instance-1-us-east-1b	Available	Reader instance	Aurora MySQL	us-east-1b	db.r6g.2xlarge	1 Information

## Part 3: App Tier Instance Deployment

### Step-by-Step Guide:

#### 1. Create App Tier Instance:

- Launch an EC2 instance for hosting the application layer.
- Select **Amazon Linux 2 AMI** and select free tier eligible **T.2 micro** as instance type.
- Next, we will proceed without a key pair for architecture.
- In Subnet select **Private-Subnet-Az1** and under SG select **Private-instance-SG**
- In an additional setting attach the role which we have created at the start.
- At last Launch the Instance.



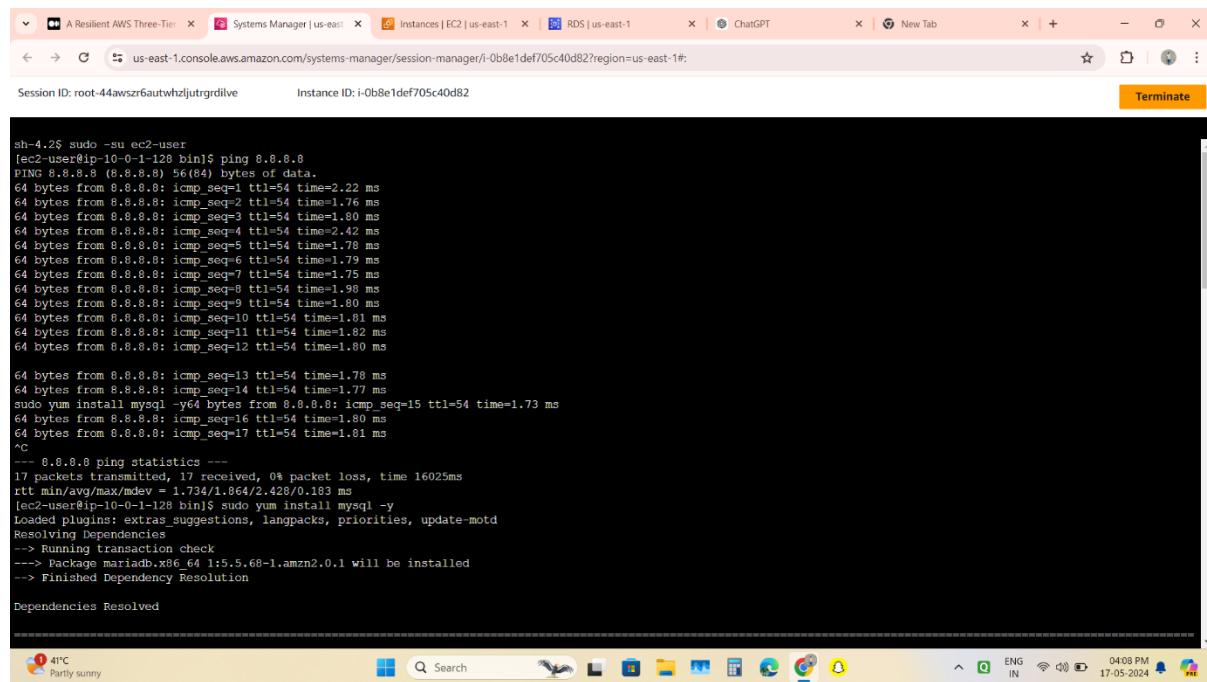
## 2 Configure Software Stack for Node.js Application:

- First Connect the instance through Session Manager.
- When you first connect to your instance, you will be logged in as ssm-user which is the default user. Switch to ec2-user

**Sudo -su ec2-user**

- Now check that you are connected to the Internet By

**Ping 8.8.8.8**



The screenshot shows a web browser window with several tabs open. The active tab is titled "Session ID: root-44awszr6autwhzljutrgdilive" and "Instance ID: i-0b8e1def705c40d82". The terminal window displays the following command and output:

```
sh-4.2$ sudo -su ec2-user
[ec2-user@ip-10-0-1-128 bin]$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=54 time=2.22 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=54 time=1.76 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=54 time=1.80 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=54 time=2.42 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=54 time=1.78 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=54 time=1.79 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=54 time=1.75 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=54 time=1.98 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=54 time=1.80 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=54 time=1.81 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=54 time=1.82 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=54 time=1.80 ms

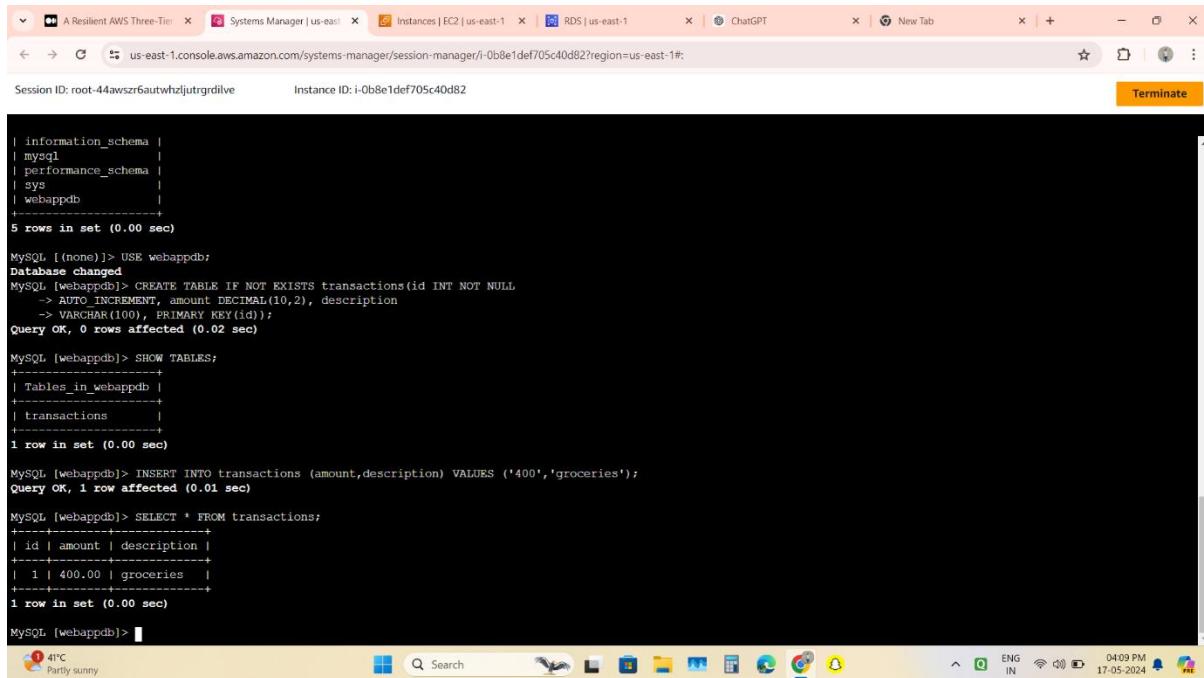
64 bytes from 8.8.8.8: icmp_seq=13 ttl=54 time=1.78 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=54 time=1.77 ms
sudo yum install mysql -y
64 bytes from 8.8.8.8: icmp_seq=15 ttl=54 time=1.73 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=54 time=1.80 ms
64 bytes from 8.8.8.8: icmp_seq=17 ttl=54 time=1.81 ms
^C
-- 8.8.8.8 ping statistics --
17 packets transmitted, 17 received, 0% packet loss, time 16025ms
rtt min/avg/max/mdev = 1.734/1.864/2.428/0.183 ms
[ec2-user@ip-10-0-1-128 bin]$ sudo yum install mysql -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package mariadb.x86_64 1:5.5.68-1.amzn2.0.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

The browser taskbar at the bottom shows various icons and system status information, including the date and time (17-05-2024, 04:08 PM), battery level (4%), and network connectivity.

### 3 Setup Database Schema:

- Configure the database schema and tables required by the application.
- Run All these Commands for configuring the Database.



The screenshot shows a browser window with multiple tabs open, including 'A Resilient AWS Three-Tier...', 'Systems Manager | us-east-1', 'Instances | EC2 | us-east-1', 'RDS | us-east-1', and 'ChatGPT'. The main content area is a terminal window titled 'Session ID: root-44awszr6autwhzljutrgdive' and 'Instance ID: i-0b8e1def705c40d82'. The terminal displays the following MySQL session:

```
| information_schema |
| mysql |
| performance_schema |
| sys |
| webappdb |
+-----+
5 rows in set (0.00 sec)

MySQL [(none)]> USE webappdb;
Database changed
MySQL [webappdb]> CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL
-> AUTO_INCREMENT, amount DECIMAL(10,2), description
-> VARCHAR(100), PRIMARY KEY (id));
Query OK, 0 rows affected (0.02 sec)

MySQL [webappdb]> SHOW TABLES;
+-----+
| Tables_in_webappdb |
+-----+
| transactions |
+-----+
1 row in set (0.00 sec)

MySQL [webappdb]> INSERT INTO transactions (amount,description) VALUES ('400','groceries');
Query OK, 1 row affected (0.01 sec)

MySQL [webappdb]> SELECT * FROM transactions;
+----+----+-----+
| id | amount | description |
+----+----+-----+
| 1 | 400.00 | groceries |
+----+----+-----+
1 row in set (0.00 sec)

MySQL [webappdb]>
```

### Configure App Instance

By Running these commands in Session Manager

Step	Command/Instruction
1. Upload app-tier folder to S3 bucket	<code>aws s3 cp app-tier/ s3://BUCKET_NAME/app-tier/ --recursive</code>
2. Install NVM	<code>curl -o https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh   source</code>
3. Install Node.js and use version 16	<code>nvm install 16</code>
4. Install PM2	<code>nvm use 16</code>
5. Download code from S3 bucket to instance	<code>aws s3 cp s3://BUCKET_NAME/app-tier/ app-tier --recursive</code>
6. Navigate to app directory	<code>cd ./app-tier</code>
7. Install dependencies	<code>npm install</code>
8. Start the app with PM2	<code>pm2 start index.js</code>
9. Check if the app is running	<code>pm2 list</code>
10. View logs if necessary	<code>pm2 logs</code>
11. Configure PM2 startup	<code>pm2 startup</code>
	<code>Copy and paste the command provided by the output in your terminal,</code>

## Test App Tier

Now let's run a couple tests to see if our app is configured correctly and can retrieve data from the database.

To check out our health check endpoint, copy this command into your SSM terminal. This is our simple health check endpoint that tells us if the app is simply running. **sudo curl <http://localhost:4000/health>**

The command responded with the following message: "This is the health check" which means our health check is running correctly as indicated below.



Session ID: avmash-  
Instance ID: i-094879c56a5f11240  
Terminate

```
[ec2-user@ip-10-0-2-113 app-tier]$ sudo curl http://localhost:4000/health
"This is the health check"(ec2-user@ip-10-0-2-113 app-tier)$
```

Next, let's test our database connection. We can do that by hitting the following endpoint locally:

**curl <http://localhost:4000/transaction>**

Received the following response after executing the command:



Session ID: avmash-  
Instance ID: i-094879c56a5f11240  
Terminate

```
[ec2-user@ip-10-0-2-113 app-tier]$ curl http://localhost:4000/transaction
{"result": [{"id": 1, "amount": 400, "description": "groceries"}]}[ec2-user@ip-10-0-2-113 app-tier]$
```

The two above responses indicate that our networking, security, database, and app configurations are correct. Our app layer is fully configured and ready to go.

# Part 5: Internal Load Balancing and Auto Scaling

In this section, we will create an Amazon Machine Image (AMI) of the app tier instance we just created, and use that to set up autoscaling with a load balancer in order to make this tier highly available.

## App Tier AMI

Let's navigate to **Instances** on the left-hand side of the EC2 dashboard. Select the app tier instance we created and under **Actions** select **Image and Templates**. Click **Create Image**.

Let's give the image a name and description and then click **Create image**. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left-hand navigation panel of the EC2 dashboard.

The screenshot shows the AWS EC2 AMI details page. The URL is [us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ImageDetails:imageId=ami-0e84ccf4010c9db84](https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ImageDetails:imageId=ami-0e84ccf4010c9db84). The page displays the 'Image summary for ami-0e84ccf4010c9db84'. Key details include:

- AMI ID:** ami-0e84ccf4010c9db84
- Image type:** machine
- Platform details:** Linux/UNIX
- Root device type:** EBS
- Owner account ID:** 378064363647
- Architecture:** x86\_64
- Usage operation:** RunInstances
- Source:** 378064363647/WebTier-AMI
- Virtualization type:** hvm
- Creation date:** Sat May 18 2024 19:27:13 GMT+0530 (India Standard Time)
- Kernel ID:** -
- Description:** AMI for the web tier of my service
- Product codes:** -
- RAM disk ID:** -
- Deprecation time:** -
- Last launched time:** Sat May 18 2024 19:37:46 GMT+0530 (India Standard Time)
- Block devices:** /dev/xvda=snap-0627e6a8c23c4adcf:8:true:gp2
- Deregistration protection:** Disabled

The left sidebar shows the navigation menu with sections like Instances, Images, and Network & Security. The 'Images' section is currently selected, showing 'AMIs' and 'AMI Catalog'. The bottom of the page includes standard AWS footer links and a weather widget indicating 41°C and mostly cloudy conditions.

## Target Group

While the AMI is being created, let's go ahead and create our target group to use with the load balancer. On the EC2 dashboard, navigate to **Target Groups** under **Load Balancing** on the left-hand side.

Click on **Create Target Group**.

The purpose of forming this target group is to use our load balancer so it may balance traffic across our private app tier instances. Let's select Instances as the target type and give it a name.

Let's set the protocol to **HTTP** and the port to 4000. Remember that this is the port our Node.js app is running on. Select the VPC we've been using thus far, and then change the health check path to be **/health** to indicate the health check endpoint of our app and click **Next**.

We'll **NOT** register any targets for now, so let's just skip that step, click **Next** and create the target group.

## Internal Load Balancer

We're going to create an internal load balancer for our three-tier. On the left-hand side of the EC2 dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

We'll be using an **Application Load Balancer** for our **HTTP** traffic, give it a name (**App-Tier-Internal-lb**), select **Internal** under Scheme, and click the create button for that option.

Let's select the correct network configuration for our VPC and private subnets.

Next, select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created. Let's select it from the dropdown list, and create the load balancer.

The screenshot shows the AWS CloudWatch Metrics interface. A histogram titled 'AWS Lambda Metrics' displays the number of requests per second over a 1-hour period. The x-axis represents time from 10:00 to 11:00, and the y-axis represents the number of requests. The distribution is highly skewed, with most requests occurring in short bursts.

Internal Load Balancer is created with two availability zones.

## Launch Template

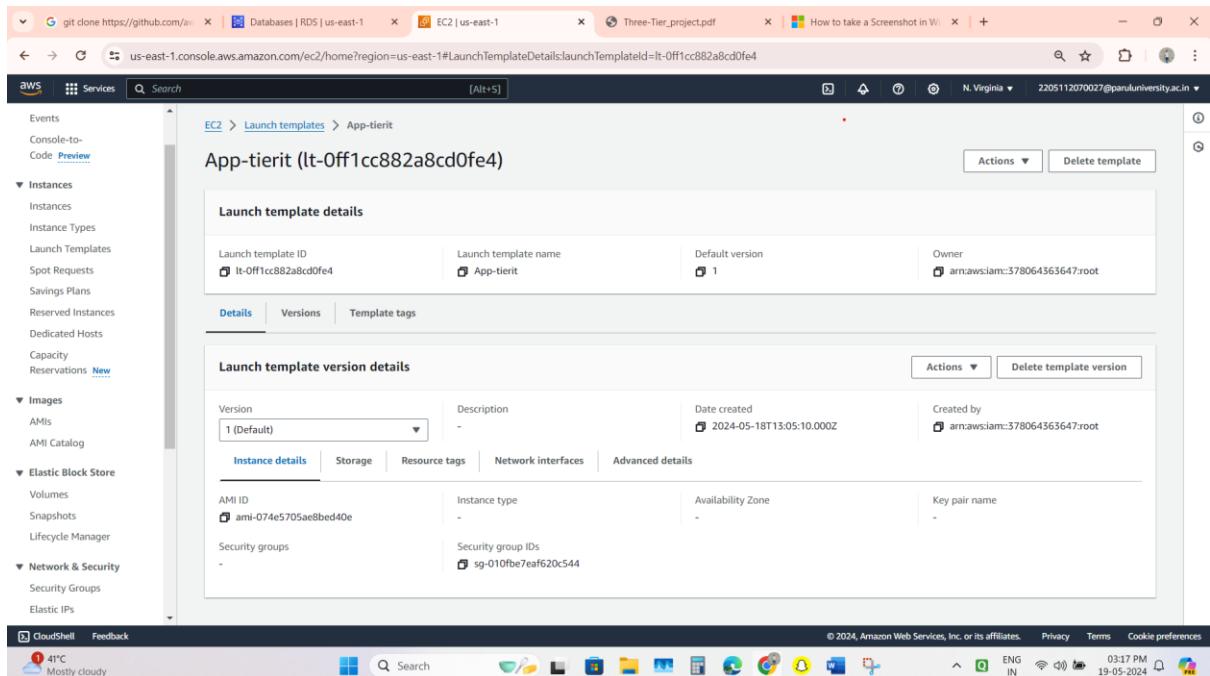
Now, we need to create a Launch template with the AMI we created earlier before we configure Auto Scaling. On the left side of the EC2 dashboard, let's navigate to

**LaunchTemplate** under **Instances** and click **Create Launch Template**.

Name the Launch Template, and then under **Application and OS Images** select **MY AMI** and include the app tier AMI we previously created.

Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network information in the autoscaling group.

Set the correct security group for our app tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.



## Auto Scaling

We will now create the Auto Scaling Group for our app instances. On the left-hand side of the EC2 dashboard, navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto ScalingGroup**.

Let's give our Auto Scaling group a name, and then select the Launch Template we just created and click **Next**.

Next, we're going to select on the **Choose instance launch options** page, set our VPC, and the private instance subnets for the app tier, and continue.

For this next step, we'll attach this Auto Scaling Group to the Load Balancer we just created by selecting the existing load balancer's target group from the dropdown. Then, click next.

We're going to set the desired 2, minimum 2, and maximum 2 capacity of our Auto Scaling **Group size** and **Scaling policies**. Review and then Create Auto Scaling Group.

Let's find out if our internal load balancer and autoscaling group are configured correctly. The autoscaling group will spin up 2 new app tier instances. We can test if this is working correctly by deleting one of our new instances manually and waiting to see if a new instance is booted up to replace it.

The screenshot shows the AWS CloudShell interface with multiple tabs open. The main tab displays the 'Auto Scaling group details' for the 'APPTierASG' group. The 'Details' tab is selected, showing the following configuration:

Auto Scaling group name	Desired capacity	Desired capacity type	Amazon Resource Name (ARN)
APPTierASG	2	Units (number of instances)	arn:aws:autoscaling:us-east-1:378064363647:autoScalingGroup:30ccb652-a907-4d58-93bd-5466f7fb49c:autoScalingGroupName/APPTierASG
Date created	Minimum capacity	Status	Sat May 18 2024 18:53:23 GMT+0530 (India Standard Time)
	Maximum capacity		2

Below the 'Group details' section is the 'Launch template' section, which lists the launch template configuration:

Launch template	AMI ID	Instance type	Owner
lt-off1cc882a8cd0fe4 App-tier	ami-074e5705ae8bed40e	-	arn:aws:siam::378064363647:root
Version	Security groups	Security group IDs	Create time
Default	-	sg-010fbe7ea620c544	Sat May 18 2024 18:35:10 GMT+0530 (India Standard Time)

## Part 6: Web Tier Instance Deployment

In this section, we will deploy an EC2 instance for the web tier and make all necessary software configurations for the NGINX web server and React.js website.

- Update Config File

Before we create and configure the web instances (web tier), let's modify the **application-code/nginx.conf** file from the repo we previously downloaded. First, navigate to your internal load balancer's details page and copy the DNS entry into a notepad and save.

Next, open the folder where the repo was downloaded on your command prompt terminal to update the **application-code/nginx.conf** file. Edit the file using this command:

```
vim nginx.conf
:set nu
```

Scroll down to **line 58** and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer's DNS entry. Save the file by pressing the '**esc** **keypad ,** **shift : + wq**' without the single quotes.

```
 35 # for more information.
 36 include /etc/nginx/conf.d/*.conf;
 37
 38 server {
 39     listen      80;
 40     listen      [::]:443;
 41     server_name ~^;
 42
 43     #health check
 44     location /health {
 45         default_type text/html;
 46         return 200 "HTTP/1.0 200 OK<p>Web Tier Health Check</p>\n";
 47     }
 48
 49     #react app and front end files
 50     location / {
 51         root   /home/ec2-user/web-tier/build;
 52         index index.html index.htm;
 53         try_files $uri /index.html;
 54     }
 55
 56     #proxy for internal lb
 57     location /api/ {
 58         proxy_pass http://internal-App-Tier-Internal-LB-1900078888.us-east-1.elb.amazonaws.com:80/;
 59     }
}
```

Now, let's upload the '**nginx.config**' file and the **application-code/web-tier** folder to the s3 bucket we created for this lab. Navigate back to the Amazon S3 dashboard, click '**Buckets**' on the left hand, and select our bucket | **Upload**.

The screenshot shows the AWS S3 console interface. The left sidebar is titled 'Amazon S3' and includes sections for 'Buckets', 'Access Grants', 'Access Points', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', 'IAM Access Analyzer for S3', and 'Storage Lens'. Under 'Storage Lens', there are links for 'Dashboards', 'Storage Lens groups', and 'AWS Organizations settings'. A 'Feature spotlight' section is also present. The main content area shows the 'theawsthreetier-workshop' bucket. The navigation bar at the top shows the path 'Amazon S3 > Buckets > theawsthreetier-workshop'. Below the path, there are tabs for 'Objects' (which is selected), 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Objects' tab displays a table with three items:

Name	Type	Last modified	Size	Storage class
<a href="#">app-tier/</a>	Folder	-	-	-
<a href="#">nginx.conf</a>	conf	May 18, 2024, 18:45:25 (UTC+05:30)	2.5 KB	Standard
<a href="#">web-tier/</a>	Folder	-	-	-

Below the table, there is a search bar with the placeholder 'Find objects by prefix' and a 'Copy S3 URI' button. The top navigation bar also includes 'Actions', 'Create folder', and 'Upload' buttons.

## Web Instance Deployment

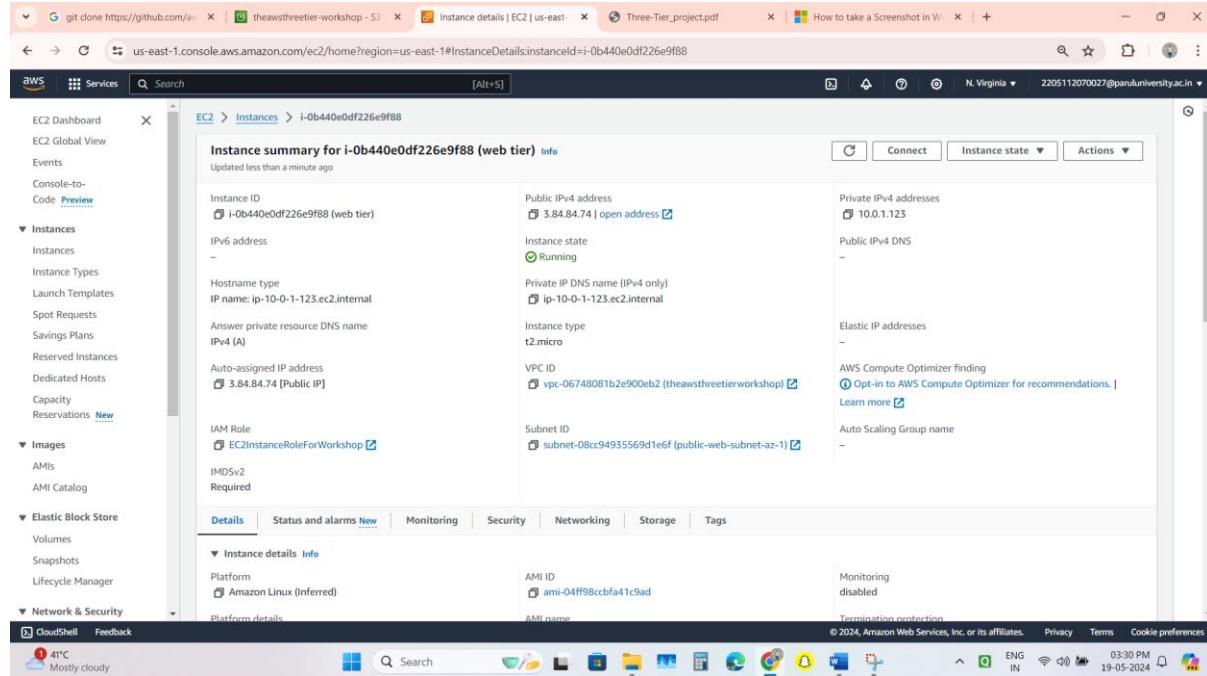
Using the EC2 service dashboard, click on **Instances** on the left-hand side and then **Launch Instances** to start the process.

Let's name our app instance '**Web-Tier**' for the three-tier architecture and select the **Amazon Linux2 AMI** for our application and operating system image.

We'll be using the free tier eligible **T.2 micro** instance type so let's select that to proceed. Even though 'Proceed without a key is (Not recommended)', we'll proceed without a key pair for architecture.

Earlier we created a security group for our public web layer instances, so go ahead and select that alongwith our VPC and the '**public-web-subnet-az-1**' under **Network settings**.

We'll keep the default configuration settings for **Configure storage**, and in advanced details add **IAM role ec2-three-tier-access-role** that we have created at starting, but review the **Summary**, and click **Launch** instance to create the '**appLayer**' instance.



## Connect to Instance

Let's follow the same steps we used to connect to the first app instance and change the user to **ec2-user**. Test connectivity here via ping as well since this instance should have internet connectivity:

```
Sudo -su ec2-user  
Ping 8.8.8.8
```

*Note: If you don't see a transfer of packets then you'll need to verify your route tables attached to the subnet that your instance is deployed in.*

I was able to ping the Google server IP Address 8.8.8.8 from the App Tier instance command prompt terminal successfully.



```
Session ID: avinash-dash0rdielyh0gnd3nm           Instance ID: i-0f2357739c8d9aa63
[...]
dh-4.25:~$ sudo -u ec2-user
[ec2-user@ip-10-0-0-221 ~]$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=1.03 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=1.04 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=1.06 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=1.06 ms
...
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 300ms
rtt min/avg/max/mdev = 1.032/1.054/1.059/0.014 ms
[ec2-user@ip-10-0-0-221 ~]$
```

## Configure Web Instance

We now need to install all of the necessary components needed to run our front-end application. Let's start by installing NVM and node on the instance

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

```
source ~/.bashrc nvm install 16 nvm use 16
```

Now we need to download our web tier code from our s3 bucket:

```
cd ~
```

```
aws s3 cp s3://BUCKET_NAME/web-tier/ web-tier --recursive
```

**Replace [BUCKET\_NAME] with your bucket name.**

Navigate to the web-layer folder and create the build folder for the react app so we can serve our code using the below commands:

```
cd ~/web-tier npm install npm run build
```

NGINX can be used for different use cases like load balancing, content caching etc, but we will be using it as a web server that we will configure to serve our application on port 80, as well as help direct our API calls to the internal load balancer. Let's run the below command to proceed

```
sudo amazon-linux-extras install nginx1 -y
```

We will now have to configure NGINX. Navigate to the Nginx configuration file with the following commands and list the files in the directory:

```
cd /etc/nginx
```

Is

Let's update the '**nginx.conf**' file with the one we uploaded to S3 bucket. We'll remove the file and replace the bucket name below:

```
sudo rm nginx.conf
```

```
sudo aws s3 cp s3://BUCKET_NAME/nginx.conf .
```

The bucket name should be replaced.

Let's restart Nginx with the following command:

```
sudo service nginx restart
```

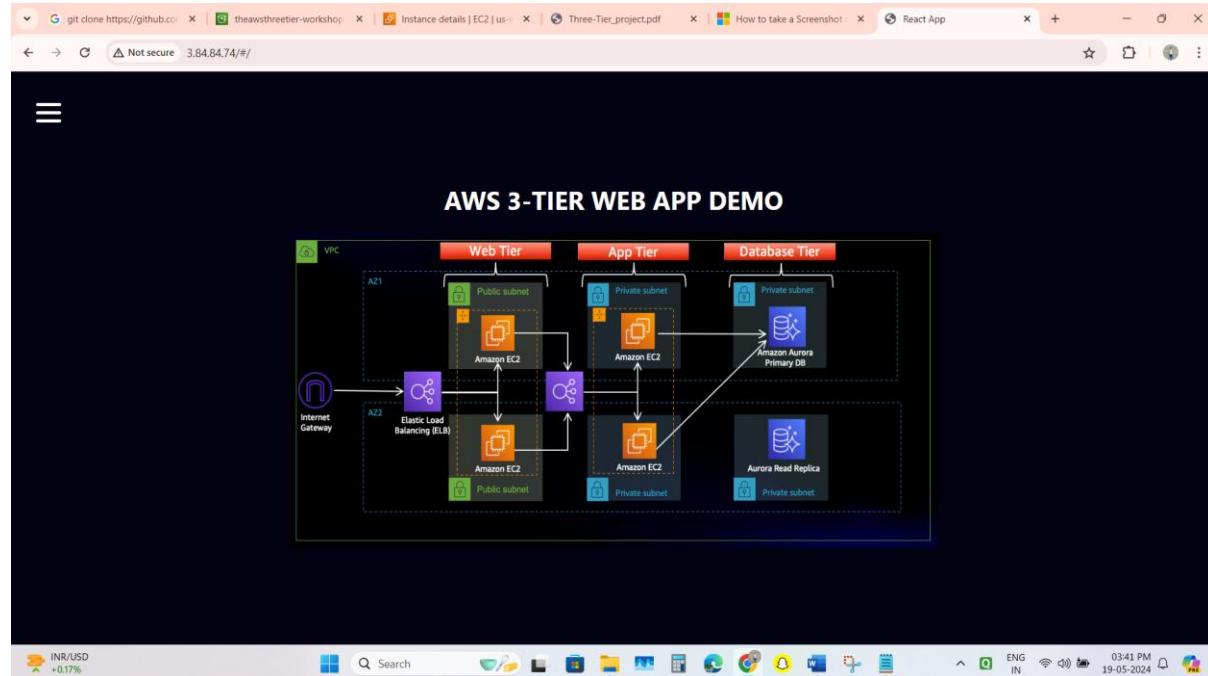
Let's make sure Nginx has permission to access our files by executing the command:

```
chmod -R 755 /home/ec2-user
```

And to make ensure the service starts on boot, run this command:

```
sudo chkconfig nginx on
```

Now let's copy and plug in the public IP of our web tier instance to see our website. The public IP can be found on the App Tier Instance details page on the EC2 dashboard. Voila, the website is working correctly.



Let's do the same for our database tier and if it's connected and working correctly.

AURORA DATABASE DEMO PAGE

ID	AMOUNT	DESC
1	400	groceries
2	0	
3	500	wheat
4	0	
5	0	
6	0	
7	0	
8	500	dff
9	500	Ajay

## Part 7: External Load Balancer and Auto Scaling

In this section of the workshop, we will create an Amazon Machine Image (AMI) of the web tier instance we just created, and use that to set up autoscaling with an external facing load balancer in order to make this tier highly available.

### Web Tier AMI

Let's go to Instances on the left-hand side of the EC2 dashboard. Select the web tier instance we created and under Actions select Image and Templates. Click Create Image.

We'll give the image a name and description and then click Create image. This will take a few minutes, but if you want to monitor the status of image creation you can see it by

clicking AMIs under Images on the left-hand navigation panel of the EC2 dashboard.

The screenshot shows the AWS EC2 Launch Template details page. The left sidebar includes links for EC2 Dashboard, EC2 Global View, Events, Console-to-Code Preview, Instances, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, CloudShell, and Feedback. The main content area shows the 'Launch template details' section with the launch template ID lt-08f3e2976d939fcf3, name web-tier-launch-template, and default version 1. It also shows the 'Launch template version details' section for version 1 (Default), which includes the AMI ID ami-0e84ccf4010c9db84, instance type t2.micro, availability zone us-east-1a, and security group sg-0a546a9b1ac1d4859.

## Target Group

On the EC2 dashboard let's navigate to Target Groups under Load Balancing on the left-hand side. Click on Create Target Group.

The purpose of forming this target group is to use our load balancer so it may balance traffic across our public web tier instances. Select Instances as the target type and give it a name.

Let's set the protocol to HTTP and the port to 80. Remember this is the port NGINX is listening on.

Select the VPC we've been using thus far, and then change the health check path to be /health. Click Next.

Go ahead and create the target group. We are NOT going to register any targets for now.

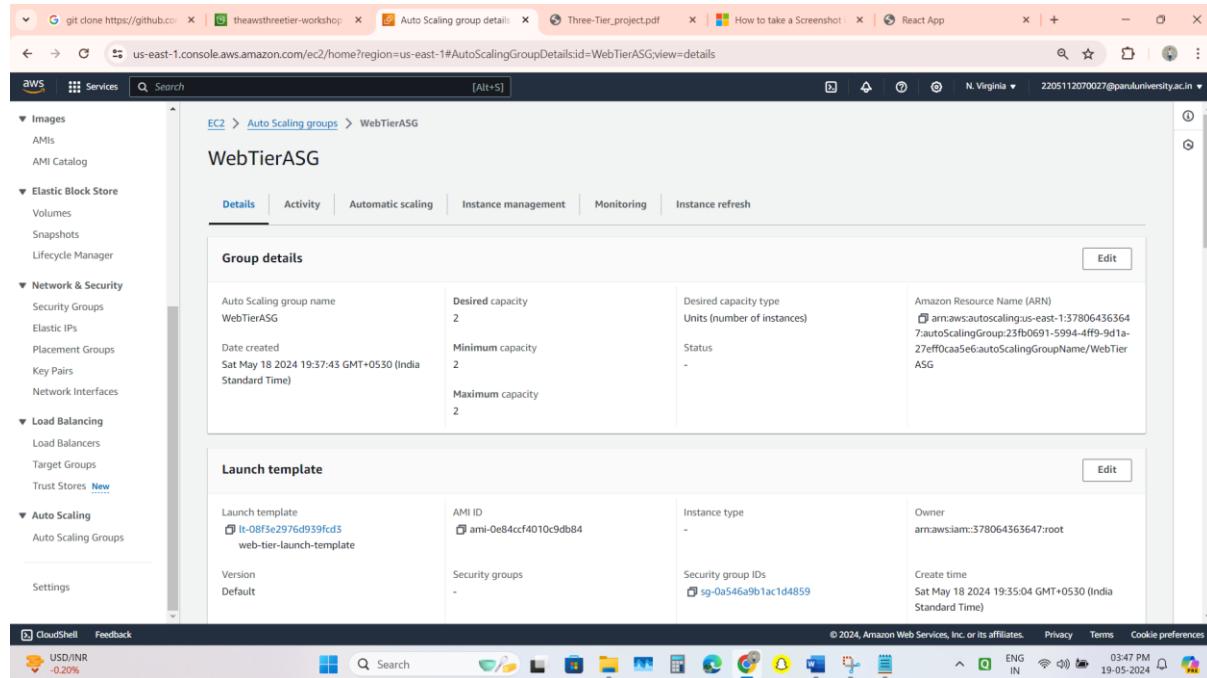
## Internet Facing Load Balancer

On the left-hand side of the EC2 dashboard select Load Balancers under Load Balancing and click Create Load Balancer.

Select Application Load Balancer for our HTTP traffic. Click the Create button for that option. After giving the load balancer a name, be sure to select internet facing since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.

Select the two public subnets to proceed.

Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our target group that we just created, so select it from the dropdown, and create the load balancer.



The screenshot shows the AWS CloudWatch Metrics console with a search bar at the top containing the query: `aws cloudwatch metrics filter @version >= 1.0 AND @namespace = "AWS/CloudWatchMetricsInsights" AND @metricName = "MetricsSearchResults" AND @label = "MetricsSearchResults" AND @statistic = "Sum" AND @unit = "Count" AND @value >= 1`. Below the search bar, there are tabs for `Metrics`, `Logs`, and `CloudWatch Metrics Insights`. The `CloudWatch Metrics Insights` tab is selected. On the left, there's a sidebar with navigation links for `Metrics`, `Logs`, `CloudWatch Metrics Insights`, `CloudWatch Metrics Insights API`, and `CloudWatch Metrics Insights Metrics`. The main content area displays a table with columns: `Time`, `MetricsSearchResults`, and `Count`. There are three rows of data:

Time	MetricsSearchResults	Count
2024-05-19T19:35:04Z	MetricsSearchResults:{"@version":1,"@namespace":"AWS/CloudWatchMetricsInsights","@metricName":"MetricsSearchResults","@label":"MetricsSearchResults","@statistic":"Sum","@unit":"Count","@value":1}	1
2024-05-19T19:35:04Z	MetricsSearchResults:{"@version":1,"@namespace":"AWS/CloudWatchMetricsInsights","@metricName":"MetricsSearchResults","@label":"MetricsSearchResults","@statistic":"Sum","@unit":"Count","@value":1}	1

## Launch Template

Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left-hand side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

Let's name our Launch Template, and then under **Application and OS Images** include the app tierMY AMI you created.

Let's set the correct security group for our web tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.

Accept the rest of the default configuration settings and click **Create launch template**.

The screenshot shows the AWS EC2 Launch Template details page. On the left, there's a navigation sidebar with links like EC2 Dashboard, EC2 Global View, Events, Console-to-Code, Instances, Images, and Network & Security. The main content area has a breadcrumb trail: EC2 > Launch templates > web-tier-launch-template. The title is "web-tier-launch-template (lt-08f3e2976d939fc3)". Below the title are two tabs: "Details" (selected), "Versions", and "Template tags". The "Launch template details" section shows the Launch template ID (lt-08f3e2976d939fc3), Launch template name (web-tier-launch-template), Default version (1), and Owner (arn:aws:iam::378064363647:root). The "Launch template version details" section shows Version 1 (Default) with a description of "launch template for the web tier", Date created (2024-05-18T14:05:04.000Z), and Created by (arn:aws:iam::378064363647:root). The "Instance details" tab is selected, showing AMI ID (ami-0e84ccf4010c9db84), Instance type (t2.micro), Availability Zone (us-east-1a), and Key pair name (None). Security groups are listed as sg-0a546a9b1ac1d4859.

## Auto Scaling

Once we have got the Launch Template created, let's go ahead to create the Auto Scaling Group for our web instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **AutoScaling** and click **Create Auto Scaling group**.

Let's give our Auto Scaling group a name, and then select the Launch Template we just created and click next.

On the **Choose instance launch options** page, let's set our VPC, and the public subnets for the web tier and proceed to the next step.

For this next step, we're going to attach the Auto Scaling Group to the Load Balancer we just created by selecting the existing web-tier load balancer's target group from the dropdown. Then, click next.

For **Configure group size and scaling policies**, set desired, minimum and maximum capacity to 2. Click Next (3x) to review and then Create Auto Scaling Group.

**Group details**

Auto Scaling group name	Desired capacity	Desired capacity type	Amazon Resource Name (ARN)
WebTierASG	2	Units (number of instances)	arn:aws:autoscaling:us-east-1:378064363647:autoScalingGroup:23fb0691-5994-4ff9-9d1a-27ef0caa5e6:autoScalingGroupName/WebTierASG
Date created	Minimum capacity	Status	
Sat May 18 2024 19:37:43 GMT+0530 (India Standard Time)	2	-	
	Maximum capacity		
	2		

**Launch template**

Launch template	AMI ID	Instance type	Owner
<a href="#">lt-08f3e2976d939fc3</a> web-tier-launch-template	<a href="#">ami-0e84ccf4010c9db84</a>	-	arn:aws:siam::378064363647:root
Version	Security groups	Security group IDs	Create time
Default	-	<a href="#">sg-0a546a9b1ac1d4859</a>	Sat May 18 2024 19:35:04 GMT+0530 (India Standard Time)

Let's test both our external load balancer and autoscaling group to see if they are configured correctly. The autoscaling group is spinning up 2 new web tier instances. Let's head over to the EC2 instances dashboard and delete one manually and wait to see if a new instance is booted up to replace it. Voila, it worked as intended!

Let's test if our entire architecture is working by plugging in our external facing load balancer, DNS name into your browser.

