Deploying a Three-tier Architecture in AWS Using the Console

By Philip Essel,
Certified Cloud Solutions Architect:
philipessel2006@gmail.com

Reason for Doing Project

The goal of this project is to build a cost-effective, scalable, and highly available three-tier architecture in AWS. By implementing this design, I aim to gain hands-on experience with core AWS services and showcase my ability to architect a production-level infrastructure, which can be leveraged for both professional development and potential job opportunities.

This project will allow me to demonstrate proficiency in creating modern, cloud-native applications while following industry standards for security, scalability, and fault tolerance.

Additionally, I seek to gain deeper knowledge of managing different application layers, optimizing performance with services such as Auto Scaling and Load Balancing. This project will also help me explore best practices for cost optimization while delivering a reliable architecture for web applications.

Project Overview

In this project, I will build a three-tier architecture in AWS. This consist of the presentation layer, the application layer, and the database layer. The architecture will be designed to be scalable, highly available, and support fault tolerance.

- **Presentation Layer**: A web server will serve the front end of the application using Amazon Elastic Load Balancing (ELB) to distribute traffic to multiple web servers running in Amazon EC2 Auto Scaling groups.
- **Application Layer**: The logic of the application will reside in an EC2 instance managed by an Auto Scaling group to handle dynamic content processing.
- **Database Layer**: A managed relational database, such as Amazon Aurora, will be used to store and manage application data.

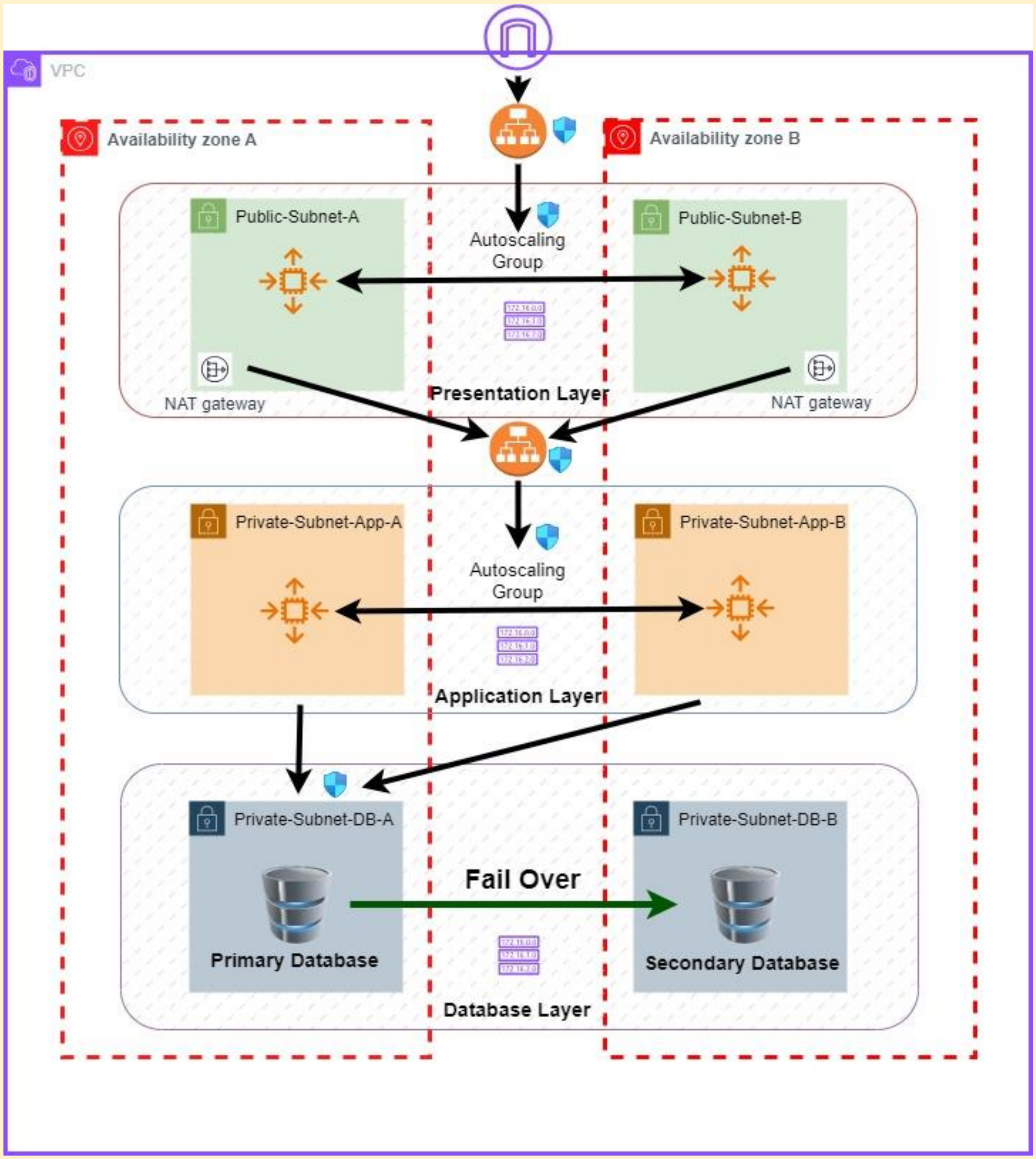
This project will demonstrate how the three-tier work together to create a reliable, scalable and fault-tolerant three-tier application. Security measures such as VPC, security groups, and IAM will be implemented to control access to resources.

Technologies Used

Technologies and services used for this project include:

1. **Amazon EC2**: To host the application servers in the application tier.
2. **Amazon RDS**: For the relational database in the database tier.
3. **Elastic Load Balancing (ELB)**: For distributing incoming traffic across multiple application servers.
4. **Amazon S3**: For storing the application codes.
5. **Amazon VPC**: To isolate and secure the network.
6. **Auto Scaling**: For automatically scaling the application servers based on demand.
7. **IAM Roles and Policies**: To manage access to AWS services securely.

Architectural Diagram



Project Steps

The following are the detailed steps for this project using the AWS Console.

Step 1: Create an S3 Bucket

Note: We are creating this bucket because this is where we will store our application code.

1. Log in to the AWS Management Console.
2. Navigate to the **S3** service.
3. Click on **Create bucket**.
4. **Bucket name**: Enter a globally unique name (Eg, **my-website-bucket**).
5. **Region**: Select the AWS region closest to your target audience.
6. **Block all public access**: Leave this in its default state. This is because we will not be accessing the S3 bucket directly. Rather, we will access it through our external load balancer.
7. Leave all other settings in the AWS S3 configuration dashboard as default.
8. Click on **Create bucket**.

Step 2: Download Application Code unto your local Machine

Note: The application code for this project contains two folders (App-tier folder and the Web-tier folder) and an Nginx.conf file. The web-tier folder contains the code for the presentation layer (the code for our website) whiles the app-tier folder contains the code for the application layer. The Nginx.conf file will be used to replace the default Nginx.conf file that comes with Nginx installation.

1. Go to the following GitHub Repository: https://github.com/philipessel/Deploying-Three-Tier-Architecture
2. Click on the **Code** tab on the right side (green rectangle).
3. Click on **Download Zip**.
4. Click on **Upload**.
5. The zip folder would be downloaded. Copy it and place it on your desktop.
6. Right-Click on it and select **Extract All**.
7. Select or create folder where files will be extracted to.
8. Click **Extract**.
9. The application code will be downloaded. It consist of web-tier folder, app-tier folder and nginx.conf file.

Step 3: Upload Application Code into S3 Bucket

1. Open your newly created bucket.
2. Click on the **Upload** button.
3. Click on **Add folder**.
4. Navigate to your desktop.
5. Select the **Application-Code** folder.
6. Click on **Upload**.

Step 4: Create IAM ROLE (EC2 Instance Role)

Note: We will be creating this IAM role to enable our EC2 instances to call AWS Services on our behalf. We will attach to this role, the necessary permissions to enable the role perform the required function. This IAM role will then be attached to all our EC2 instances that we will create in this project.

1. Navigate to the **IAM dashboard**.
2. Click **IAM**.
3. At **the Access management** on the left panel, locate **Roles**.
4. Click **Create role**.
5. At the **Trusted entity type** section, Choose **AWS service**.
6. At the **Use case** section, Choose **EC2 service**.
7. Click **Next**.
8. It's time to attach permissions to our role. We will attach two permissions, **AmazonS3ReadOnlyAccess** and **AmazonSSMManagedInstanceCore**. The **AmazonS3ReadOnlyAccess** permission will ensure that our EC2 instances can securely download objects and files from S3 whiles the permission **AmazonSSMManagedInstanceCore** will ensure that we can use System Manager Session Manager to securely connect to our instances without SSH key. Follow the next steps to enable you attach these permissions to your ec2 Role.
9. First search (*filter policy by property or policy name*) for the permission **AmazonS3ReadOnlyAccess** and attach.
10. Repeat step 9 for the permission **AmazonSSMManagedInstanceCore** and attach.
11. Click **Next**: permissions will be shown to have been attached.
12. Give the role a **name**: Eg, **ec2-instance-role**.
13. **Description**: here, you will describe your role (Eg, **ec2 instance role**).
14. Everything else should be kept as default.

15. Click **Create role**.

Step 5: Create a VPC

Note: This is our logically isolated network into which all our resources for this project (except s3) will be placed. This provides some level of security to our architecture. This VPC would have its own CIDR IP address.

1. Navigate to the **VPC Dashboard** and click **Create VPC**.
2. Choose **VPC only** option.
3. Enter **VPC Name**: Eg. my-three-tier-vpc
4. At the **IPv4 CIDR block**: select **IPv4 CIDR manual input**
5. Enter **IPv4 CIDR:** eg. 10.0.0.0/16 (adjust based on your needs)
6. Leave **Tenancy**: as Default
7. Click **Create VPC**.

Step 6: Create Subnets

Note: Our VPC will further be divided into smaller networks called subnets. Creating subnets help provide high availability and connectivity options for our resources. Each of these subnets should have their own CIDR range. The CIDR range of the subnet should be a subset of the VPC CIDR range. The subnets we would be creating are in two categories, those that are public (because we want them to interact with the internet) and those that are private (because we don't want them to interact with the internet). It should be noted that per our architecture diagram, our VPC span two availability zones. Also, there are three layers (presentation, application and database layers). Each layer would contain two subnets, one in each availability zone. So in total, we would be creating 6 subnets (2 public subnets for the presentation layer, 2 private subnets for the application layer and 2 private subnets for the database layer).

1. Go to **Subnets** in the VPC Dashboard.
2. Select **Create subnet**.
3. Select VPC in which you will create your subnet: Eg. my-three-tier-vpc
    - Create 2 Public Subnets for the Presentation Layer:
        - **Name**: Public-Subnet-A
        - **Availability Zone**: Choose AZ 1a
        - **IPv4 CIDR blocks**: 10.0.1.0/24
        - Click **Add new subnet**
        - **Name**: Public-Subnet-B

        - **Availability Zone**: Choose AZ 1b
        - **IPv4 CIDR blocks**: 10.0.2.0/24
        - Click **Add new subnet**
    - Create 2 Private Subnets for the Application Layer:
        - Name: Application-Layer-Subnet-A
        - **Availability Zone**: Choose AZ 1a
        - **IPv4 CIDR blocks**: 10.0.3.0/24
        - Click **Add new subnet**
        - Name: Application-Layer-Subnet-B
        - **Availability Zone**: Choose AZ 1b
        - **IPv4 CIDR blocks**: 10.0.4.0/24
        - Click **Add new subnet**
    - Create 2 Private Subnets for the Database Layer:
        - **Name**: DB-Subnet-A
        - **Availability Zone**: Choose AZ 1a
        - **IPv4 CIDR blocks**: 10.0.5.0/24
        - Click **Add new subnet**
        - **Name**: DB-Subnet-B
        - **Availability Zone**: Choose AZ 1b
        - **IPv4 CIDR blocks**: 10.0.6.0/24

4. Click **Create Subnets**.

Step 7: Create an Internet Gateway

Note: To enable internet connectivity, we would create this internet gateway. It would act as a modem that connects a PC to the internet. It should also be noted that after creating this internet gateway, it has to be attached to our VPC. Not doing this would render the internet gateway useless.

1. Navigate to **Internet Gateways**.
2. Click **Create Internet Gateway**.
    - Name: my-internet-gateway

- Click **Create internet gateway**

3. Attach the Internet Gateway to your **VPC** (my-three-tier-vpc).
    - Click **Actions**.
    - Select **Attach to VPC**.
    - Select your VPC.
    - Click **Attach internet gateway**.

Step 8: Create 2 NAT Gateways

Note: We created the internet gateway to enable our VPC and the resources within it to interact with the internet. It should however be noted that the only resources in the VPC that can interact with the internet are those in public subnet. All resources that are in private subnets will not be able to interact with the internet. This is good for security purpose. But often times, you might want your private resources to connect to the internet for the purpose of updates. That is where the Nat Gateway comes in handy. It will enable your private resources to connect to the internet for the purpose of updates. Whiles creating our Nat Gateway, it is very important for us to note that it has to be placed in the public subnet. In our case, we would create two Nat Gateways. We will place the first one in the public subnet of AZ 1a (**Public-Subnet-A**) and place the second one in the public subnet of AZ 1b (**Public-Subnet-A**).

1. Creating Nat Gateway in **Public-Subnet-A**
    - Navigate to **NAT Gateways** and click **Create NAT Gateway**.
        - Name: NAT-GW-AZ-A
        - **Subnet**: Choose your **Public-Subnet-A**.
        - **Connectivity type**: Public
        - **Elastic IP allocation Address**: click **Allocate elastic IP**.
        - Click **Create NAT Gateway**.
2. Creating Nat Gateway in **Public-Subnet-B**
        - Repeat the steps for the first NAT Gateway but choose Public**-Subnet-B**.
        - Name: NAT-GW-AZ-B
        - **Subnet**: Choose your **Public-Subnet-B**.
        - **Connectivity type**: Public
        - **Elastic IP allocation Address**: click **Allocate elastic IP**.
        - Click **Create NAT Gateway**.

Step 9: Create Route Tables for Our Subnets

Note: We have been able create Subnets, Internet Gateway and Nat Gateway. However, for these components to be able to communicate with each other, they need communication paths called routes. These paths (routes) can be created using route tables. A route table contains a set of rules called routes that are used to determine where the network traffic is directed. When you create a brand new VPC, AWS creates a route table called the main route table and applies it to the entire VPC. AWS assumes that when you create a new VPC with subnets, you want traffic to flow between those subnets. The default configuration of the main route table is to allow traffic between all subnets that are local to the VPC. However for best practice we would need to create our own route tables.

At the presentation layer, we will create a single route table whose target is the Internet Gateway. We will then associate this route table to the two public subnets (**Public-Subnet-A** and **Public-Subnet-B**). This particular route table would create a communication path (route) between the internet gateway and our 2 public subnets.

At the application layer, we will create two different route tables. The target for the first route table will be NAT-GW-AZ-A and it will be associated with **Private-Subnet-App-A** whiles the target for the second route table will be NAT-GW-AZ-B and it will also be associated with **Private-Subnet-App-B**. Note that at the presentation layer, a single route table will serve two public subnets whiles at the application layer two separate route tables will serve the two private subnets. The reason for this is that at the presentation layer, a single Internet Gateway serves both subnets (**Public-Subnet-A** and **Public-Subnet-B**). So there is no need to create separate route tables for each subnet. At the application layer however, we created two different NAT Gateways. One was placed in the **Public-Subnet-A** (this was called NAT-GW-AZ-A) and the other was placed in the **Public-Subnet-B** (this was called NAT-GW-AZ-B)**.** Therefore two different route tables must be created. This will be done to fulfil the high availability and fault-tolerance architecture we are aiming at.

1. Create a single route table for the 2 public subnets (**Public-Subnet-A** and **Public-Subnet-B** in the presentation layer) targeting Internet Gateway.
    - Navigate to the Route Table dashboard.
    - Click **Create route table**.
    - **Name**: give your route table a name (Eg. **public-subnet-route-table**).
    - Select your preferred VPC: Eg. my-three-tier-vpc
    - Click **Create route table**.
    - Navigate to the bottom of your route table and click **routes**.
    - Click **Edit routes** to add a route for internet traffic:
    - Click **Add route**.
        - Destination: 0.0.0.0/0
        - Target: **Internet Gateway** (my-internet-gateway).

- Click **Save changes**.
- Navigate to the bottom of the same route table and click **Subnet associations**.
- Click **Edit subnet associations**.
- Select the <u>two public subnets in the presentation layer</u>: **Public-Subnet-A** and **Public-Subnet-B**.
- Click **Save associations**.

2. Create route table for first private subnet (**Private-Subnet-App-A** in the Application Layer) targeting NAT-GW-AZ-A.
   - Navigate to the Route Table dashboard.
   - Click **Create route table**.
   - **Name**: Give the route table a name (Eg. private-subnet-app-route-table-A).
   - Select your preferred VPC: Eg. my-three-tier-vpc
   - Click **Create route table**.
   - Navigate to the bottom of your route table and click **routes**.
   - Click **Edit routes**:
   - Click **Add route**.
     - Destination: 0.0.0.0/0
     - Target: NAT **Gateway** (NAT-GW-AZ-A).
   - Click **Save changes**.
   - Navigate to the bottom of the same route table and click **Subnet associations**.
   - Click **Edit subnet associations**.
   - Select the appropriate <u>subnet in the application layer</u>: **Private-Subnet-App-A**.
   - Click **Save associations**.
3. Create route table for second private subnet (**Private-Subnet-App-B** in the Application Layer) targeting NAT-GW-AZ-B.
   - Navigate to the Route Table dashboard.
   - Click **Create route table**.
   - **Name**: Give it a name (Eg. private-subnet-app-route-table-B)
   - Select your preferred VPC: Eg. my-three-tier-vpc
   - Click **Create route table**.

- Navigate to the bottom of your route table and click **routes**.
- Click **Edit routes**:
- Click **Add route**.
  - Destination: 0.0.0.0/0
  - Target: NAT **Gateway** (NAT-GW-AZ-B).
- Click **Save changes**.
- Navigate to the bottom of the same route table and click **Subnet associations**.
- Click **Edit subnet associations**.
- Select the appropriate subnet in the application layer: **Private-Subnet-App-B**.
- Click **Save associations**.

Step 10: Create Security Groups

Note: We have been able to create communication paths (routes) between our subnets using route tables. These routes will serve as roads through which resources in our VPC will communicate with each other. However, not any type of traffic can travel on these roads (routes). These type of traffic that can pass through these roads (routes) should be controlled. Two main tools can be used to achieve this: Network Access Control Lists (Network ACLs) and Security Groups (SG). This project will use AWS default Network ACL but we will be creating our own security groups (SG's are mandatory in AWS but Network ACL are optional). SG's are stateful and by default, they block all inbound traffic and allows all outbound traffic. In this project we will only add inbound rules without tempering with outbound rules.

We will create 5 Security Groups that would help the layers to securely communicate with each other: These include external load balancer security group (this will allow inbound traffic from the internet and allow outbound traffic to the presentation layer security group), presentation layer security group (this will allow inbound traffic from the external load balancer security group and allow outbound traffic to the internal load balancer security group), internal load balancer security group (this will allow inbound traffic from the presentation layer security group and allow outbound traffic to the application layer security group), application layer security group (this will allow inbound traffic from the internal load balancer security group and allow outbound traffic to the database layer security group), and database layer security group (this will allow inbound traffic from the application layer security group and allow outbound traffic to the database layer).

Finally, note that when you select the type of inbound rule, AWS automatically selects the protocol and port range for you. For example, when you select HTTPS as the type, AWS will select TCP as the protocol and 443 as the port range.

1. Create the External Load Balancer Security Group (First security group):
   - **Name**: Give it a name (Eg. External-LB-SG).
   - **Description**: Eg. External load balancer security group.

- Select the **VPC**: Eg. my-three-tier-vpc
- At the **Inbound rules** section, Click **Add rule**.
    - **Type**: HTTP
    - **Protocol**: TCP
    - **Port range**: 80
    - **Source**: 0.0.0.0/0
    - **Description**: allow http traffic from internet
- Click **Add rule**.
    - **Type**: HTTPS
    - **Protocol**: TCP
    - **Port range**: 80
    - **Source**: 0.0.0.0/0
    - **Description**: allow https traffic from internet
- Ignore outbound rule.
- Click **Create security group**.

2. Create the Presentation Layer Security Group (Second security group):
    - **Name**: Give it a name (Eg. Presentation-Layer-SG).
    - **Description**: Eg. Presentation-Layer-Security-Group.
    - Select the **VPC**: Eg. my-three-tier-vpc
    - At the **Inbound rules** section, Click **Add rule**.
        - **Type**: HTTP
        - **Protocol**: TCP
        - **Port range**: 80
        - **Source**: Select the external load balancer security group.
        - **Description**: allow http traffic from external load balancer security group.
    - Click **Add rule**.
        - **Type**: HTTPS
        - **Port range**: 443

        - **Source**: Select the external load balancer security group.
        - **Description**: allow https traffic from external load balancer security group.
    - Ignore outbound rule.
    - Click **Create security group**.

3. Create the Internal Load Balancer Security Group (Third security group):
    - **Name**: Give it a name (Eg. Internal-LB-SG).
    - **Description**: Eg. Internal load balancer security group.
    - Select the **VPC**: Eg. my-three-tier-vpc
    - At the **Inbound rules** section, Click **Add rule**.
        - **Type**: HTTP
        - **Protocol**: TCP
        - **Port range**: 80
        - **Source**: Select the presentation layer security group.
        - **Description**: allow http traffic from presentation layer security group.
    - Click **Add rule**.
        - **Type**: HTTPS
        - **Protocol**: TCP
        - **Port range**: 443
        - **Source**: Select the presentation layer security group.
        - **Description**: allow https traffic from presentation layer security group.
    - Ignore outbound rule.
    - Click **Create security group**.
4. Create the Application Layer Security Group (fourth security group):
    - **Name**: Give it a name (Eg. Application-Layer-Security-Group).
    - **Description**: Eg. Database-Layer-Security-Group.
    - Select the **VPC**: Eg. my-three-tier-vpc
    - At the **Inbound rules** section, Click **Add rule**.
        - **Type**: Custom TCP Rule
        - **Protocol**: TCP

        - **Port range**: 8080 (or your application port)
        - **Source**: Select the internal load balancer security group.
        - **Description**: allow application traffic from internal load balancer security group.
    - Click **Add rule**.
        - **Type**: SSH
        - **Protocol**: TCP
        - **Port range**: 22
        - **Source**: My IP
        - **Description:** Allow SSH access for administration
    - Ignore outbound rule.
    - Click **Create security group**.

5. Create the Database Layer Security Group (fifth security group):
    - **Name**: Give it a name (Eg. Database-Layer-Security-Group).
    - **Description**: Eg. Database-Layer-Security-Group.
    - Select the **VPC**: Eg. my-three-tier-vpc
    - At the **Inbound rules** section, Click **Add rule**.
        - **Type**: MYSQL/Aurora
        - **Protocol**: TCP
        - **Port range**: 3306
        - **Source**: Select the security group of application load balancer.
        - **Description**: allow traffic from application layer security group.
    - Ignore outbound rule.
    - Click **Create security group**.

Step 11: Create your Subnet group for the Database Layer

Note: A subnet group in AWS is a collection of subnets that typically spread across multiple Availability Zones. You have to define your subnet group for your database instances, like those in Amazon RDS or Aurora. When creating a database instance, AWS uses the subnet group to determine which subnets and AZs to place your instances in. This is to ensure high availability, fault tolerance, and optimized network traffic. Subnet groups are critical for multi-AZ database deployments, providing failover capability and isolating database resources in private subnets for security.

1. Navigate to **RDS Dashboard**.
2. At the left side of the panel, click **Subnet groups**.
3. Click **Create** DB s**ubnet group**.
   - **Name**: Give the subnet group a name (Eg. my-three-tier-database-subnet-group).
   - **Description**: MyDatabase-subnet-group.
   - **VPC**: Choose your VPC. Eg. my-three-tier-vpc
4. Add your subnets.
   - **Availability zones**: select your availability zones (**AZ 1a** and **AZ 1b**) that contain your database subnets.
   - **Subnets**: select your database subnets (**DB-Subnet-A** and **DB-Subnet-B**).
5. Click, **Create**, to create the database subnet group.

Step 12: Launch your Aurora Database for the Database Layer

1. Navigate to **RDS Dashboard**.
2. Click **Databases** on the left panel.
3. Click **Create Database**.
4. Choose a database creation method: select **standard create**.
5. Choose a database engine option: select **Aurora** (MySQL compatible).
6. Choose template: select **Dev/Test**.
7. Go to **Settings**:
   - Name of Database: Give the database a name (Eg. my-three-tier-database).
   - Credential setting:
     - Master Username: **admin**.
     - Ignore: manage master credential in AWS secret manager and auto generate a password.
     - Master Password: give it a suitable password (eg. **my-password-123456**) and confirm.
8. At the cluster storage configuration, select **Aurora Standard**.
9. Go to **instance configuration**.

- DB instance class: select **memory optimized**.
- Select instance class based on your needs: e.g., db.t3.micro for testing.

10. Availability and durability**.**
    - At the **Multi-AZ Deployment** section: select **create an Aurora Replica** or Reader node in a different AZ.
11. Connectivity**.**
    - At the **Compute resource** section: select **Don't connect to an EC2 compute resource**.
    - At the **Network type** section: select **IPv4**.
    - At the **Virtual private cloud (VPC)** section: choose your vpc **my-three-tier-vpc**.
    - At the **Database subnet group** section: select the subnet group you created (**my-three-tier-database-subnet-group**).
    - At the **Public access** section, select **no public access**.
    - At the **VPC security group** section, select **choose existing VPC security group** and select the security group you created for your database (Database-Layer-Security-Group).
12. Database authentication
    - At the **Database authentication options**: select **Password authentication**.
13. Monitoring
    - Uncheck the **Turn on performance insights**. We will not use it for now.
14. Select **Create database**.

Step 13: Launch EC2 Instance for the Application Layer

1. Navigate to **EC2**.
2. Click **Instances** on the left panel.
3. Click **Launch Instances**.
4. Configure your instance:
    - Name: Give it a name (Eg, **my-application-layer-instance**).
    - **AMI**: Choose from the options. Eg, **Linux**.
    - **Instance Type**: Choose an instance type Eg, **t2.micro for testing**.
    - **Key pair name**: select **proceed without a key pair**.
    - **Network Settings**: at this section, select E**dit**.

        - **VPC**: Select your VPC (**my-three-tier-vpc**).
        - **Subnet**: Choose one of the **application layer subnets** you created earlier on (Eg. **Application-Layer-Subnet-A**).
    - **Security Group**: Choose **select existing security group** and select the security group you created for your application layer **(Application-Layer-Security-Group)**.
    - Click on **Advance networking configuration**.
        - At the IAM instance profile, select your IAM role you created at the beginning of the project (**my-three-tier-iam-role**).
    - Keep all other settings as default.
    - Select **Launch instance**.

Step 14: Create Application Server, Install Packages and Test Connection.

Note: When you launch an EC2 instance, it is essentially a basic virtual machine that comes with an operating system (depending on the AMI you choose), but it is not necessarily configured to perform any specific server role by default.

To make the EC2 instance function as a server (such as a web server, application server, database server, file server, etc.), you need to install and configure additional software or packages that provide the necessary functionality.

In our case, we intend to create an application server from the above EC2 instance. This server will handle business logic, process API requests, and interacts with our Aurora database in the **database layer**. We will install 2 main things (Node.js and MySQL). The Node.js will power the backend application logic, interact with our database and process requests from the **presentation layer** (in our case, Nginx web server) whiles the MySQL will serve as the relational database management system responsible for managing the data stored in the database layer.

1. First, connect to the EC2 Instances using **Session Manager**.

    Note: Connecting to the instance can be achieved by three main methods (SSH, EC2 Instance Connection and Session Manager). But in our case we will use session manager.

    - Select the instance you just created (**my-application-layer-instance**).
    - Click the **Connect** tap at the top left corner.
    - Select **Session Manager**. Then click **Connect** at the bottom.
    - A new tab with SSM will be open.
    - Log into the user with following command.: `Sudo –su ec2-user`

2. Second, Install MySQL.

- Command 1: Use the command below to download the MySQL 8.0 community release RPM file from the official MySQL website. The RPM file is necessary to configure the MySQL repository on your system.

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

- Command 2: Use the command below to install the downloaded RPM package, which adds the MySQL repository to your system's package manager (dnf). This repository allows you to easily install and manage MySQL packages. The `-y` option automatically confirms the installation.

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

- Command 3: Use the command below to import the GPG key for the MySQL repository to your system. This ensures that the packages installed from the MySQL repository are trusted and have not been tampered with.

```
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2023
```

- Command 4: Use the command below to install the MySQL client. The client is essential for connecting to and interacting with MySQL databases from your server's command line.

```
sudo dnf install mysql-community-client -y
```

- Command 5: Use the command below to check whether MySQL has been installed.

```
mysql --version
```

3. Third, test connection between the instance (application server) and the database server.

   Note: This will be done to ensure that the application server is communicating with the database server.

   - Command 1: Use the command below to remotely connect to your MySQL database instance hosted on AWS RDS. Remember to replace `<RDS-Endpoint>` with your RDS Endpoint (or your database server endpoint) and the `<username>` with your username (**admin**). After pressing enter, you would be required to provide your password (**my-password-123456**).

```
mysql -h <RDS-Endpoint> -u <username> -p
```

   - Command 2: Use the command below to exit from the database server.

```
exit;
```

4. Fourth, transfer the app-tier code from the S3 bucket to our instance.

   Note: Until this point, our app-tier code is stored in our S3 bucket. We will need to transfer it to our instance that will be hosting our application server before we can deploy it.

   - Command 1: Use the command below to change the directory to the home directory of the default ec2-user on the EC2 instance. This prepares the environment for further commands by ensuring you are working in the correct directory.

```
cd /home/ec2-user
```

- Command 2: Use the command below to download the application code from an S3 bucket to your EC2 instance. The `--recursive` flag ensures that all files and subdirectories are copied, not just individual files. This is useful when you need to deploy application code stored in an S3 bucket to the EC2 instance. Remember to replace `<YOUR-S3-BUCKET-NAME>` with your bucket name.

```
sudo aws s3 cp s3://<YOUR-S3-BUCKET-NAME>/application-code/app-
tier app-tier --recursive
```

- Command 4: Use the command below to change the directory to the app-tier directory, which contains the application code. This ensures that subsequent commands operate in this folder.

```
cd app-tier
```

- Command 5: The command below ensures that the ec2-user owns the app-tier directory and all of its contents. This gives the ec2-user full control over these files, which is important for running and modifying the application.

```
sudo chown -R ec2-user:ec2-user /home/ec2-user/app-tier
```

- Command 6: The command below to sets file permissions for the app-tier folder. The permission 755 allows the ec2-user to read, write, and execute the files, while other users can only read and execute them. This is useful for allowing the application to run while keeping the code secure.

```
sudo chmod -R 755 /home/ec2-user/app-tier
```

5. Fifth, Install Node.js for our application server.

   Note: We will install the node.js on this instance to power the backend application logic, interacts with our database and process requests from the **presentation layer** (in our case, Nginx web server)

- Command 1: This command below will use curl to download and execute the installation script for **NVM (Node Version Manager)**. NVM is commonly used to install and manage multiple versions of Node.js on a system.

```
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.39.7/install.sh | bash
```

- Command 2: This command below will reload the shell configuration (.bashrc file). This will make NVM available in the current shell session after it is installed.

```
source ~/.bashrc
```

- Command 3: This command below will install **Node.js version 16** using NVM. Node.js version 16 is commonly used in production environments. For this project, that is what we will use although you can choose a different version depending on your project requirements.

```
nvm install 16
```

- Command 4: This command below will ensure that the current Node.js version is set to 16 for the shell session. It will allow you to run Node.js applications with version 16.

  ```
  nvm use 16
  ```

- Command 5: This command will install **PM2** globally via npm. PM2 is a process manager for Node.js applications that helps to keep apps alive, manage logs, and enable auto-restart on crashes.

  ```
  npm install -g pm2
  ```

- Command 6: This command will install project-specific dependencies listed in the package.json file. These dependencies are needed to run or develop the project.

  ```
  npm install
  ```

- Command 7: This command below will automatically fixe vulnerabilities in the installed Node.js packages. This is a security best practice to ensure that known issues in dependencies are addressed.

  ```
  npm audit fix
  ```

6. Sixth, start the index.js file.

- Command 1: Use the command below to start the Node.js application (index.js) using PM2. PM2 is the process manager for Node.js

  ```
  pm2 start index.js
  ```

  Command 2: Use the command below to show real-time logs of the running applications managed by PM2. Make sure to run `Ctrl+C` to exit from this mode.

  ```
  pm2 logs
  ```

- Command 3: Use the command below to generate and configure a startup script to start PM2 and your application at system boot. This command, by itself, doesn't finalize the setup, which requires further actions in the next step.

  ```
  pm2 startup
  ```

- Command 4: This command below will help PM2 to integrate with **systemd** (Linux system manager) and ensures that PM2 restarts automatically when the system reboots. The command uses the correct Node.js path and references the correct user (ec2-user) for an EC2 instance environment.

  ```
  sudo env PATH=$PATH:/home/ec2-
  user/.nvm/versions/node/v16.20.2/bin /home/ec2-
  user/.nvm/versions/node/v16.20.2/lib/node_modules/pm2/bin/pm2
  startup systemd -u ec2-user --hp /home/ec2-user
  ```

- Command 5: The command below will save the current PM2 process list and ensure the processes will automatically restart after a server reboot.

```
pm2 save
```

- Command 6: Use the command below to send a request to the local server running on port 4000 (the port used by the Node.js app) to check if the server is responding.

```
curl http://localhost:4000
```

- Command 6: This next command is similar to the previous command but will be used to check a custom health endpoint. It's good for testing whether the health-check endpoint is functioning.

```
curl http://localhost:4000/health
```

Step 15: Create AMI, Launch Template, Target Group, Internal Application Load Balancer and Autoscaling Group

1. We will first create an **amazon machine image** (AMI) of the application layer EC2 instance we created earlier.
    - Navigate to **EC2 Dashboard** >
    - Select the instances on the left panel.
    - Select the EC2 instance you created earlier for the application layer (**my-app-tier-instance**).
    - Select **Actions** at the top right corner.
    - Select **Image and template**.
    - Select **Create image**.
        - **Name**: give your image a name (eg, **application-layer-ec2-image**).
        - **Description**: describe your image (eg, **application layer ec2 image**).
        - Keep everything else as default.
        - Click **Create image**.
2. It's time to create our **launch template** from the **application layer AMI** we have just created. At the instances section on the left panel, select **Launch Templates**.
    - Select **Create Launch Templates**.
        - **Name**: give your launch template a name. (Eg, **application-layer-launch template**).
        - Go down to **Application and OS Images**.
        - Choose **My AMIs**.
        - Select the AMI you created earlier (**application-layer-ec2-image**).

        - **Instance type**: select **instance type** (eg, t2.micro).
        - **Firewall (security group)**: select the security group you created for the application layer (Application-Layer-Security-Group).
        - At the Storage (volumes), leave default **EBS volumes**.
        - Go to **Advanced details**.
        - IAM Instance profile: select the IAM role you created earlier (**ec2-instance-role**).
        - Click **Create launch template**.

3. We will now create our **target group**. At the load balancing section on the left panel, select **Target Groups**.
    - Select **Create target groups**.
        - At **choose a target type,** select **instances.**
        - **Name:** give your target group a name (Eg. application-layer-target-group)**.**
        - Select **Protocol** of HTTP and **Port** 4000.
        - Select your VPC (**my-three-tier-vpc**).
        - Health check protocol: **HTTP**.
        - Health check path: **/health**.
        - Select **Advanced health check settings**.
        - Healthy threshold: reduce the value to 2
        - Click **Next**.
        - Click **Create target group**.
4. At this point, we will create our **internal load balancer**. At the load balancing section on the left panel, select **Load Balancers**.
    - Select **Create Load balancer**.
        - We would create an **application load balancer** so select it.
        - **Scheme**: select **internal.**
        - Select your VPC (**my-three-tier-vpc**).
        - **Mappings**: select your 2 availability zones (**AZ 1a and AZ 1b**), each with its subnet (**Application-Layer-Subnet-A** and **Application-Layer-Subnet-B**). This is where the internal load balancer would direct traffic.
        - **Security group**: select the security group you created for the internal load balancer (Internal-LB-SG).

- Listener and routing
  - **Protocol**: HTTP
  - **Port**: 80
  - **Target group**: select your target group (**application-layer-target-group**).
  - Click **Create load balancer**.
  - Click view **load balancer** to confirm it's created.

5. It is time to create our **autoscaling group** using the launch template we have just created. At the EC2 Dashboard section on the left panel, go to **Auto Scaling** (at the bottom) and select **Auto Scaling Groups**.
   - Select **Create auto scaling group**.
     - **Name**: give your auto scaling group a name. (Eg, application-layer-autoscaling-group).
     - **Launch template**: select the launch template you just created (**application-layer-launch template**).
     - **Version**: maintain the default version of the launch template.
     - At the bottom, Select **Next**.
     - At the **Network** section, choose your VPC (**my-three-tier-vpc**).
     - **Availability Zones and subnets**: choose your two availability zones (**AZ 1a** and **AZ 1b**) and their subnets in the application layer (**Application-Layer-Subnet-A** and **Application-Layer-Subnet-B**).
     - Select **Next**.
     - Select **Attached to an existing load balancer**.
     - Select **Choose from your load balancer target groups**.
     - Select **target group**: choose the target group you created (**application-layer-target-group**).
     - Keep all other settings as default.
     - Go down and click **Next**.
     - At the **Group size** section, choose your minimum, maximum and desired capacity
       - ❖ Desired capacity: 2
       - ❖ Minimum capacity: 2
       - ❖ Maximum capacity: 2
     - At the **scaling policy** section, select **None**.

- Click **Next**.
- Click **Next**.
- Click **Next**.
- Preview your settings.
- Click **Create Auto Scaling group**.

Step 16: Launch EC2 Instance for the Presentation Layer

1. Navigate to **EC2**.
2. Click **Instances** on the left panel.
3. Click **Launch Instances**.
4. Configure your instance:
   - Name: give a name. Eg, **my-presentation-layer-instance**.
   - **AMI**: Choose from the options. Eg, **Ubuntu**.
   - **Instance Type**: Choose an instance type Eg, **t2.micro for testing**.
   - **Key pair name**: select **proceed without a key pair**.
   - **Network Settings**: at this section, select E**dit**.
     - **VPC**: Select your VPC (**my-three-tier-vpc**).
     - **Subnet**: Choose one of the **presentation layer subnets** you created earlier on.
     - **Auto assign public IP**: Enable it.
   - **Security Group**: Choose **select existing security group** and select the security group you created for your application layer **(Presentation-Layer-Security-Group)**.
   - Click on **Advance networking configuration**.
     - At the IAM instance profile, select your IAM role you created at the beginning of the project (**my-three-tier-iam-role**).
   - Keep all other settings as default.
   - Select **Launch instance**.

Step 17: Connect to EC2 Instance, Create your Webserver (By Installing Nginx) and Install Packages (Node.JS).

Note: As I have explained earlier on, the above EC2 instance we have just launched is not configured to perform any specific server role by default. For us to make it function as a server, we need to install and configure additional software or packages that provide the necessary functionality. In our case, we want to turn it into a webserver so we would install Nginx. We will also install Node.js to power the backend application logic and processes dynamic contents from the web server (Nginx), interacting with database and handling business logic.

1. First, we will connect to the EC2 Instances using **Session Manager**.

   Note: to be able to install our packages on our instance, we would first need to connect to it. Connecting to it can be done through 3 methods (SSH, EC2 Instant Connect and through Session Manager) but we are going to use Session Manager because we do not want to use any key pair).

   - Select the instance you just created (**my-presentation-layer-instance**).
   - Click the **Connect** tap at the top left corner.
   - Select **Session Manager**. Then click **Connect** at the bottom.
   - A new tab with SSM will be open.
   - Log into the user with following commands: `Sudo –su ec2-user`

2. Second, we will deploy and configure our web-tier code on our EC2 instance.

   Note: Until now, our web-tier code (website code) is stored on S3. But we will need to transfer it to our instance that would host our webserver. Then, we can deploy it there.

   - Command 1: Use the command below to change the current directory to the `ec2-user` home directory. The `ec2-user` is where files and directories will be managed for the application code.

     ```
     cd /home/ec2-user
     ```

   - Command 2: Continue with the command below. It will copy the application code from the S3 bucket into the EC2 instance. Make sure to replace `<YOUR-S3-BUCKET-NAME>` with your bucket name. In our case, the bucket name is **my-website-bucket**. Also note that. The `--recursive` flag ensures that all files and subdirectories within the specified S3 path (`web-tier`) are copied. This command transfers your application code from S3 to the EC2 instance, making it available for deployment:

     ```
     sudo aws s3 cp s3://<YOUR-S3-BUCKET-NAME>/application-code/web-
     tier web-tier --recursive
     ```

   - Command 3: To enable you list files in the current directory (`/home/ec2-user`) use the command below. The `-lrt` in the command will ensure that the files are listed in long format, sorted by modification time in reverse order (newest last):

     ```
     ls -lrt
     ```

- Command 4: To enable you to navigate into the web-tier directory, use this command below. This directory is where our copied application code resides:

  ```
  cd web-tier/
  ```

- Command 5: To display the files within the web-tier directory (organized in order), use this command below. We will do this to confirm the presence of the copied files and subdirectories:

  ```
  ls -lrt
  ```

- Command 6: To enable you set the necessary permissions to manage and run the files within the `web-tier` directory use the two commands below. This is crucial for the application to function correctly and to allow for secure access. The first command will change ownership of the entire `web-tier` directory to the `ec2-user` user and group. The second command will set read, write, and execute permissions for the owner (`ec2-user`) and read/execute permissions for everyone else on all files and directories within `web-tier`.

  ```
  sudo chown -R ec2-user:ec2-user /home/ec2-user/web-tier
  ```

  ```
  sudo chmod -R 755 /home/ec2-user/web-tier
  ```

3. Third, we will install Node.js for our web server.

   Note: The node.js on this instance will enable the webserver to generate dynamic content.

- Command 1: This command below uses curl to download and execute the installation script for **NVM (Node Version Manager)**. NVM is commonly used to install and manage multiple versions of Node.js on a system.

  ```
  curl -o- https://raw.githubusercontent.com/nvm-
  sh/nvm/v0.39.7/install.sh | bash
  ```

- Command 2: This command below reloads the shell configuration (.bashrc file) to make NVM available in the current shell session after it is installed.

  ```
  source ~/.bashrc
  ```

- Command 3: This command below installs **Node.js version 16** using NVM. Node.js version 16 is commonly used in production environments, though you can choose a different version depending on your project requirements.

  ```
  nvm install 16
  ```

- Command 4: This command below ensures that the current Node.js version is set to 16 for the shell session. It will allow you to run Node.js applications with version 16.

  ```
  nvm use 16
  ```

- Command 5: This command below will navigate to the web-tier directory. This is where your web application code is located.

  ```
  cd /home/ec2-user/web-tier
  ```

- Command 6: This command will install project-specific dependencies listed in the package.json file. These dependencies are needed to run the web application.

  ```
  npm install
  ```

- Command 7: This command below will automatically fix vulnerabilities in the installed Node.js packages. It's a security best practice to run this, but note that some vulnerabilities may require manual fixes.

  ```
  npm audit fix
  ```

- Command 8: This command below will run the build script defined in the package.json file, which usually compiles the application for production use. This step is essential in many web development setups for compiling assets and preparing the application for deployment.

  ```
  npm run build
  ```

4. Fifth Install Nginx.

   Note: As we have said earlier on, Nginx is what converts the ec2 instance into a webserver.

- Command 1: Use the command below to install the Nginx web server on your Amazon Linux-based EC2 instance.

  ```
  sudo yum install nginx -y
  ```

- Command 2: Use the command below to changes the current directory to the nginx configuration folder.

  ```
  cd /etc/nginx
  ```

- Command 3: Use the command below to rename the existing nginx.conf file to nginx-backup.conf to keep a backup before replacing it with a new configuration file.

  ```
  sudo mv nginx.conf nginx-backup.conf
  ```

- Command 4: Use this command below to copy the new nginx.conf file from our S3 bucket to the current directory. Make sure to replace `<YOUR-S3-BUCKET-NAME>` with your bucket name:

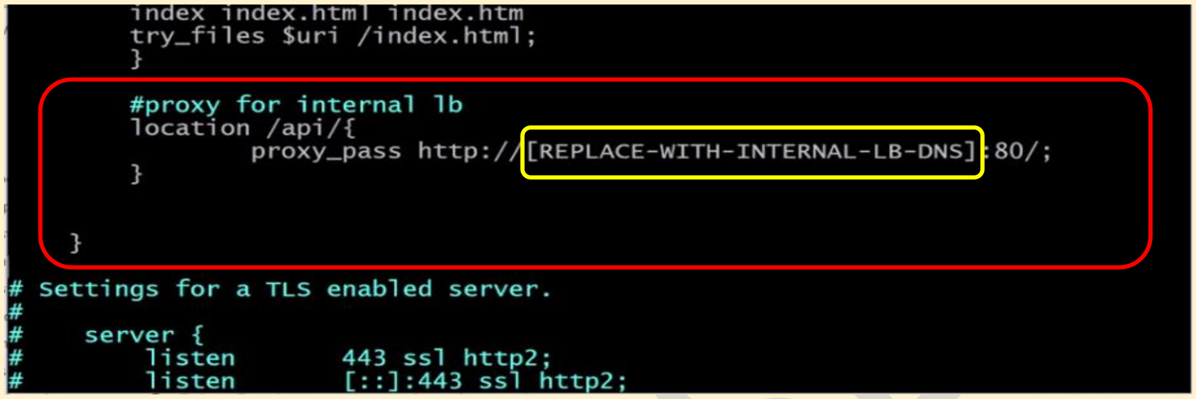
  ```
  sudo aws s3 cp s3://<YOUR-S3-BUCKET-NAME>/application-
  code/nginx.conf .
  ```

- Command 7: Use this command below to open the nginx.conf file in vim text editor for modification. This command should be followed by pressing the lower case letter "i" (this enables you to switch vim to insert mode so that you can edit the file).

  ```
  sudo vim nginx.conf
  ```

- Go to the section of the nginx configuration file where you would see **proxy for internal lb** (Red section below). At this point, we will modify the configuration by ensuring that the DNS name of our internal load balancer is included in the nginx configuration file.

This could have been done directly on the Nginx Configuration file before uploading it to the S3 bucket but since we have already loaded this file to the S3 bucket that is why we will modify it through the command line interface using vim editor.



```
        index index.html index.htm
        try_files $uri /index.html;
        }

        #proxy for internal lb
        location /api/{
                proxy_pass http://[REPLACE-WITH-INTERNAL-LB-DNS]:80/;
        }


    }
# Settings for a TLS enabled server.
#
#    server {
#        listen       443 ssl http2;
#        listen       [::]:443 ssl http2;
```

- Swap "[REPLACE-WITH-INTERNAL-LB-DNS] with the **DNS name** of your internal load balancer.
- Command 8: Use this command below to save and exit the nginx.conf file after making changes in vim.

  ```
  :wq
  ```

- Command 10: Because we have changed the nginx configuration file, we need to restart nginx to apply the new configuration changes.

  ```
  sudo service nginx restart
  ```

- Command 11: Use the command below to ensure that nginx starts automatically on system boot for Amazon Linux-based systems.

  ```
  sudo chkconfig nginx on
  ```

- Command 12: To ensure that nginx runs in the back-end continuously, use the command below:

  ```
  sudo chkconfig nginx on
  ```

Step 18: Create AMI, Launch Template, Target Group, External Application Load Balancer and Autoscaling Group

1. We will first create an **amazon machine image** (AMI) of the presentation layer EC2 instance we have just created.
   - Navigate to **EC2 Dashboard**.
   - Select **instances** on the left panel.
   - Select the EC2 instance you just created for the presentation layer (**my-presentation-layer-instance**).
   - Select **Actions** at the top right corner.

   - Select **Image and template**.
   - Select **Create image**.
     - **Name**: give your image a name (eg, **presentation-layer-ec2-image**).
     - **Description**: describe your image (eg, **presentation layer ec2 image**).
     - Keep everything else as default.
     - Click **Create image**.

2. It’s time to create our **launch template** from the **presentation layer AMI** we have just created. At the instances section on the left panel, select **Launch Templates**.
   - Select **Create Launch Templates**.
     - **Name**: give your launch template a name. (Eg, **presentation-layer-launch-template**).
     - Go down to **Application and OS Images**.
     - Choose **My AMIs**.
     - Select the AMI you created earlier (**presentation-layer-ec2-image**).
     - **Instance type**: select **instance type** (eg, t2.micro).
     - **Firewall (security group)**: select the security group you created for the presentation layer (**Presentation-Layer-Security-Group**).
     - At the Storage (volumes), leave default **EBS volumes**.
     - Go to **Advanced details**.
     - **IAM Instance profile**: select the IAM role you created earlier (**ec2-instance-role**).
     - Click **Create launch template**.
3. We will now create our **target group**. At the load balancing section on the left panel, select **Target Groups**.
   - Select **Create target groups**.
     - At **choose a target type,** select **instances.**
     - **Name:** give your target group a name (Eg. presentation-layer-target-group)**.**
     - Select **Protocol** of HTTP and **Port** 80.
     - Select your VPC (**my-three-tier-vpc**).
     - Health check protocol: **HTTP**.
     - Health check path: **/health**.
     - Select **Advanced health check settings**.

- Healthy threshold: reduce the value to 2
- Click **Next**.
- Click **Create target group**.

4. At this point, we will create our **external load balancer**. At the <u>load balancing section</u> on the left panel, select **Load Balancers**.
   - Select **Create Load balancer**.
     - We would create an **application load balancer** so select it.
     - **Scheme**: select **internet-facing.**
     - Select your VPC (**my-three-tier-vpc**).
     - **Mappings**: select your 2 availability zones (**AZ 1a and AZ 1b**), each with its subnet (**Public-Subnet-A** and **Public-Subnet-B**). This is where the external load balancer would direct traffic.
     - **Security group**: select the security group you created for the external load balancer (external-LB-SG).
   - Listener and routing
     - **Protocol**: HTTP
     - **Port**: 80
     - **Target group**: select your target group (**presentation-layer-target-group**).
     - Click **Create load balancer**.
     - Click view **load balancer** to confirm it's created.
5. It is time to create our autoscaling group using the launch template we have just created. At the EC2 Dashboard section on the left panel, go to **Auto Scaling** (at the bottom) and select **Auto Scaling Groups**.
   - Select **Create auto scaling group**.
     - **Name**: give your auto scaling group a name. (Eg, application-layer-autoscaling-group).
     - **Launch template**: select the launch template you just created (presentation-layer-launch template).
     - **Version**: maintain the default version of the launch template.
     - At the bottom, Select **Next**.
     - At the **Network** section, choose your VPC (**my-three-tier-vpc**).
     - **Availability Zones and subnets**: choose your two availability zones (**AZ 1a** and **AZ 1b**) and their subnets in the presentation layer (**Public-Subnet-A** and **Public-Subnet-B**).

- Select **Next**.
- Select **Attached to an existing load balancer**.
- Select **Choose from your load balancer target groups**.
- Select target group: choose the target group you created (**presentation-layer-target-group**).
- Keep all other settings as default.
- Go down and click **Next**.
- At the **Group size** section, choose your minimum, maximum and desired capacity
    - ❖ Desired capacity: 2
    - ❖ Minimum capacity: 2
    - ❖ Maximum capacity: 2
- At the **scaling policy** section, select **None**.
- Click **Next**.
- Click **Next**.
- Click **Next**.
- Preview your settings.
- Click **Create Auto Scaling group**.

Step 19: Testing and Validation

1. Test the entire architecture by accessing the website via the **External Load Balancer**. At the load balancing section on the left panel, select **Load Balancers**.
2. Locate the DNS of the external load balancer.
3. Copy it and paste it in your web browser.
4. Press Enter.
5. Your website will be displayed.
6. Play around on the website's interface and enter some data into it to ensure its working.

END