Three Tier Application in AWS

Introduction

A three-tier application architecture is a common design pattern used to develop and deploy applications, consisting of three layers:

-Presentation Tier

-Application Tier

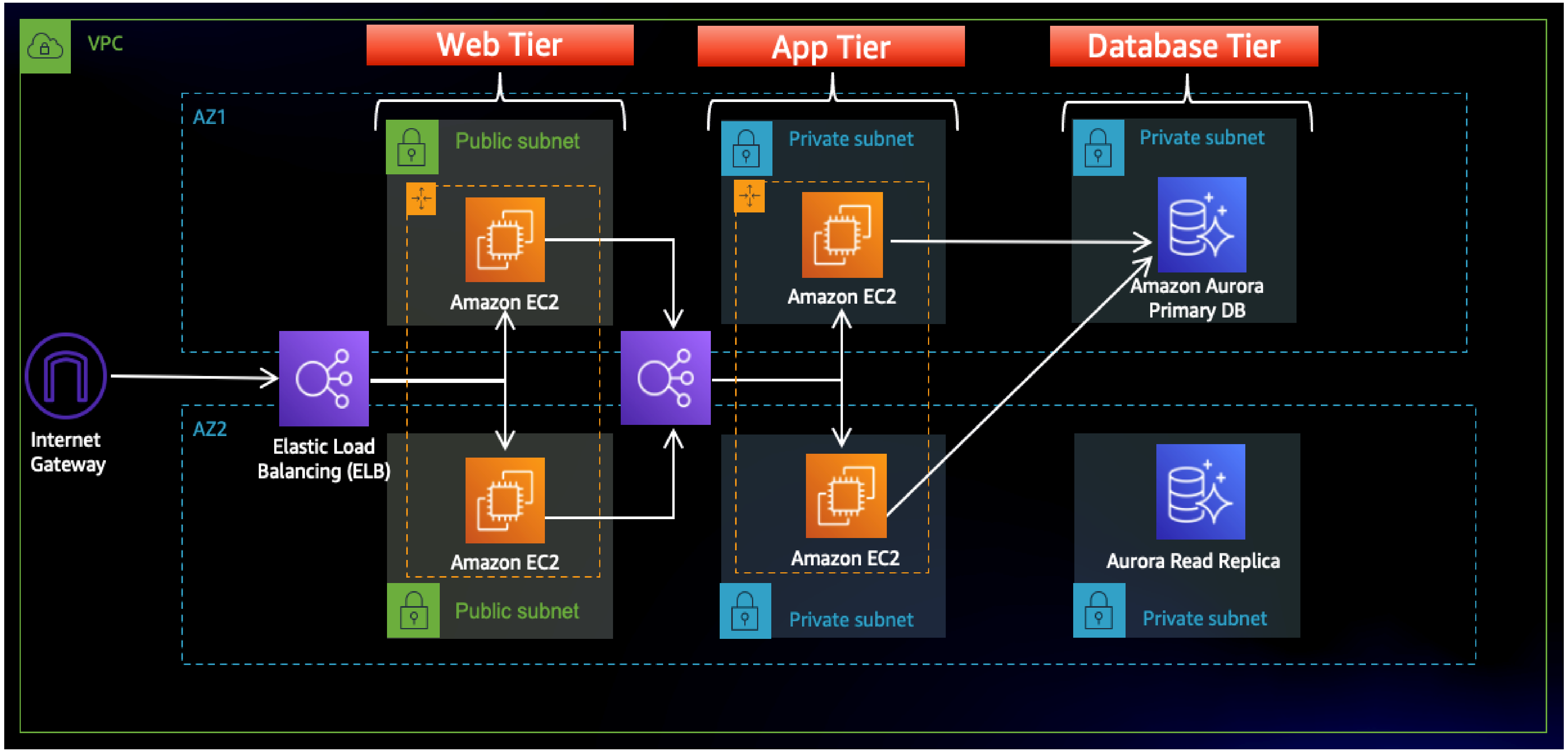
-Data Tier

-Each tier serves a specific purpose and can be scaled, managed, and deployed independently.

Three Tiers

Presentation Tier	Application Tier	Data Tier
• This tier is responsible for handling user interactions and presenting information to users. • Common components in this tier include web servers, load balancers, and content delivery networks (CDNs). • AWS services commonly used for the presentation tier include: -Amazon EC2 -Elastic Load Balancing (ELB) -Amazon CloudFront -Elastic Beanstalk	• This tier contains the business logic and application processing components of the system. • Common components in this tier include application servers, APIs, and middleware. • AWS services commonly used for the application tier include: -Amazon EC2 or AWS Lambda -Amazon API Gateway	• This tier stores and manages data used by the application. • Common components in this tier include databases, data warehouses, and data lakes. • AWS services commonly used for the data tier include: -Amazon RDS -Amazon DynamoDB -Amazon Redshift -

Architecture



Setup

- Download source code from below GitHub repository to your local machine:

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

Extract files/folders from your zip folder and use accordingly.

IAM Role Setup

IAM EC2 Instance Role Creation-

1. Navigate to the IAM dashboard in the AWS console and create an EC2 role.
2. Select EC2 as the trusted entity.
3. When adding permissions, include the following AWS managed policies. You can search for them and select them. These policies will allow our instances to download our code from S3 and use Systems Manager Session Manager to securely connect to our instances without SSH keys through the AWS console.

- **AmazonSSMManagedInstanceCore**
- **AmazonS3ReadOnlyAccess**

4. Give your role a name, and then click **Create Role.**

Networking Setup

1.<u>VPC and Subnet Creation-</u>

i) Make sure VPC only is selected, and fill out the VPC Settings with a Name tag and a CIDR range of your choice.

2.<u>Subnet Creation-</u>

i) Create 6 subnets across 2 AZ.s within created VPC. That means that three subnets will be in one availability zone, and three subnets will be in another zone.

ii) Specify unique CIDR range for each subnet.

Internet Connectivity

<u>1.Create internet gateway-</u>

i) Create your internet gateway by simply giving it a name and clicking Create internet gateway.

ii) Attach internet gateway to your VPC.

<u>2.Create NAT Gateway-</u>

i) Fill in the Name, choose one of the public subnets and then allocate an Elastic IP.

Routing Configuration

1.Create route table-

i) First create one route table for the web layer *public subnets* and name it accordingly.

ii) After creating the route table, scroll down and click on the Routes tab and Edit routes.

iii) Add a route that directs traffic from the VPC to the internet gateway.

iv) Select Subnet Associations and click Edit subnet associations.

v) Select the two web layer public subnets and click Save associations.

vi) Now create 2 more route tables, one for each app layer private subnet in each AZ. These route tables will route app layer traffic destined for outside the VPC to the NAT gateway in the respective availability zone, so add the appropriate routes for that.

vii) Once the route tables are created and routes added, add the appropriate subnet associations for each of the app layer private subnets.

Security Groups

1.Create Security Group-

i) The first security group is for the public, type a name and description, add an inbound rule to allow HTTP type traffic for your IP.

ii) Create second security group is for the public instances in the web tier. Type a name and description, add an inbound rule that allows HTTP type traffic from your previously created security group.

   Then, add an additional rule that will allow HTTP type traffic for your IP.

iii) The third security group will be for our internal load balancer. Create new security group and add an inbound rule that allows HTTP type traffic from your public instance security group.

iv) The fourth security group is for our private instances. Type name and description, add an inbound rule that will allow TCP type traffic on port 4000 from the internal load balancer security group. You should also add another route for port 4000 that allows your IP for testing.

v) The fifth security group for protects our private database instances. Add an inbound rule that will allow traffic from the private instance security group to the MYSQL/Aurora port (3306).

Database Deployment

<u>1. Create Subnet Groups-</u>

i) Click Create DB subnet group.

ii) Give your subnet group a name, description, and choose the VPC we created.

iii) Add subnets that we created for database.

<u>2.Create Database-</u>

i) Navigate to RDS dashboard and click Create database.

ii) Start with a Standard create for this MySQL-Compatible Amazon Aurora database. Leave the rest of the defaults in the Engine options as default.

iii) Under the Templates section choose Dev/Test. Under Settings set a username and password.

iv) Next, under Availability and durability change the option to create an Aurora Replica or reader node in a different availability zone. Under Connectivity, set the VPC, choose the subnet group we created earlier, and select no for public access.

v) Set the security group we created for the database, make sure password authentication is selected as our authentication choice, and create the database.

App Tier Instance Deployment

1.Create EC2 Instance-

i) Navigate to the EC2 dashboard, click Launch Instances and add tag.

ii) Select the first Amazon Linux 2 AMI

iii) Select t.2 micro instance type.

iv) Configure Instance Details and make sure to select to correct Network, subnet, and IAM role. Note that this is the app layer, so use one of the private subnets we created for this layer.

v) Select key-pair as proceed without a keypair

vi) Keep the defaults setting for storage.

vii) Select appropriate Security Group for private app tier.

viii) Click Launch.

Connect to Instance

When the instance state is running, connect to your instance by clicking the checkmark box and click the connect button.

Select the Session Manager tab, and click connect.

Firstly, you will be logged in as ssm-user which is the default user. Switch to ec2-user by executing the following command in the browser terminal:

```
sudo -su ec2-user
```

Configure Database

Installation-

Install following package for mysql server using wget command:

wget https://repo.mysql.com/mysql57-community-release-el7.rpm

Install on your local machine after using:

sudo yum install mysql –y

Initiate your DB connection with your Aurora RDS writer endpoint. In the following command, replace the RDS writer endpoint and the username, and then execute it in the browser terminal:

mysql -h CHANGE-TO-YOUR-RDS-ENDPOINT -u CHANGE-TO-USER-NAME –p

Enter password and connect to your database

Database Operations

i) Create a database called webappdb with the following command using the MySQL CLI:

```
CREATE DATABASE webappdb;
```

ii) You can verify that it was created correctly with the following command:

```
SHOW DATABASES;
```

iii) Create a data table by first navigating to the database we just created:

```
USE webappdb;
```

iv) Create the following transactions table by executing command:

```
CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL
AUTO_INCREMENT, amount DECIMAL(10,2), description
VARCHAR(100), PRIMARY KEY(id));
```

Database Operations

v) Verify the table was created:

```
SHOW TABLES;
```

vi) Insert data into table for use/testing later:

vii) INSERT INTO transactions (amount,description) VALUES ('400','groceries’);

viii) Verify that your data was added by executing the following command:

```
SELECT * FROM transactions;
```

ix) When finished, just type exit and hit enter to exit the MySQL client.

Configure App Instance

Open the application-code/app-tier/DbConfig.js file from the GitHub repo and edit for the hostname, user, password and database.

1.S3 Bucket Creation-

i) Navigate to the S3 service and create a new S3 bucket.

ii) Give it a unique name, and then leave all the defaults as in.

iii) Upload the **app-tier** folder to the S3 bucket.

iv) Go back to your SSM session. Start by installing NVM (node version manager) using following command:

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

v) Next, install a compatible version of Node.js and make sure it's being used.

```
nvm install 16
nvm use 16
```

vi) PM2 is a daemon process manager that will keep our node.js app running when we exit the instance.

```
npm install -g pm2
```

Configure App Instance

vi) Now we need to download our code from our s3 buckets onto our instance. In the command below, replace BUCKET_NAME with the name of the bucket:

```
cd ~/
aws s3 cp s3://BUCKET_NAME/app-tier/ app-tier –recursive
```

vii) Navigate to the app directory, install dependencies, and start the app with pm2.

```
cd ~/app-tier
npm install
pm2 start index.js
```

viii) To make sure the app is running correctly run the following: pm2 list

To look at the latest errors, use this command: pm2 logs

ix) Right now, pm2 is just making sure our app stays running when we leave the SSM session.

```
pm2 startup
```

x) Save the current list of node processes with the following command:

```
pm2 save
```

Test App Tier

i) Run following command to simple health check endpoint that tells us if the app is simply running:

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

ii) Next, test your database connection:

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

If these both commands shows appropriate output then proceed with further process.

App Tier AMI

1. 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.
2. 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.

Target Group

i) 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 with our load blancer so it may balance traffic across our private app tier instances.

ii) Select Instances as the target type and give it a name.

iii) Then, set the protocol to HTTP and the port to 4000. Remember that this is the port our Node.ja app is running on.

iv) Select the VPC and then change the health check path to be /health. This is the health check endpoint of our app. Click Next.

v) We are NOT going to register any targets for now, so just skip that step and create the target group.

Internal Load Balancer

1. On EC2 dashboard select Load Balancers under Load Balancing and click Create Load Balancer.
2. Application Load Balancer is for our HTTP traffic so click the create button for that option.
3. After giving the load balancer a name, be sure to select internal since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.
4. Select the correct network configuration for VPC and private subnets.
5. 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.

Launch Template

1.Before we configure Auto Scaling, we need to create a Launch template with the AMI. On EC2 dashboard navigate to Launch Template under Instances and click Create Launch Template.

2.Name the Launch Template, and then under Application and OS Images include the app tier AMI.

3.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.

4.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

1. Create the Auto Scaling Group for our app instances. On EC2 dashboard navigate to Auto Scaling Groups under Auto Scaling and click Create Auto Scaling group.
2. Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.
3. On the Choose instance launch options page set your VPC, and the private instance subnets for the app tier.
4. For this next step, 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.
5. For Configure group size and scaling policies, set desired, minimum and maximum capacity to 2. Click skip to review and then Create Auto Scaling Group.

Web Tier Instance Deployment

1. Open up the application-code/nginx.conf file from the repo we downloaded. Scroll down to line 58 and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer’s DNS entry.
2. Then, upload this file and the application-code/web-tier folder to the s3 bucket you created earlier.

Web Instance Deployment

App Tier Instance Deployment, with the exception of the subnet. We will be provisioning this instance in one of our **public subnets**. Make sure to select the correct network components, security group, and IAM role. **This time, auto-assign a public ip** on the **Configure Instance Details page**. Remember to tag the instance with a name so we can identify it more easily.

Connect to Instance

- Follow the same steps used to connect to the 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
```

Configure Web Instance

1. Start by installing NVM and node :
```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install 16
nvm use 16
```
2. 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
```
3. Navigate to the web-tier folder and create the build folder for the react app:
```
cd ~/web-tier
npm install
npm run build
```
4. 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.
```
sudo amazon-linux-extras install nginx1 -y
```

Configure Web Instance

5. Navigate to the Nginx configuration file with the following commands and list the files in the directory:

```
cd /etc/nginx
 ls
```

6. Then, restart Nginx with the following command:

```
sudo service nginx restart
```

7. To make sure Nginx has permission to access our files execute this command:

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

8. And then to make sure the service starts on boot, run this command:

```
sudo chkconfig nginx on
```

To test if your entire architecture is working, navigate to your external facing loadbalancer, and plug in the DNS name into your browser amd then hit enter.

It will display your web page by performing appropriate function.

Thank you!