\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

VPC creation and subnets -

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vpc - 10.0.0.0/16

6 subnets ->

\*\*\*\*\*\*\*\*\*\*\*\*

1.10.0.0.0/24 - web-1b

2.10.0.1.0/24 - app-1b

3.10.0.2.0/24 - db-1b

4.10.0.3.0/24 - web-1a

5.10.0.4.0/24 - app-1a

6.10.0.5.0/24 - db-1a

-------------------

Create IGW & attach to vpc

-------------------

create 2 NAT gateway and attach them ->

1. nat-az1

2. nat-az2

------------------------------------

Routing ->

1. Public RT -> igw & ( web-az1 + web-az2 )

2. nat-az1 -> nat-az1 (app-az1)

3. nat-az2 -> nat-az2 (app-az2)

-------------------------------------

sg -

1. web-sg -> http-80 from 0.0.0.0/0 ssh from anywhere

2. internal-lb-sg -> (internal not internet facing) -> http from web-sg

3. app-sg -> http port-4000 from internal-lb-sg and SSH from web-sh

4. db-sg -> 3306 from app-sg

------------------------------------

Create s3 bucket so we can store our code in s3 –

-----------------------------------------

Database Configuration -

Subnet group - (db-az1 & db-az2)

Databse deployment -

Engine - Aurora ( 3.03.1 )

Aurora standard

------------------------------------

create app tier instance and connect to it ( amazon linux 2 )

And initiate connectivity between app tier and database execute following commands -

Note: - Copy all these MySQL commands to the note file and then start executing on an instance.

*sudo yum install mysql -y* -> install mysql client

*mysql -h* change-to-writer-database-endpoint *-u CHANGE-TO-USER-NAME –p*-> connect to DB

create database and some tables :

*CREATE DATABASE webappdb;*

*SHOW DATABASES;*

*USE webappdb;*

*CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL*

*AUTO\_INCREMENT, amount DECIMAL(10,2), description*

*VARCHAR(100), PRIMARY KEY(id));*

*SHOW TABLES;*

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

*SELECT \* FROM transactions;*

-----------------------------------------------------------------------------------------------------------------------------------

App tier configuration ->

update our database credentials for the app tier. To do this, let’s open the **application-code/app-tier/DbConfig.js** file from code.

upload the **app-tier** folder to the S3 bucket

Let us install few components that are require to run our backend code ->

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

*source ~/.bashrc*

(we are installing Node version manager to manage out backend code )

*nvm install 16*

*nvm use 16*

(installing compatible version of Node.js )

*npm install -g pm2*

(PM2 is a daemon process manager that will keep our node.js app running when we exit the instance or if it is rebooted)

we need to download our code from our S3 buckets onto our instance.

Create iam role – s3fullacess attach it to app instance  
let’s download to code from s3

*cd ~/*

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

( bucket policies )

Now we have install dependencies, and start the app with pm2

*cd ~/app-tier*

*npm install*

*pm2 start index.js*

To make sure the app is running correctly run

*pm2 list*

PM2 will make sure our app stays running when we leave the session. However, if the server is interrupted for some reason, we still want the app to start and keep running.

*pm2 startup*

\* after executing above command – copy paste the command output and run it \*

*pm2 save*

Test App Tier

*sudo curl* [*http://localhost:4000/health*](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

let’s test our database connection –

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

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.

Lets configure internal load balancer so we can high availability

Create a Target group –

Target – instances

Protocol – http port 4000

/health

Create a empty TG and don’t include any app instance

Now let us create Load balancer

- ALB   
- scheme – internal

- mappings – ( app-az1 & app-az2 )

Select TG we have created and create LB

Now let us configure a Web tier instance

Before we create and configure the web instances (web tier), let’s modify the **application-code/nginx.conf**

to line 58 and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer’s DNS entry

now

let’s upload the **‘nginx.config’** file and the **application-code/web-tier** folder to the s3 bucket

-------------------------

Launch EC2 instance – “Web” and configure it….

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*

download our web tier code from our s3 bucket

*cd ~/*

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

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.

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

We will now have to configure NGINX

cd /etc/nginx

ls

Let’s update the ‘nginx.conf’ file with the one we uploaded to S3 bucket. We’ll remove the file and replace

*sudo rm nginx.conf*

*sudo aws s3 cp s3://BUCKET\_NAME/nginx.conf .*

Let’s restart Nginx

*sudo service nginx restart*

Let’s make sure Nginx has permission to access our files

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

to make ensure the service starts on boot

*sudo chkconfig nginx on*

***\* Visit public ip of web you will get the result \****