AWS Three Tier Web Architecture

[Github-Project]

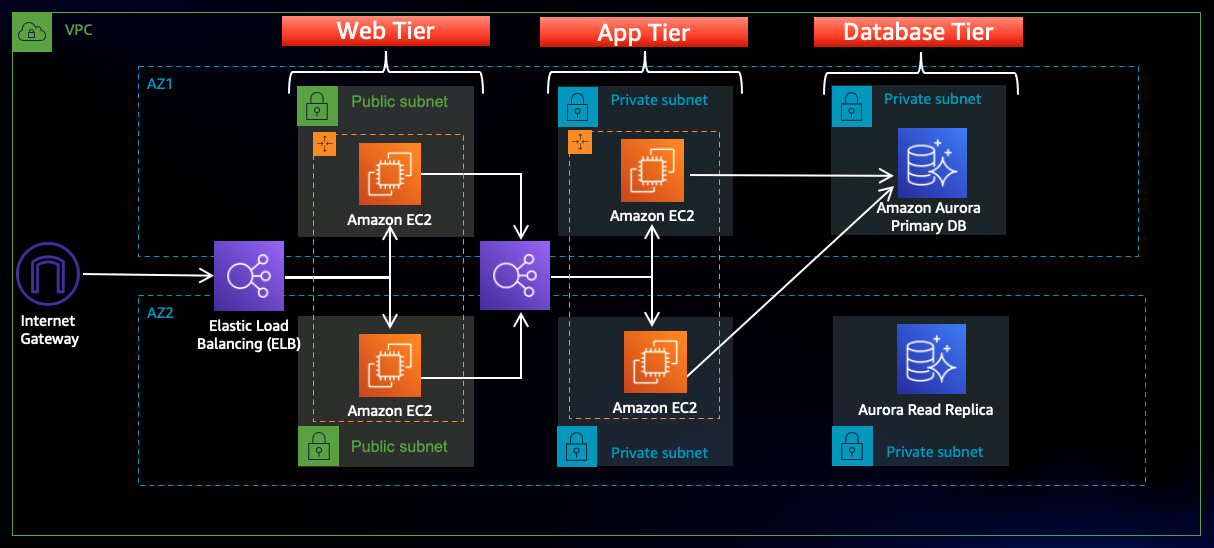
**Project Description**

This is a multi-tier web application architecture deployed on AWS (Amazon Web Services). It consists of three main tiers: web tier, application tier, and database tier, all designed with redundancy and scalability in mind. This project is robust, scalable web application architecture designed for high availability and performance.

**Architecture Overview**

In this architecture, a public-facing Application Load Balancer forwards client traffic to a web tier EC2 instances. The web tier is running Nginx webserver that is configured to serve a React.js website and redirects the API calls to the application tier’s internal facing load balancer. The internal facing load balancer then forwards that traffic to the application tier, which is written in Node.js. The application tier manipulates data in an Aurora MySQL multi-AZ database and returns it to our web tier. Load balancing, health checks and autoscaling groups are created at each layer to maintain the availability of this architecture.

<https://static.us-east-1.prod.workshops.aws/public/deeaf148-5f5f-4eac-ae36-a029faa8e4ba/static/introduction/3TierArch.png>



**Part 0: Setup**

For this workshop, I downloaded the code from Github and upload it to S3 so my instances can access it. I will also create an AWS Identity and Access Management EC2 role so it can be used with AWS Systems Manager Session Manager to connect to my instances securely and without needing to create SSH key pairs.

AWS managed policies used.

* AmazonSSMManagedInstanceCore
* AmazonS3ReadOnlyAccess

These policies will allow the instances to download code from S3 and use Systems Manager Session Manager to securely connect to the instances without SSH keys through the AWS console.

**Part 1: Networking and Security**

Building out the VPC networking components as well as security groups that will add a layer of protection around the ec2 instances.

Create an isolated network with the following components:

[VPC, Subnets, Route Tables, Internet Gateway, NAT gateway]

I will need **six** subnets across **two** availability zones. That means that **three** subnets will be in each availability zone.

A screenshot of a computer

Description automatically generated

**Internet Connectivity**

Internet Gateway: I give the public subnets in the VPC internet access, created and attach an Internet Gateway it to my VPC.

NatGateway: Instances in the app layer private subnet will need to go through a NAT Gateway that is deployed in each of my public subnets.

**Routing Configuration**

I added a route that directs traffic from the VPC to the internet gateway and edited the explicit subnet associations of the route table for subnet associations. I continued to create 2 more route tables, one for each app layer private subnet in each availability zone. These route tables will route app layer traffic destined for outside the VPC to the NAT gateway in the respective availability zone and added the appropriate routes for that.

**Security Groups**

Security groups will tighten the rules on which traffic will be allowed to my EC2 instances.

1-The public, internet facing load balancer. has an inbound rule to allow HTTP type traffic for my IP.

2-The public instances in the web tier has an inbound rule added that allows HTTP type traffic from the internet facing load balancer then I added an additional rule that will allow HTTP type traffic for my IP to allow access for testing.

3-Internal load balancer. This security group has an inbound rule that allows HTTP type traffic from my public instance security group. This will allow traffic from the web tier instances to hit my internal load balancer.

4-The fourth security group is for the private instances that has an inbound rule that will allow TCP type traffic on port 4000 from the internal load balancer security group. This is the port the app tier application is running on and allows the internal load balancer to forward traffic on this port to the private instances.

5-The fifth security group protects the private database instances. This security group has an inbound rule that will allow traffic from the private instance security group to the MYSQL/Aurora port (3306).

**Part 2: Database Deployment**

1-I created one Subnet Group.

2-Deployed MySQL-Compatible Amazon Aurora database. [Console]

3-Initiate the DB connection with the Aurora RDS writer endpoint.

4-Configured the database with some data and tables.

5-Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

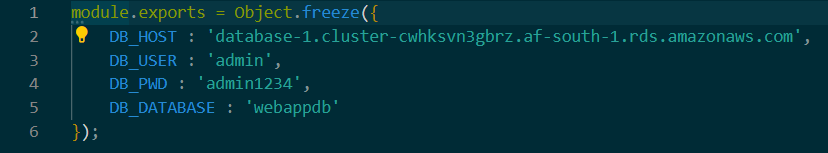
Description automatically generated

**Part 3: App Tier Instance Deployment**

Instance Deployment

1-I created an EC2 instance for my app layer. The app layer consists of a Node.js application that will run on port 4000.

2-I configure my app instance by updating my database credentials for the app tier. I modified the application-code in the DbConfig.js file and replaced the empty strings for the hostname, user, password and database with my credentials and the writer endpoint of my database as the hostname and named it webappdb for the database.



1-I Installed all the necessary components to run my backend application. I installed NVM (node version manager).

2-I downloaded the code from my s3 buckets onto my instance and installed all the dependencies.

3-Testing my App tier to see if the app is configured correctly and can retrieve data from the database

4-Health check = app is running

5-Database connection = local endpoint

A screen shot of a computer

Description automatically generated

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

1-I created AMI image from my application instance: Used with Auto Scaling.

2-I created a target group to use with my load balancer and set HTTP protocols on port 4000 where the Node.js app is running. The purpose of this target group is to use with my load balancer so it may balance traffic across my private app tier instances.

3-I created an application load balancer to route traffic from my web tier to my app tier, this application load balancer listens for HTTP traffic on port 80 and will forward traffic to my App-Tier Target Group.

4-I created the Launch Template from my application instance AMI for Auto Scaling [ Configuration setting same as application instance]

5-I created Auto Scaling group and attach to my internal load balancer and configured the group size and scaling policies.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

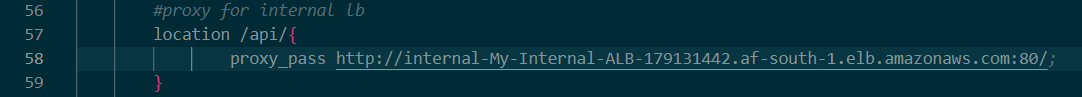
Description automatically generated

**Part 5: Web Tier Instance Deployment**

1-Update Config File and replaced the internal load balancer dns with my internal load balancer

A computer screen with a blue background

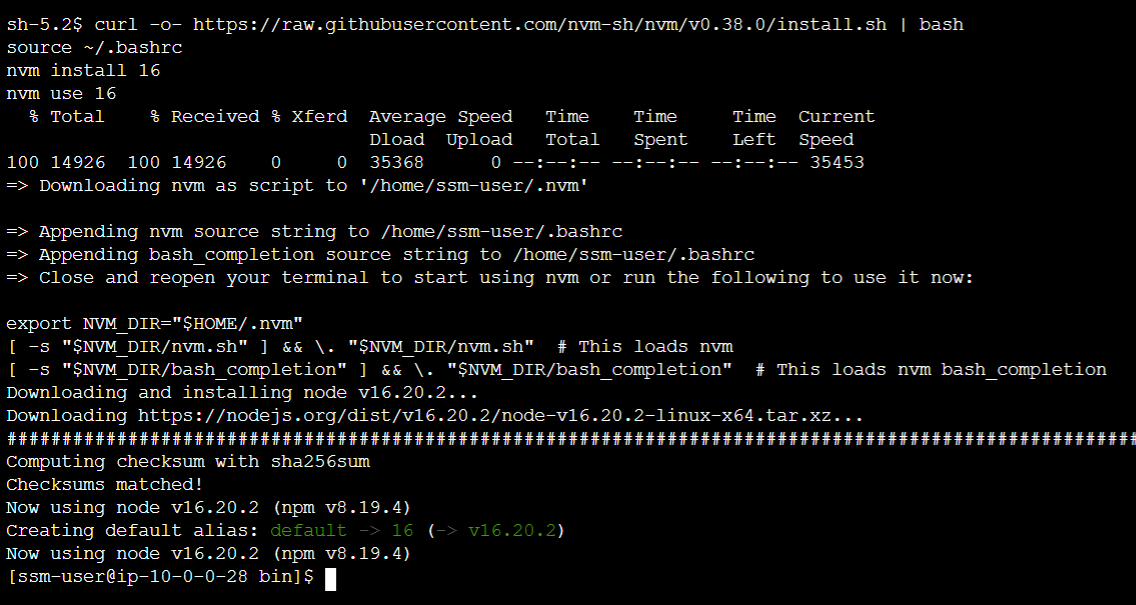
Description automatically generated



2-Web instance deployment [ Similar as/Ref: part 3 ]

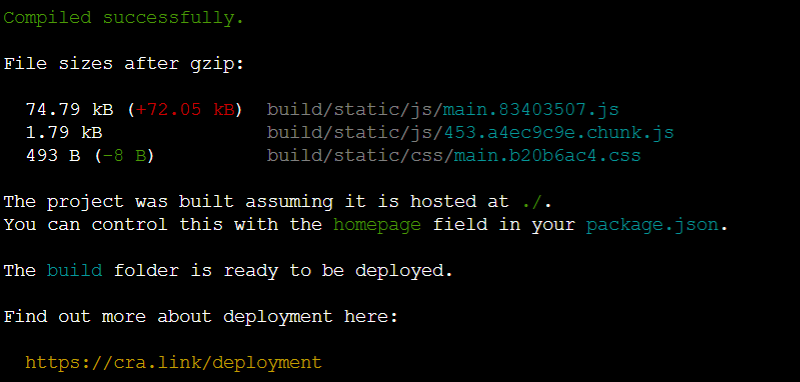
3-Configure Web Instance

4-Installed all of the necessary components to run my backend application. I installed NVM (node version manager).



5-Download the web tier code from my s3 bucket.

6-Create the build folder for the react app so it can serve the code.



The web server will be configured to serve the application on port 80, as well as help direct the API calls to the internal load balancer.

**Part 6: External Load Balancing and Auto Scaling**

1-Created AMI image from my web application instance: Used with Auto Scaling.

2-Create target group to use with my load balancer and set HTTP protocols on pot 4000 where the Node.js app is running. The purpose of this target group is to use with my load balancer so it may balance traffic across my public web tier instances.

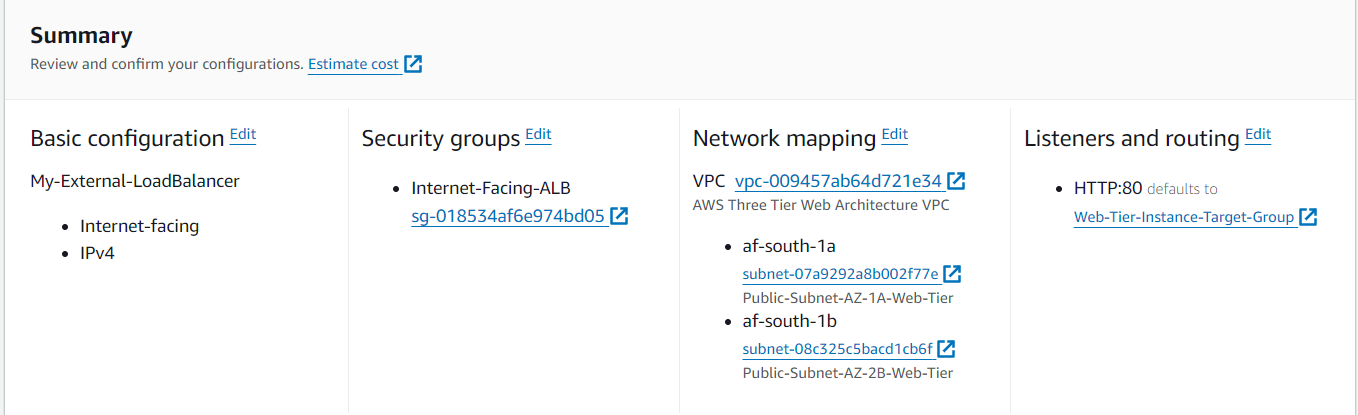
3-Create application load balancer to route traffic from my internet gateway to my public web tier, this application load balancer listens for HTTP traffic on port 80 and will forward traffic to my Web-Tier Target Group.

4-Create Launch Template from my web app instance AMI for Auto Scaling [ Configuration setting same as web app instance]

5-Create Auto Scaling group and attach to my external load balancer and configured the group size and scaling policies.

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Completes successfully!