Atypon Training Assignment
Amazon Web Services

Yazan Abdullah

Contents

List of Tables

Chapter 1

Introduction

In this report, I will outline my solution to the AWS Assignment.

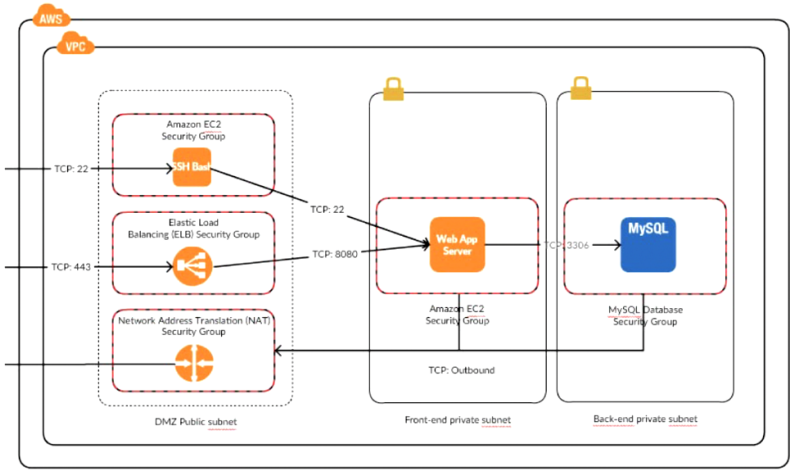

Figure 1.1: Assignment Architecture Diagram

I implemented all the components in the architecture diagram (Figure 1), in addition to the extra webserver.

I also created a domain name for my website and connected it to the database server.

Throughout my solution, I tried to follow the best practices, like using an IAM user, and validating my domain using DNS validation.

Chapter 2

VPC

The first thing I did was create a VPC within which I placed all the network components required in the assignment.

I named it `assignment-vpc`, and assigned the (`10.0.0.0/16`) CIDR block to it.

I chose to place it in **US East (N. Virginia)** region because it contains most of the services on AWS.

2.1 Subnets

After creating the vpc, I added three subnets (DMZ, Back-end and Front-end) to it. As shown in table 2.1:

	DMZ	Front-end	Back-end
VPC	assignment-vpc	assignment-vpc	assignment-vpc
Availability Zone	us-east-1f	us-east-1f	us-east-1f
CIDR block	10.0.0.0/24	10.0.1.0/24	10.0.2.0/24
Route Table	public-route-table	private-route-table	private-route-table

Table 2.1: Subnets Details

2.2 Route Tables

I created two route tables, one for public subnets (DMZ), and another for private subnets (Front-end and Back-end). You can find their details, in the following subsections:

Public Subnets

To make the DMZ subnet public (can access and is accessible from the Internet), I enabled the (Auto-assign public IPv4 address) option, then I created a new Internet gateway. Finally, I routed the Internet traffic through the gateway, using the `assignment-public-route-table`.

The public route table looks as follows:

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-026922dce927febaa

Table 2.2: Public Route Table

Private Subnets

Although, the Front-end and Back-end subnets are private, they still need to access the Internet so the instances in them can install and update software.

To provide this access, I created an Elastic IP Address, and attached it to a newly created NAT gateway. Finally, I created a route table that routed the traffic to the NAT gateway, and named it `assignment-private-route-table`.

The private route table looks as follows:

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	nat-050bcb20441db54e1

Table 2.3: Private Route Table

Chapter 3

EC2

3.1 Instances

I used EC2 instances to host the database (database server), serve webpages (web app servers), and to control access to the private network (Bastion host).

Table 3.1 summarizes the details of my instances:

Instance	Bastion Host	Web Server 1	Web Server 2	Database Server
AMI	Amazon Linux	Ubuntu	Ubuntu	Ubuntu
Type	t2.micro	t2.micro	t2.micro	t2.micro
Subnet	DMZ	Front-end	Front-end	Back-end
Volumes	8 GiB gp3	8 GiB gp3	8 GiB gp3	8 GiB gp3
Security Group	SSHable	WebServer	WebServer	DatabaseServer

Table 3.1: EC2 Intances

3.2 Load Balancing

I added a classic internet-facing load balancer to distribute the traffic between the two web servers.

I placed it in the DMZ subnet, and configured it to map network requests to the Front-end subnet.

I configured it to have two listeners, one on port 80 to handle HTTP requests, and another on port 443 to handle HTTPS requests. Note, the load balancer maps both requests into HTTP requests.

Periodically, the load balancer performs a health check for its registered instances to determine their statuses. It achieves this by pinging the index page (index.php).

I assigned to it the LoadBalancer security group, which only accepts HTTP(S) requests from any IP address.

3.3 Security Groups

I created several security groups to control the access to my servers. Table 3.2 displays the rules for the bastion host, table 3.3 displays them for the load balancer, table 3.4 is for the web servers, and table 3.5 displays the rules for the datbase server. (Note, I didn't place any restrictions on the outbound requests).

Protocol	Type	Port	Source
TCP	SSH	22	0.0.0.0/0

Table 3.2: Security Group Inbound Rules: SSHable.

Protocol	Type	Port	Source
TCP	HTTP	80	0.0.0.0/0
TCP	HTTPS	443	0.0.0.0/0

Table 3.3: Security Group Inbound Rules: LoadBalancer.

Protocol	Type	Port	Source
TCP	SSH	22	10.0.0.0/24
TCP	HTTP	80	10.0.0.0/24

Table 3.4: Security Group Inbound Rules: WebServer.

Protocol	Type	Port	Source
TCP	MYSQL/Aurora	3306	10.0.1.0/24
TCP	SSH	22	10.0.0.0/24

Table 3.5: Security Group Inbound Rules: Database.

3.4 running the instances

Bastion Host

All configurations of private servers must occur through the bastion host. To set this up, I first uploaded the relevant key pairs to the bastion host, then used them to connect to the private servers.

I used the following two commands to perform the main two operations, connection and data transfer:

```
# Connecting to a server
ssh -i "key_pair.pem" user@ip

# Sending a file to a server
scp -i "key_pair.pem" file-path user@ip:destination-dir
```

Database

To set up the database, I followed these steps:

1. SSH into the database server: through the bastion host.
2. Install MySQL Server: `sudo apt install mysql-server`.
3. Log into MySQL as root: `mysql -u root`.

4. Create a new database user: `GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' IDENTIFIED BY '123';`.
5. Create a new database: `CREATE DATABASE temp_database;`.
6. Log out of MySQL.
7. Log into MySQL as admin and create the necessary tables.

After this, I needed to modify the `/etc/mysql/mysql.conf.d/mysqld.cnf` configurations file to allow connection from any IP address. To do this, I set the value of `bind-address` to `0.0.0.0/0`, and then restarted the server, (`sudo mysql restart`).

Web Servers

After I SSHed into the webservers, I ran the following commands to set up the environment:

```
# First, installing the webserver
sudo apt install apache2

# Second, installing php
sudo apt install php libapache2-mod-php

# Third, installing packages for mysql suport
sudo apt-get install mysql-server php-mysql

# Restart the webserver to apply the changes
sudo service apache2 restart
```

Then, I placed my source code in the `/var/www/html` directory, and the server started to work.

Following, are small code snippets that interacts with the database. The firs, handles requests for adding a new name to the database; whereas, the second reads from the database and displays the names in a list.

```
# code snippet: writing to the database
```

```
function execute($sql) {
        $mysqli = new mysqli("10.0.2.75", "admin", "123",
            "temp_database", 3306);

        if ($mysqli -> connect_errno) {
                echo "Failed to connect to MySQL: " . $mysqli ->
                    connect_error;
                exit();
        }

        $result = $mysqli -> query($sql);

        $mysqli -> close();
}

if (isset($_POST['submit'])) {
        execute("INSERT INTO people VALUES('$_POST[name]');");
}
```

```
# code snippet: reading from the database

# connection to db code
$sql = "SELECT * FROM people;";

echo "<ul>";
if ($result = $mysqli -> query($sql)) {
        while ($row = $result -> fetch_row()) {
                echo "<li>$row[0]</li>";
        }
        $result -> free_result();
}
echo "</ul>";

$mysqli -> close();
```

Chapter 4

Route53 and Certificate Manager

To set up a domain name for my website, I first bought `rackfada.world` domain name from GoDaddy domain registrar. Then to connect this domain to my AWS load balancer, I Used Route53 service to create a hosted zone; then, I copied the name servers links to the GoDaddy registrar.

Then, I created an alias record that points to the load balancer in the hosted zone. And I created a CNAME record for the subdomain (www).

With this done, my domain name started working, but only for HTTP requests. So I used the AWS Certificate Manager service to request a public SSL/TLS certificate, and validated my domain using DNS validation.

I, then, added an HTTPS listener to the Load Balancer, and provided the certificate to complete the addition.

End of the Report
Thank You