AWS CUSTOM VPC SETUP AND CONFIGURATION

NOUREDDIN LUTFI

ATYPON JAVA & DEVOPS

Introduction

The project involves the development of a simple system within a Virtual Private Cloud (VPC) on AWS. The figure below shows the network layout of the system.

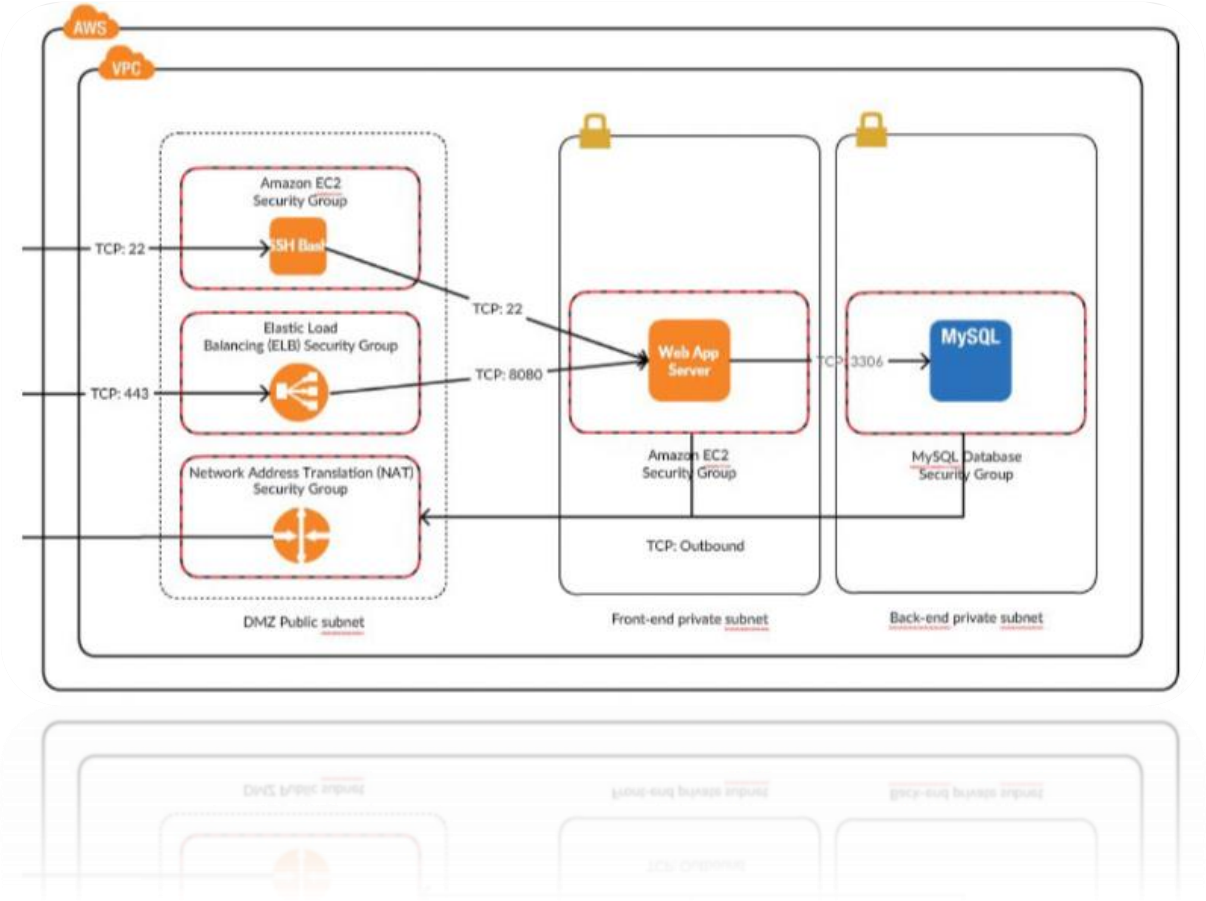

Figure 1. Network Layout of the AWS VPC System

The VPC is divided into three subnets:

1. **DMZ Public Subnet:** This subnet contains an **EC2 instance** configured as a host, enabling developers to securely connect via SSH to the internal network. It also includes a **load balancer** for managing incoming traffic distribution to the web servers and a **NAT gateway** that allows instances in the private subnets to access the internet securely.
2. **Front End Private Subnet:** This subnet hosts **two web servers** that handle the Front-End application. These servers receive traffic through the load balancer and interact with the backend database to fetch and store application data. They can also reach the internet through the NAT gateway for necessary updates or outbound connections.
3. **Back End Private Subnet:** This subnet contains a **MySQL database server** responsible for storing the application's data. The database server is isolated from direct inbound internet access, with database connections restricted to the web servers in the Front-End Private Subnet. However, it can access the internet for updates and other outbound communication through the NAT gateway.

The architecture is designed to provide enhanced security and scalability by separating public and private resources, implementing security measures like **security groups,** below is a figure illustrating the overall design and layout of the **implemented** VPC setup.

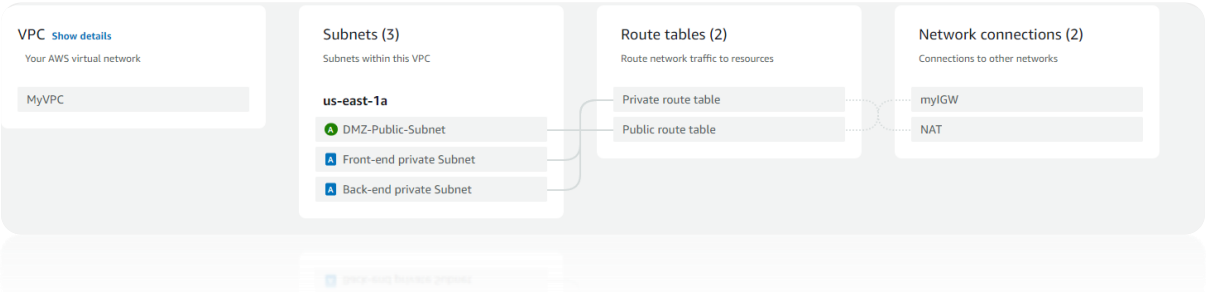

Figure 2. VPC Resource Map

Main Project Setup

The project sets up a Virtual Private Cloud (VPC) on AWS, named **MyVPC**, featuring **multiple subnets**, an **Internet Gateway**, **routing tables**, and **security groups** for managing access and network traffic:

❖ VPC and Subnets

1. **VPC Configuration:**
   - Created a VPC named **MyVPC** with the CIDR block 10.0.0.0/16.
   - Attached an Internet Gateway (**myIGW**) to allow the VPC to connect to the internet.

2. **Subnets:**
   - **DMZ Public Subnet:** CIDR block 10.0.0.0/24 with auto-assign IPv4 addresses enabled. This subnet hosts a **NAT gateway** and the **(bastion host)** for **SSH access**.
   - **Front End Private Subnet:** CIDR block 10.0.1.0/24, designed to **host web servers**.
   - **Back End Private Subnet:** CIDR block 10.0.2.0/24, reserved for the **database server**.

❖ Routing Tables

1. **Public Route Table routes (attached with public subnet):**
   - 10.0.0.0/16 (local)
   - 0.0.0.0/0 (Internet Gateway myIGW)

2. **Private Route Table routes (attached with private subnets):**
   - 10.0.0.0/16 (local).
   - 0.0.0.0/0 (NAT Gateway in DMZ Public Subnet).

❖ DMZ Public Subnet Configuration

1. **Bastion Host (SSH Bash):**
   - Created an EC2 instance named **SSH Bash** (AWS Linux) for secure SSH access with its own key pair "**SSH Bash.pem**".
   - Attached a security group allowing **SSH** access (port 22) from anywhere (0.0.0.0/0) and named it **SSH Bash security group**.

2. **Load Balancer:**
   - Created a **security group** named **Allow Port 443** to allow incoming HTTPS traffic (port 443).
   - Configured to **listen** on **port 443** and **route** traffic to web servers on **port 8080**.

3. **NAT Gateway:**
   - Allows instances in private subnets (Front and Backend) to **access the internet** while keeping them isolated from direct inbound traffic.

❖ Front End Subnet Configuration

1. **Created a Security Group for Web Servers (MyWebServerSG):**
   - Allows **SSH** access from the **SSH Bash security group (SSH Bash instance)**.
   - Allows **HTTP** traffic on port **8080** from the **Load Balancer's security group (Load Balancer)**.

2. **Web App Server:**
   - **Ubuntu instance**, attached to **MyWebServerSG** with its own key pair: **web.pem**.
   - Set up with **Apache**, customized to display index page (**Server 1**). The server is configured to use **www-data** for hosting **instead** of **root** to **enhance security**.
   - The listening port was changed from **80** to **8080**.

3. **Web App Server 2:**
   - Another Ubuntu instance with a **similar setup** as the first Web App Server with its own key pair: **web2.pem**.
   - Also configured with Apache and customized with its own index page (**Server 2**).

❖ Back End Subnet Configuration

1. **Database Endpoint**
   - Created an **endpoint** named **DataBase Endpoint** to facilitate access to the database.
   - This way is **much more secure** since we **separate** database administrators from web developers (as they access using the bastion host).

2. **Security Group for Database Endpoint (DBEndPoint):**
   - Allows **HTTPS connections** from **anywhere** (0.0.0.0/0).
   - Outbound rules permit SSH access to the **DataBaseRules security group** and **HTTPS** access to anywhere (0.0.0.0/0).

3. **Security Group for Database (DataBaseRules):**
   - Allows **SSH access from** the **DBEndPoint security group**.
   - Permits **MySQL** traffic (**port 3306**) from the **MyWebServerSG security group**.
   - Outbound rules **allow all TCP traffic** to **anywhere** (0.0.0.0/0).

4. **MySQL Installation:**
   - Installed **MySQL Server** on the instance.
   - Allowed **remote connections** (changed bind address in the configuration file to 0.0.0.0 instead of 127.0.0.1).
   - Created a database called "**test_db**" with table "**users**", the table contains two fields id and username.
   - Added a user (**noor**) with all privileges and host must be in (10.0.2.0/24) subnet so **only web servers** can access the database.

The following figures show the **final output** of web servers' responses using **load balancer**, **before adding extra steps**.

Note that the connection is made on port 443, the default port for **HTTPS**, but for simplicity, we used port 443 **without SSL certificates**.

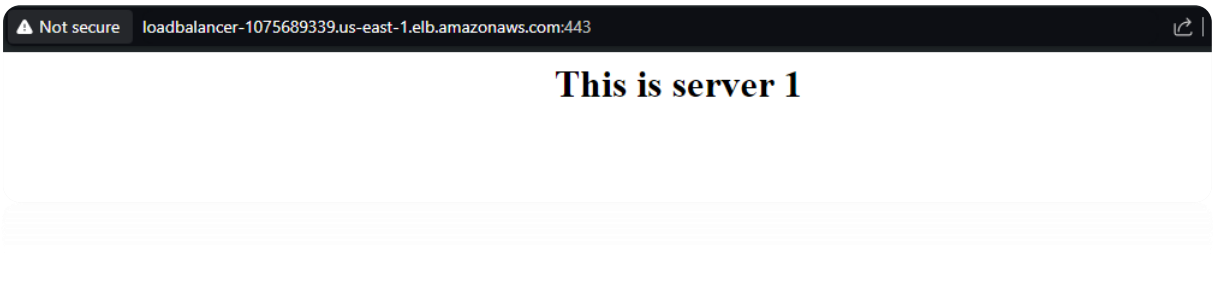

Figure 3. Server 1 index.html

Figure 4. Server 2 index.html

Extra Setup

Now we will connect the web servers with the database. Also, we will use Route53 service for domain name.

❖ Extra Setup Configuration

1. **PHP and Environment Configuration:**
   - On both web servers, **installed PHP and necessary extensions**.
   - Used **Composer** to install the **vlucas/phpdotenv** package for managing **environment variables**.
   - On both web servers, created a **.env** file in the project directory to **store environment variables securely**, and changed the permissions on the file so only web server can read the credentials.

2. **Adding PHP Script:**
   - Added a **PHP script** (index.php) to the web server.
   - The PHP script was configured to **read database credentials** from the **.env** file using the **phpdotenv package**.
   - This script enables interaction with the **MySQL database**, retrieving data and **displaying** it on the web page as **json** (PoC).

3. **Domain Configuration:**
   - Purchased a domain (**lutfe.site**) from **domain.com**.
   - Created a **hosted zone** in **AWS Route 53** for lutfe.site.
   - Configured the **name servers** from Route 53 **on domain.com**.
   - Waited for the **DNS** changes to propagate **globally** (approximately 5 hours).
   - Added an **A record** (**CNAME** works too) with an **alias** in Route 53 to point the domain (lutfe.site) **directly** to the **load balancer's DNS name**, ensuring that the **website** is **accessible** via the **domain**.

The following figures show the **final output** of web servers' responses using **domain name**.

Figure 5. Server 1

Figure 6. Server 2

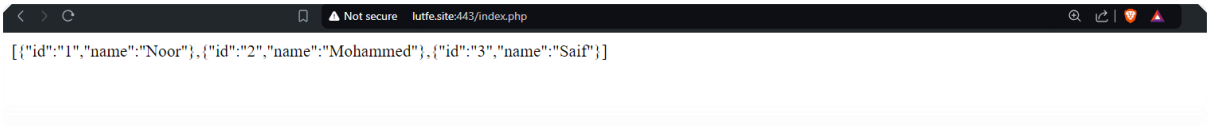

Figure 7. Servers Response with database connection

Conclusion

In this project, a Virtual Private Cloud (VPC) environment was successfully set up in AWS using various components, including public and private subnets, security groups, and a load balancer. The configuration ensured secure access to web and database servers while maintaining a scalable and robust infrastructure. By leveraging tools like Route 53 for DNS management and using environment variables for database credentials, the system was designed to be both secure and efficient. This setup provides a solid foundation for deploying web applications with best practices in mind, ensuring both performance and security.