***Group Project***

Group 1

Astha (8912446)

Anmol Singh (8975825)

Panth Patel (8975227)

INFO2350 – Advanced Cloud Infrastructure Administration

Nikola Cedic

**Task 1- Architecture Diagram**

A screenshot of a computer

AI-generated content may be incorrect.

**Description:**

* This required creating a scalable, secure AWS cloud architecture that could accommodate both a database backend and a web application. In essence, the diagram serves as a blueprint for illustrating how all the AWS resources are interconnected and communicate with each other. It must follow AWS best practices for security and scalability and clearly distinguish between the private and public-facing components.

**Preparation:**

* VPC CIDR: Assign all AWS resources to a particular Virtual Private Cloud (10.0.0.0/16).

Subjects:

* A Public Subnet for NAT Gateway and Web Server (172.16.0.0/24)
* A Database Server Private Subnet (172.16.1.0/24)

Components of Routing:

* To support internet traffic, the internet Gateway hangs off the VPC.
* The public subnet’s route table is accessible from the outside.
* For private resources (DB) to safely access the internet without being exposed, NAT gateways were deployed to the public subnet.

Groups for Security:

* One for the WebServer, which opens SSH, HTTP, and HTTPS.
* One is for the DBServer, which does not accept MySQL traffic from the internet; it only accepts it from the WebServer.

Diagram Tool Used:

* Lucidchart/draw.io was used to visually create the diagram, which included IP ranges, resource IDs, and clean labels.

**Observation:**

* The Public and Private layers are well separated in diagram.
* VPC, subnet, EC2 instances, gateways, and security controls are all properly tagged.

Demonstrates secure communication:

* The WebServer is accessible to users via SSH and an HTTP/S browser.
* WebServer and DBServer communicate via internal IP on port 3306 (MySQL).
* Since DBServer doesn’t have a public IP, it is totally safe from the outside.
* Strategic placement of the NAT Gateway enables the database to download packages without being online.

**Reflection:**

* We basically realized how crucial planning is before execution when we were creating the architecture diagram. When we had to connect actual components like VPC, subnets, NAT, and EC2, we realized how crucial proper structure and flow are. Initially, it was just box and arrow sketching. With the aid of this diagram, we were better able to understand how to secure resources placement, traffic flow, and database segregation. We were also aware of how important it was to mark everything accurately, including gateways, public/private zones, and CIDRs, since a single incorrect mark in the diagram will cause confusion when it is later deployed in real-time. This project gave us the confidence to move forward with a well-defined idea of what our infrastructure needs to be.

**Task-2**

A computer screen with a message

AI-generated content may be incorrect.

**VPC successfully created with CIDR block 10.0.0.0/16.**

A screenshot of a computer

AI-generated content may be incorrect.

**Two subnets created — one marked public and the other private.**

A screenshot of a computer

AI-generated content may be incorrect.

**Internet Gateway created and attached to the VPC.** A screenshot of a computer

AI-generated content may be incorrect.

**Public route table created and associated with public subnet.**

A screenshot of a computer

AI-generated content may be incorrect.

**NAT Gateway launched in public subnet for private subnet access.**

A screenshot of a computer

AI-generated content may be incorrect.

**DBServer security group configured to allow HTTP, MySQL, and SSH access only from specific IPs or security groups.**

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**WebServer security group configured with inbound rules for MySQL, SSH, HTTP, and HTTPS access**.A computer screen with a message

AI-generated content may be incorrect.

**WebServer EC2 instance is running in the public subnet with an assigned public IP and DNS for internet access.**

A computer screen with a message

AI-generated content may be incorrect.

**Both EC2 instances (WebServer and DBServer) are running, with the DBServer placed in the private subnet without a public IP.**

A screenshot of a computer

AI-generated content may be incorrect.

**Web server tested successfully in browser using its public IP.Web server tested successfully in browser using its public IP.**

A screenshot of a computer

AI-generated content may be incorrect.

**Connected to the web server via SSH and accessed the MariaDB database from the private database server using MobaXterm.**

**Description:**

* This exercise had the objective of building the infrastructure manually from the AWS Management Console. The goal was to make the diagram come to life-launch the web app in a public subnet and the database in a private subnet with all the configurations.

**Observation:**

* Every resource was appropriately produced and in line with the architecture.
* Because it is public, a browser could use the public IP to access the web server.
* The WebServer SSH login went smoothly.
* Used the MySQL client to connect to MariaDB running on DBServer via the internal IP of WebServer.
* Every security rule worked as it should, and the database is not connected to the internet.
* Screenshots of the VPC, subnets. EC2 status, IP addresses, and connectivity tests have been taken.

**Reflection:**

* Task 2 was very exciting and hands-on. We truly understood what goes on behind cloud networking after setting up every component from scratch in AWS Console. There were some challenging aspects, such as setting up the NAT Gateway, allocating Elastic Ips, and correctly connecting route tables. However, manually configuring everything got us acquainted with every detail of the AWS environment. Some problems, like misconfigured security groups and EC2 connectivity issues, were fixed by step-by-step troubleshooting. The most satisfying part was watching our web application run in the browser and then using WebServer and MobaXterm to connect to the private database. It gave participants practical experience setting up a secure application infrastructure.

**Task-3**

A screenshot of a computer

AI-generated content may be incorrect.

**Successfully connected to the MariaDB server from web server EC2 using MobaXterm after initial login errors were corrected.**

A screenshot of a computer

AI-generated content may be incorrect.

**Web server successfully displays a custom HTML message in the browser, confirming deployment and connectivity.**

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**CloudFormation stack successfully created all required AWS resources with status CREATE\_COMPLETE.**

A screenshot of a computer

AI-generated content may be incorrect.

**CloudFormation stack successfully created subnets, route tables, and associations for public and private networking.**

A screenshot of a computer

AI-generated content may be incorrect.

**CloudFormation stack completed creation of VPC, public subnet, EC2 web instance, and security group.**

A computer screen with a computer screen

AI-generated content may be incorrect.

**Custom VPC named “WebVPC1” is configured with public/private subnets, route tables, and both NAT and Internet gateways.**

A computer screen with a message

AI-generated content may be incorrect.

**WebAppInstance EC2 is running in the public subnet with public and private IPs, confirming successful deployment.**

A computer screen with a computer screen

AI-generated content may be incorrect.

**DBInstance EC2 is running in the private subnet with only a private IP, confirming secure backend setup.**

**Description:**

* This was accomplished by using AWS CloudFormation to automate the deployment . Reducing human intervention, preventing human error, and guaranteeing repeatability and predictability in infrastructure deployment were the objectives.

**Observation:**

* The CloudFormation stack was successfully uploaded and deployed.
* With all resources created as anticipated, the stack status is CREATE\_COMPLETE.
* WebAppInstance EC2 is launched with both public and private IP addresses in a public subnet.
* With just a private IP, DBInstance EC2 is launched in a private subnet.
* The web server displayed custom HTML content, verifying the deployment of the web application.
* Verified networking, IPs, and security group associations using AWS Console and Describe Instances.
* MySQL connectivity between WebServer and DB was confirmed from the terminal using MobaXterm.
* Stack creation, outputs, EC2 status, and test results are all included in screenshots.

**Preparation:**

* At first, automating everything with CloudFormation was a little confusing, especially when writing YAML syntax and specifying dependencies. However, there was a tremendous sense of accomplishment after the stack was successfully tested and deployed. The fact that the entire infrastructure-including the VPC, subnets, EC2s, security groups, and more-could be built with a single script was simply astounding. This exercise demonstrated the true benefits of infrastructure as Code, which include time savings, the ability to replicate the configuration at any time, and the avoidance of human error. Now that we’ve done it, we feel more relaxed managing more complex infrastructure in a production setting.

**Task 4: Security Considerations**

**Brief Report on Security Decisions and Trade-offs**

* The Following important security configuration and considerations were made when creating a scalable and secure AWS infrastructure:

**Problems and Solutions in Security**

1. Making the database server private

**Challenge:** To prevent exploitation or data breaches, the database must be made private.

**Solutions:**

* The database (DBServer EC2) was positioned in a private subnet without a public IP address as a solution. All incoming traffic from the public internet is blocked, and the VPC can only communicate with its internal resources.

1. Safe Interaction Between Database and Web Layers.

**Challenge:** Only allow secure communication between database and web servers.

**Solutions:**

* The DBServer security group was configured to only permit traffic from the WebServer security group for MySQL port (336), not from any public IP address.

1. Enabling Private Subnet Internet Access Without Exposure.

**Challenge:** Let the private DBServer download dependencies or updates without making it accessible to the public online.

**Solution:** To allow the private subnet to route outgoing internet traffic while blocking incoming traffic, a NAT gateway was set up in the public subnet.

**Security Decisions Summary:**

* Region\Security Mechanism
* Only WebServer EC2 allows public access.

**Database Protection**: Private subnet, no public IP address, access control, traffic managed by security groups internet for Private Subnet NAT Gateway for outgoing internet access only.

**Trade-offs Made:**

**Security vs. Complexity:**

* Although they increased architectural complexity, the use of NAT Gateway, public and private subnets, and multiple security groups greatly enhanced network separation and security.

**Cost vs. Protection:**

* Adding a NAT Gateway costs money on AWS, but it is necessary to protect the DBServer while permitting outgoing internet access.

**Control vs. Accessibility:**

* By preventing direct SSH access to the DBServer, extra steps are created (Such as using the WebServer as a jump host), which hinders debugging but also grants least privilege.

The Safety Structure aligns with AWS best practices and successfully lowers the attack surface without compromising backend database and web application functionality.

**Description:**

* Data and network security in cloud infrastructure were the focus of this activity. The primary goals were to use internal secure communication, restrict access to only the necessary areas, and avoid directly exposing sensitive systems.

**Observation:**

* The database was accessed internally and was isolated.
* WebServer was used as a jump host to access the database for configuration.
* For every security policy, the least privilege model was used.
* NAT Gateway effectively enabled private outgoing access.
* Network isolation was verified by the database server’s lack of a public IP address.
* Ping test results, NAT gateways status, and security group configurations are all shown in the screenshots.

**Reflection:**

* We gave cloud security a lot of thought after completing task 4. Although we frequently hear that AWS is secure, this task made us realize that it is our duty to properly implement it. We discovered that even small errors, such as exposing that database or setting up incorrect inbound rules, can have major consequences. We were able to obtain a layered security model by utilizing tight security groups, NAT Gateway, and private subnets. We were able to be aware of the trade-offs, such as the additional cost of using a NAT gateway or the complexity that would result from turning off SSH to the DBServer. All things considered, we now fully comprehended the significance of adhering to the “least privilege” principle and how AWS best practices contribute to the development of a secure system.