INFO2350 – Group Project Document

**Project Title: Designing a Secure and Scalable AWS Cloud Infrastructure**

**Course: INFO2350 – Advanced Cloud Computing**

**Group: G-04**

**Section: S-02**

**Team:**

**Maharshi Parekh (8992478)**

**Dhrumit Vekariya (8918788)**

**Cristian Osorio Vera (8903136)**

**Parth Trivedi (8827151)**

**Submission Date: August 05, 2025**

Contents

[Task 1: Architecture Diagram 3](#_Toc205227515)

[Description: 3](#_Toc205227516)

[Preparations 3](#_Toc205227517)

[Observation: 3](#_Toc205227518)

[DIAGRAM: 3](#_Toc205227519)

[TASK 2: AWS CONSOLE IMPLEMENTATION 4](#_Toc205227520)

[DESCRIPTION: 4](#_Toc205227521)

[PREPARATIONS: 4](#_Toc205227522)

[OBSERVATION: 4](#_Toc205227523)

[SCREENSHOTS: 4](#_Toc205227524)

[REFLECTION: 10](#_Toc205227525)

[TASK 3: TERRAFORM AUTOMATION (AWS CLOUDSHELL) 11](#_Toc205227526)

[DESCRIPTION: 11](#_Toc205227527)

[PREPARATIONS: 11](#_Toc205227528)

[SCREENSHOT: 11](#_Toc205227529)

[Task 4: Security Considerations Report 23](#_Toc205227530)

[Challenge 1: Database Protection 23](#_Toc205227531)

[Challenge 2: Controlled Web Access 23](#_Toc205227532)

[Challenge 3: Secure Auto Scaling 24](#_Toc205227533)

[Additional Best Practices 24](#_Toc205227534)

[Conclusion 24](#_Toc205227535)

# Task 1: Architecture Diagram

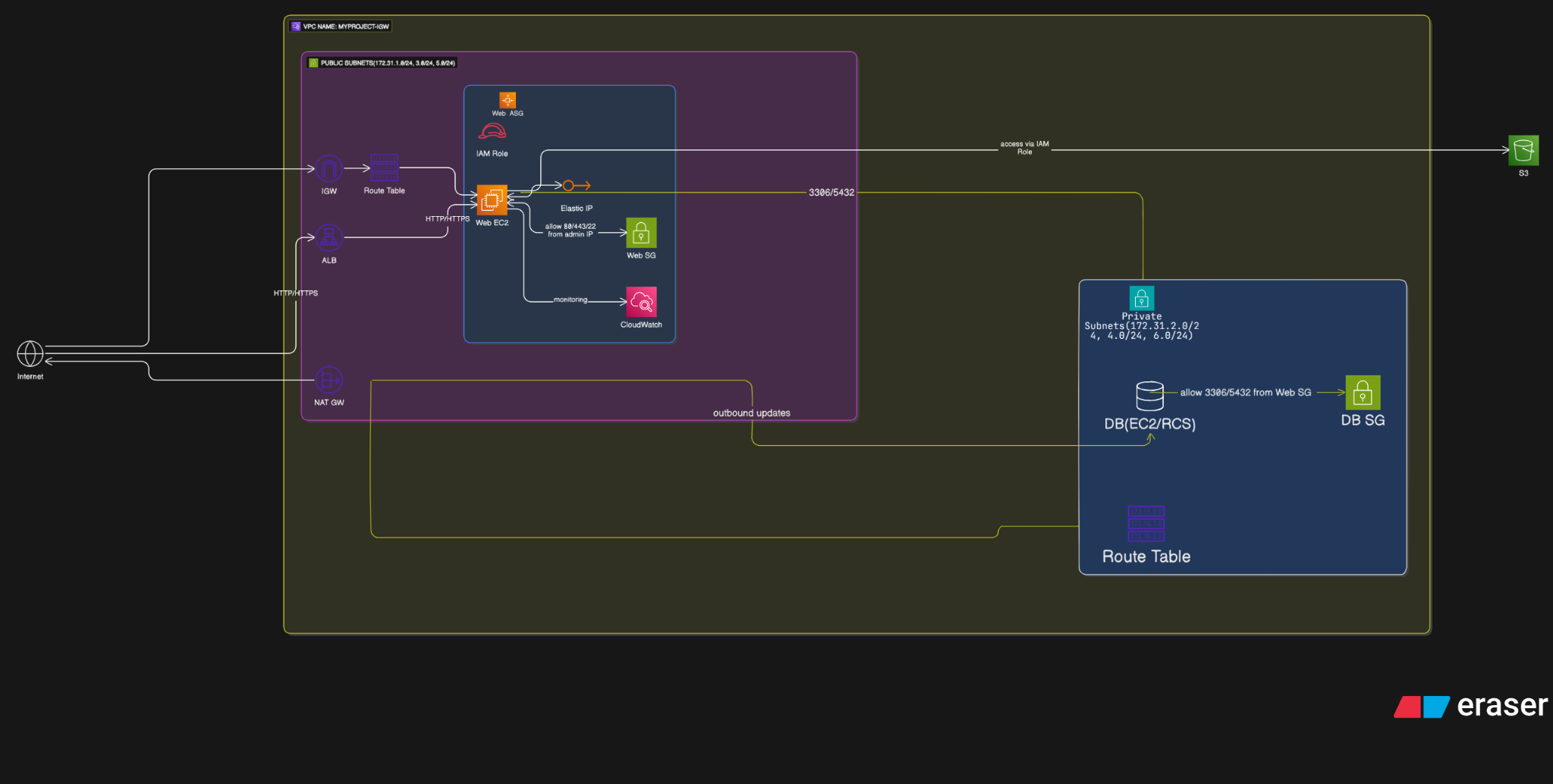
Description: The task consisted of creating secure and scalable AWS architecture, with the detailed architecture diagram showing placement and relationships between the major components such as VPC, subnets, EC2 instance, RDS database, route tables, security groups, ALB, IGW, and NAT Gateway.

Preparations: We specifically planned CIDR allocations and subnet distribution in respect to Availability Zones. Eraser.io and collaborative drawing tools were used to draw, collaborate, and plan. AWS best practices were researched and discussed to make sure there was appropriate segmentation between public and private subnets.

Observation: This diagram includes:

* VPC (CIDR: 172.31.0.0/16)
* 3 public + 3 private subnets
* EC2 instances in public subnets
* RDS in private subnets
* ALB in front of EC2
* Route tables with IGW and NAT setup

# DIAGRAM:



LINK: <https://app.eraser.io/workspace/BmGH6gxF2f8BMwqtfuqn?origin=share>

Reflection: This exercise helped us recognize how and why networking components can be connected securely and efficiently. This exercise provided a basis for following in class deployments and automation.

# TASK 2: AWS CONSOLE IMPLEMENTATION

DESCRIPTION:

For task two, we manually built the infrastructure above via the AWS Console. Essentially, we recreated our architecture by launching and configuring each of the following components: VPC, subnets, route tables, EC2, RDS.

PREPARATIONS:

We planned all IP ranges and subnets names. AWS Learner Lab access was configured, and individual responsibilities were assigned for deploying each component through the console.

## OBSERVATION:

* VPC: MyProject-IGW with CIDR 172.31.0.0/16
* 3 public + 3 private subnets across AZs
* Route tables manually associated
* EC2 (Amazon Linux 2023) deployed with Apache
* SSH access verified using PuTTY
* Browser loaded index.html page successfully

SCREENSHOTS:

**VPC Creation**

Created a custom VPC named ‘MyProject-IGW’ with the CIDR block 172.31.0.0/16.

A screenshot of a computer

AI-generated content may be incorrect.

**Route Tables Overview**

Configured multiple route tables, including one dedicated for public subnets.

**A screenshot of a computer

AI-generated content may be incorrect.Subnets Overview**

Created 6 subnets across different availability zones (3 public, 3 private).

**A screenshot of a computer

AI-generated content may be incorrect.**

**Edit Subnet Associations**

Selected specific subnets to associate with your public route table.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Subnet Associations Confirmed**

The public route table is explicitly associated with 3 public subnets.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Subnet Naming Example**

One of the public subnets is named ‘PrivateSubnet-A’.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Final Route Table View**

The route table 'MyProject-rtb-public' is associated with the correct subnets.

**A screenshot of a computer

AI-generated content may be incorrect.Resource Map Overview**

The VPC resource map shows subnet and route table connections, and IGW attachment.

A screenshot of a computer

AI-generated content may be incorrect.

Screenshot not available.

**Web Server Configuration and Test Screenshots**

1. SSH connection established using PuTTY:

A computer screen shot of a black screen

AI-generated content may be incorrect.

2. Browser successfully displays the deployed index.html file from EC2:

A screenshot of a computer

AI-generated content may be incorrect.

3. Apache installation, service start, and sample web page created from EC2 terminal:

A screenshot of a computer

AI-generated content may be incorrect.

## REFLECTION:

We became proficient in AWS networking and console navigation. This task also reinforced the importance of precise route table associations and firewall rules for proper access.

# TASK 3: TERRAFORM AUTOMATION (AWS CLOUDSHELL)

## DESCRIPTION:

In this task, we employed Terraform to automate our AWS infrastructure deployment. The objective was to create a repeatable and scalable solution using Infrastructure as Code (IaC) .

PREPARATIONS:

Terraform was installed in AWS CloudShell. We created configuration files: main.tf, provider.tf, variables.tf, rds.tf, and ec2.tf. AWS credentials were configured via environment variables.   
OBSERVATION:

* Rans terraform init , plan, and apply
* Successfully created: VPC, subnets, route table, EC2, ALB, RDS, and EIP
* Confirmed in AWS Console
* Terraform state reflected all resources as expected

## SCREENSHOT:

**Practical Demonstration:**

Installed terraform and We created the files in visual code studio for terraform to automate our webserver, load balancer, ec2 instance and security groups.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

We set the variables in our command prompt to map it to the learner’s lab by using the access key, ID, and token.

A screenshot of a computer screen

AI-generated content may be incorrect.

Changed the directory on cmd to have the correct path to validate our code and see if it is working.

So, we ran a terraform init command to initialize the system.

A screenshot of a computer

AI-generated content may be incorrect.

After the initialization is done, we ran terraform plan command to have the system ready and see if the code is correct and readable and is ready to be deployed in our AWS.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Once the plan is done and no errors are received, we did terraform apply to deploy the ec2, load balancer in the AWS learner’s lab.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

The command worked successfully and, in the dashboard below you can see that instance, load balancer, key pair, security groups etc. are created successfully.

A screenshot of a computer

AI-generated content may be incorrect.

EC2 instance created

A screenshot of a computer

AI-generated content may be incorrect.

Security groups created

A screenshot of a computer

AI-generated content may be incorrect.

Volume created

A computer screen with a message

AI-generated content may be incorrect.

Load Balancer created

A screenshot of a computer

AI-generated content may be incorrect.

Key pair created

A screenshot of a computer

AI-generated content may be incorrect.

RDS database created

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

VPC created

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Internet gateway created

A screenshot of a computer

AI-generated content may be incorrect.

Subnet Created

A screenshot of a computer

AI-generated content may be incorrect.

Route Tables Created

A computer screen shot of a computer

AI-generated content may be incorrect.

An Elastic IP is created to assign a public IP address

A screenshot of a computer

AI-generated content may be incorrect.

Terraform state list command to see what we have created on AWS

A computer screen shot of a black screen

AI-generated content may be incorrect.

# Task 4: Security Considerations Report

## Challenge 1: Database Protection

Problem: RDS databases are vulnerable if exposed to the internet. Attacks like brute force or SQL injections are more likely.

Mitigation:

- RDS deployed in private subnets (172.31.3.0/24, 172.31.4.0/24), with no public IP.

- RDS security group (db-sg) allows inbound MySQL traffic only from EC2 via web-sg.

- RDS configured with publicly accessible = false.

- Multi-AZ deployment ensures high availability.

Verification: Unable to connect to RDS externally. Connection via internal EC2 verified.

Trade-offs:

- Added complexity and cost due to use of NAT Gateway

- Slight performance latency during patching.

## Challenge 2: Controlled Web Access

Problem: The application should be accessible globally via HTTP/HTTPS but must remain protected from unauthorized access or DDoS.

Mitigation:

- ALB is the public-facing endpoint.

- web-sg only allows inbound HTTP (80) and HTTPS (443).

- SSH access is restricted to a specific IP range.

- HTTPS enabled using AWS Certificate Manager (optional in production).

- Auto Scaling Group configured to maintain 2–4 healthy instances.

- EC2 hardened with minimal software and updates using user data script.

Verification:

- Verified accessibility via public ALB DNS over HTTP.

- Ports other than 80/443 blocked.

- Load test ensured ASG launches new instances.

Trade-offs:

- ALB and ASG increase cost.

- IP restrictions improve security but reduce access flexibility

## Challenge 3: Secure Auto Scaling

Problem: As traffic increases, new instances must be securely and consistently configured.

Mitigation:

- Auto Scaling Group uses a Launch Template with pre-configured AMI, SG, and user data.

- New instances are automatically registered with the ALB target group.

- Health checks ensure only healthy instances serve traffic.

- RDS remains private and unaffected by scaling.

Verification:

- Load test triggered scale-out.

- Verified new instances had correct SG and Apache setup.

- RDS remained unreachable externally.

Trade-offs:

- Slight latency during health checks and instance launch

- Manual scaling would be faster but error-prone and insecure

## Additional Best Practices

- IAM Roles: EC2 instances use IAM roles for secure service access

- Monitoring: Enabled CloudTrail and CloudWatch for auditing and metric collection

- Encryption: RDS and ALB should be encrypted at rest and in transit

- Backups: RDS configured with 7-day backup retention for data recovery

## Conclusion

The infrastructure solves many of the security concerns with private subnet use with RDS, a public-facing ALB for the web tier with secure SGs, and the creation of instances via Auto Scaling and Launch Templates to ensure consistent set up. These choices in design provide a secure and scalable AWS environment, whilst managing cost, performance and operational complexity.