EGL211 Project

Group 2

[TEAM MEMBER 1]

[TEAM MEMBER 2]

[TEAM MEMBER 3]

[TEAM MEMBER 4]

ERI WANG

Summary

This project focuses on designing and implementing a secure, scalable, and highly available AWS cloud environment to host a public web application. The solution adopts a two-tier architecture consisting of application and database layers deployed within a custom Virtual Private Cloud (VPC) across multiple Availability Zones to ensure fault tolerance and reliability.

Network segmentation is achieved through the use of public and private subnets, route tables, Internet and NAT gateways, and security groups configured according to the principle of least privilege.

Application scalability and availability are enhanced using an Application Load Balancer and Auto Scaling Group, while Amazon RDS MySQL with Multi-AZ deployment provides a reliable and resilient database backend. Data durability and operational continuity are supported through Amazon S3 storage, automated EBS snapshots, and AWS Backup.

Additional security measures include encrypted EBS volumes, controlled administrative access through a bastion host, Cloudflare traffic filtering, and Zero Trust authentication to protect the application from direct public exposure. Monitoring and alerting are implemented using Amazon CloudWatch and Amazon SNS to improve operational visibility and incident response. Overall, the project demonstrates how integrated AWS services can be used to build a secure, resilient, and production-like cloud infrastructure while balancing performance, scalability, and security requirements.

Table of Content

Table of Responsibilities

TABLE 1

TABLE OF RESPONSIBILITIES

Group Member	Task Allocated	Report Contribution	Remark
[TEAM MEMBER 1]	VPC configuration, Database Implementation	Summary, 6.2.2 Database Implementation, Individual Reflection, Recommendations(RDS Proxy, Secret manager, Blue/Green RDS version), Conclusion,Appendices	VPC configuration - 20% Database Implementation - 100%
[TEAM MEMBER 2]	VPC configuration, EBS Snapshot	6.4 EBS Snapshot, Individual Reflection, Recommendations(Vault lock, Malware protection, Cross-Account copy), Conclusion, References	VPC configuration - 20% EBS Snapshot - 100%
[TEAM MEMBER 3]	VPC configuration, Load Balancer & EC2 implementation.	6.2.1 Load Balancer & EC2, Individual Reflection, Recommendations(IPv6 implementation& Stricter IAM), Conclusion, Appendices, References	VPC configuration - 20% Load Balancer & EC2 Implementation - 100%

[TEAM MEMBER 4]	VPC configuration, Bastion, Additional Security Measures	6.3 Bastion, 6.7 Addition security measure, Individual Reflection,Conclusion, Appendices, Recommendation	VPC configuration - 20% Bastion - 100% Additional Security Measures - 60%
ERI WANG	VPC configuration, S3, Additional Security Measures	Provide structure of report, standardize formatting. Task 6.1, 6.5, 6.6, 6.7 (agent forwarding). Recommendation (S3), Conclusion, References, Appendices	VPC configuration - 20% S3 - 100% Additional Configuration - 100% Additional Security Measures - 40%

Introduction

This Project aims to create an AWS environment to host a public web server to manage management/services, and our primary goal is to create a secure, scalable, and highly available cloud infrastructure for the customer.

Key objective of this project:

1. Virtual Private Cloud (VPC) with public and private subnets across 2 Availability Zones (AZs) with controlled outbound internet access for private instances for patching
2. Implement scalability and high availability for application and database tiers
3. Implement secure administration using a bastion host in a public subnet
4. Implement snapshots on the storage volume to provide business continuity
5. Deploy an object storage bucket for private uploads with authentication while enabling private App downloads without using access keys stored in the application code
6. Apply any additional security measures where appropriate

Body

6.1 Cloud architecture, networking and segmentation

VPC has the IPv4 CIDR of "10.0.0.0/20" to create a subnet with the size of 4091 IP, which is sufficient for current use and future expansion.

Subnets are created with the following rules in mind:

- AWS has 5 reserved IP addresses for every subnet
- Large subnets are allocated first for optimal efficiency
- Subnet CIDR follows the "Rule of Thirds" to ensure sufficient IP address capacity for immediate needs while allowing room for growth and unexpected expansion
- Public subnets have a smaller size compared to private subnets to minimise attack surfaces
- Private subnets containing EC2 instances have larger size to accommodate for horizontal scaling
- Private subnets hosting the database have a smaller size as the database relies more on vertical scaling.
- Refer to *fig. 5* for the Conceptual 2-tier application implementation architecture diagram

6.1.1 VPC Creation

The foundation of the network is a custom VPC providing a private, isolated space within the AWS Cloud. Refer to *fig. 6* For VPC creation settings

1. Navigate to the VPC dashboard and select **Create VPC**.
2. Check **VPC only** under **Resources to create**.
3. Use name tag of **grp2-prod-use1-vpc** following the naming convention of {groupNo}-{environment}-{az}-vpc
4. Enter **10.0.0.0/20** for **IPv4 CIDR**
5. Keep the default selection of **No IPv6 CIDR block** for IPv6 CIDR

6. Keep the default selection of **Default** for Tenancy
7. VPC encryption control is set to **None** to confine to the budget.
8. Click **Create VPC**

TABLE 2

VPC CONFIGURATION

VPC Name	Region	Subnet CIDR	IGW
grp2-prod-use1-vpc	us-east-1	10.0.0.0/20	grp2-prod-use1-igw

6.1.2 Subnet Creation

To ensure high availability and tier isolation, six subnets are created across two Availability Zones under **grp2-prod-use1-vpc**. Subnets are named with the convention of {groupNo}-{environment}-{az}-{visibility}-sn. The subnets configurations are made with reference to *Table. 3*

TABLE 3

SUBNET CONFIGURATION

Subnet name	Availability Zone	CIDR
grp2-prod-use1a-public-sn	us-east-1a	10.0.2.0/26
grp2-prod-use1a-app-priv-sn	us-east-1a	10.0.0.0/24
grp2-prod-use1a-db-priv-sn	us-east-1a	10.0.2.128/27
grp2-prod-use1b-public-sn	us-east-1b	10.0.2.64/26
grp2-prod-use1b-app-priv-sn	us-east-1b	10.0.1.0/24
grp2-prod-use1b-db-priv-sn	us-east-1b	10.0.2.160/27

An Internet Gateway is to be created to allow communication between the VPC and the internet.

1. In the left navigation pane, choose **Internet gateways** and then **Create internet gateway**.
2. Use the name **grp2-prod-use1-igw** following the naming convention {groupNo}-{environment}-{region}-igw
3. Click **Create an internet gateway**
4. At the Internet gateway dashboard, click **Actions** and then **Attach to VPC**
5. Select **grp2-prod-use1-vpc** and click **Attach internet gateway**.

A NAT Gateway is to be created to allow the instances in private subnets to reach the internet. Its type is Regional to simplify the multi-AZ architecture.

1. In the left navigation pane, choose **NAT gateways** and then **Create NAT gateway**
2. Use the name **grp2-prod-use1-nat-gw** following the naming convention of {groupNo}-{environment}-{region}-nat-gw
3. Check **Regional - *new***.
4. Select the VPC **grp2-prod-use1-vpc**.
5. Check **Automatic** for the Method of Elastic IP allocation
6. Click **Create NAT gateway**

6.1.3 Creating Route Tables

Private and Public subnets will have their own route table respectively, using the naming convention {groupNo}-{environment}-{az}-{visibility}-rt. Both are attached to **grp2-prod-use1-vpc**.

In each Route table, configurations are made with reference to *Table. 4*.

The public route table has both public subnets explicitly associated, forwards all traffic to the Internet Gateway; private route table has all private subnets explicitly associated, forwards all

traffic to the NAT Gateway. The private route table is set as the main route table for the VPC as a security best practice. This ensures that if a new subnet is created and someone forgets to associate it, it defaults to being private rather than public.

TABLE 4

ROUTE TABLE CONFIGURATIONS

Route Table Name	Destination	Target	Explicit Subnet Association	Is main route table
grp2-prod-use1-public-rt	0.0.0.0/0	Internet Gateway (grp2-prod-use1-igw)	• grp2-prod-use1a-public-sn • grp2-prod-use1b-public-sn	No
grp2-prod-use1-priv-rt	0.0.0.0/0	NAT Gateway (grp2-prod-use1-nat-gw)	• grp2-prod-use1a-app-priv-sn • grp2-prod-use1b-app-priv-sn • grp2-prod-use1a-db-priv-sn • grp2-prod-use1a-db-priv-sn	Yes

6.1.4 Verification of VPC Setup

To ensure the architecture setup is complete:

- Verify **grp2-prod-use1-vpc** is available under **Your VPCs** as shown in *fig. 3*
- Verify IGW **grp2-prod-use1-igw** is attached to VPC **grp2-prod-use1-vpc** under **Internet gateways**.
- Verify NAT Gateway **grp2-prod-use1a-nat-gw** is available under **NAT gateways**.
- Verify route tables are only routing their respective subnets to the respective gateway under the VPC Resource map (*fig. 4*)

6.2.1 Load Balancer & EC2

Create security groups to enforce least-privilege access:

- Application Load Balancer Security Group (grp2-prod-alb-sg): Allow HTTP and HTTPS from prefix group **Cloudflare IPv4**
- Web Server Security group (grp2-prod-webapp-sg): Allow inbound HTTP from grp2-prod-alb-sg, Allow SSH inbound from grp2-prod-bastion-sg
- Create Prefix List configuration according to step *Table. 7*.

TABLE 5

Configurations	**grp2-prod-alb-sg**	**grp2-prod-webapp-sg**
VPC	grp2-prod-use1-vpc	grp2-prod-use1-vpc
Inbound rules	Type: HTTP Source: Cloudflare IPv4 Type: HTTPS Source:Cloudflare IPv4	Type:HTTP Source: grp2-prod-alb-sg Type:SSH Source: grp2-prod-bastion-sg
Outbound rules	Type: All Traffic Source: 0.0.0.0/0	Type: All Traffic Source: 0.0.0.0/0

SECURITY GROUPS CONFIGURATION

Enable EBS Encryption to secure data-at-rest

1. In the left navigation pane, select **Dashboard**
2. In the upper-right corner of the page, choose Account Attributes, **Data protection and security**.
3. In the EBS encryption section, choose **Manage**.
4. Select **Enable**. Keep the AWS managed key with the alias aws/ebs created on your behalf as the default encryption key

5. Choose **Update EBS encryption**.

Deploy Private Web Server (grp2-prod-webapp-ec2)

- Create a temporary web server EC2 instance which serves as the initial web server where Apache (httpd) is installed. The EC2 instance acts as the base configuration for creating a reusable Amazon Machine Image(AMI).
- In the **AWS management console**, in the search box next to **Services** , search for and select **EC2.**
    - EC2 name and tags: grp2-prod-webapp-ec2
    - Instance type: t2.micro
    - Key pair: vockey
    - Network settings (VPC): grp2-prod-use-vpc
    - Network settings (Subnet): grp2-prod-use1a-app-priv-sn
    - Network settings (Auto-assign public IP): disable
    - Network settings (Firewall security groups): grp2-prod-webapp-sg
    - IAM instance profile: LabInstanceProfile
    - User data: - *Refer to Fig. 7*

Creation of Amazon Machine Image (grp2-prod-webapp-ami)

1. In the left navigation pane, select **Instances.**
2. Select the newly configured EC2: grp2-prod-webapp-ec2
3. In the actions menu, choose image and templates > Create image, then configure:
    - Image name: grp2-prod-webapp-ami
    - Image description: configured webserver instance

Target group Creation (grp2-prod-webapp-tg)

- In the left navigation pane, choose **Target Groups.**

- At the top of the screen, click on the "**Create target group**" button. Configure according to *table. 6*.

TABLE 6

TARGET GROUP CONFIGURATION

Configurations	**grp2-prod-webapp-tg**
Target type	Instances
Target group name	grp2-prod-webapp-tg
Protocol/Port	HTTP/80
IP address type	IPv4
VPC	grp2-prod-use1-vpc
Protocol version	HTTP1

Creation of SSL Certificate to enable HTTPS access to webapp (To be used in ALB deployment)

A self-signed SSL certificate was generated locally using OpenSSL via Git Bash to ensure secure private key generation and handling.

The certificate is then imported into AWS Certificate Manager and attached to the HTTPS (port 443) listener of the Application Load Balancer. This allows the ALB to handle encrypted HTTPS traffic securely.

-Refer to Fig. 8 for SSL Certificate Issued for domain egl211.rirori.xyz in AWS Certificate Manager.

- Open Git Bash and enter command retrieved from Tencentcloud(2025):

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes -subj "/CN=egl211.rirori.xyz"
```

This will produce: Both **cert.pem**(certificate Body), **key.pem**(Private Key)

- On the keyboard press Ctrl + X
- Type: %USERPROFILE%
- Right-click both cert.pem & key.pem and copy its contents.
- In the **AWS management console**, in the search box next to **Services** , search for and select **AWS Certificate Manager.**
- In the left navigation pane, choose **Import certificate.**
- Under **Certificate body** paste cert.pem contents.
- Under **Certificate Private Key** paste key.pem contents.

Deployment of Application Load Balancer (grp2-prod-web-alb)

An Application Load Balancer (ALB) is deployed to distribute incoming HTTP traffic across multiple application instances to ensure scalability and high availability.

- In the left navigation pane, choose **Load Balancers.**
- At the top of the screen, click on the "**Create load balancer**" button.
    - Type: Application Load Balancer (Layer 7)
    - Load balancer name: grp2-prod-web-alb
    - Scheme: Internet-facing
        - This allows public users to access the ALB directly from internet users through a single public DNS endpoint.
    - VPC: grp2-prod-use1-vpc
    - ALB is deployed across two public subnets in 2 separate Availability Zones:
        1. us-east-1a (grp2-prod-use1a-public-sn)
        2. us-east-1b (grp2-prod-use1b-public-sn)
    - Security group: grp2-prod-alb-sgRouting action: forward to target groups (grp2-prod-webapp-tg)
- Under Listeners and Routing, apply the configuration according to *table. 7*

TABLE 7

LISTENERS AND ROUTING CONFIGURATION

Configurations	**HTTPS:443**	**HTTP:80**
Protocol	HTTPS	HTTP
Port	443	80
Routing Action	Forward to target groups (grp2-prod-webapp-tg)	Redirect to URL(URI parts) Protocol: HTTPS Port: 443

- Under Secure listener settings:
    - Certificate source: From ACM
    - Certificate: Select the certificate imported earlier (egl211.rirori.xyz)
    - *Refer to Fig. 9 for ALB deployment review page.*

Create Simple Notification Service

SNS is implemented to provide automated notification(Email) for Auto Scaling and health-related events.

- Refer to Fig. 10 for AWS Notification in action.

- In the AWS Management console, Search and select **Amazon SNS.**
- In the left navigation pane, choose Topics.
- At the top of the screen, click on the "**Create topic**" button.
    - Type: Standard
    - Name: grp2-prod-asg-sns
- In the left navigation pane, choose Subscription.
    - Topic ARN: grp2-prod-asg-sns

- Protocol: Email
- Endpoint: [email of TEAM MEMBER 3]

Create Launch Template and Auto Scaling Group

A launch Template is created to define the configuration settings that the Auto Scaling Group will use when provisioning the new EC2 instances.

- In the left navigation pane, choose **Launch Templates.**
- At the top of the screen, click on the "**Create launch template**" button.
  - Launch template name: grp2-prod-webapp-launchtemplate
  - Description: a launch template for ASG
  - Auto Scaling guidance: **enabled**
  - Under the Application and OS Images (Amazon Machine Image) area, choose "My AMIs".
  - Click on "Owned by me"
  - Amazon Machine Image(AMI): grp2-prod-webapp-ami
  - Instance type: t2.micro
  - Key pair: vockey
  - Under Network settings area:
    - Subnet: Don't include in launch template
    - Security Group: grp2-prod-webapp-sg
  - Under Advanced details:
    - IAM instance profile: LabInstanceProfile
    - User data: -*Refer to Fig. 7 for user data script*
- Select the newly configured Launch template: grp2-prod-webapp-launchtemplate
- From the **Actions** menu, select "create Auto Scaling group"
- Create Auto Scaling group configuration according to *table. 8.*

TABLE 8

AUTO SCALING GROUP CONFIGURATION

Step 1	
Auto Scaling group name:	**grp2-prod-webapp-asg**
Launch template:	**grp2-prod-webapp-launchtemplate**
Step 2	
VPC:	**grp2-prod-use1-vpc**
Availability Zones and Subnets:	**grp2-prod-use1a-app-priv-sn** **grp2-prod-use1b-app-priv-sn**
Availability Zone distribution:	**Balanced best effort**
Step 3	
Select load balancing options:	**attach to existing load balancer**
Attach to an existing load balancer:	**Choose from your load balancer target groups (grp2-prod-webapp-tg)**
Additional health check types:	**Enable Turn on Elastic Load Balancing health checks**
Step 4	
Desired capacity:	**2**
Min desired capacity:	**2**
Max desired capacity:	**6**
Automatic scaling:	**Target tracking scaling policy**
Scaling policy name:	**grp2-prod-webapp-scalingpolicy**
Metric type:	**Application Load Balancer request count per target**
Target group:	**grp2-prod-webapp-tg**

Target value:	**1000**
Instance warmup:	**120**
Instance maintenance policy:	**launch before terminating (Max: 110)**
Monitoring:	**Enable group metrics collection within CloudWatch**
Step 5	
SNS Topic:	**grp2-prod-asg-sns**
Event types:	**Launch,Terminate, Fail to launch, Fail to terminate**
Step 6	
Key:	**Name**
Value:	**grp2-prod-webapp-asg-instance**

Auto scaling Group configured keeps a **minimum of 2 instances** running for high availability so that if one instance fails, application is able to run without any significant downtime.

ELB is enabled to ensure traffic is only routed to healthy operational instances.

The auto scaling is set to scale after receiving more than **1000 requests** per target instance based on recommendations WEI Xin. (2025, December 5)

SNS and Cloudwatch are implemented as well for monitoring.

-Refer to fig. 11. and fig. 12. for a failover demonstration.

6.2.2 Database Implementation

Create Security Group for Database

1. In the AWS Management Console, navigate to **VPC**.
2. In the left navigation pane, choose **Security Groups.**
3. Click **Create security group**.

TABLE 9

SECURITY GROUP CONFIGURATIONS

Security group name:	**grp2-prod-use1-db-sg**
Description:	**grp2-prod-use1-db-sg**
VPC:	**grp2-prod-use1-vpc**
Inbound rule	
Type:	**MySQL/Aurora**
Source:	**Custom**
	grp2-prod-webapp-sg
Outbound rule	**Leave as Default**

Refer to Appendix: *Fig. 13* **-** for the console screenshot verifying these settings.

4. Click **Create security group**

Create DB Subnet Group

1. In the AWS Management Console, navigate to **RDS**.
2. In the left navigation pane, choose **Subnet groups**.
3. Click **Create DB Subnet group**.

TABLE 10

SUBNET GROUP CONFIGURATIONS

Subnet group name:	**grp2-prod-use1-db-subnet-group**

Description:	**grp2-prod-use1-db-subnet-group**
VPC:	**grp2-prod-use1-vpc**
Add subnet	
Choose Availability zones:	**us-east-1a**
	us-east-1b
Subnets:	**grp2-prod-use1a-db-priv-sn**
	grp2-prod-use1b-db-priv-sn

Refer to Appendix: *Fig. 14* **-** for the console screenshot verifying these settings.

4. Click **Create**

Create Database

1. In the AWS Management Console, navigate to **RDS**.
2. In the left navigation pane, choose **Databases**.
3. Click **Create database**.

TABLE 11

DATA CONFIGURATIONS

Database creation method:	**Full configuration**
Engine type:	**MySQL**
Templates:	**Production**
Availability and durability:	**Multi-AZ DB instance deployment (2 instances)**
Settings	
DB instance identifier:	**grp2-prod-mysql-db**

Master username:	**admin**
Credentials management:	**Self managed**
Password:	**12345678**
Confirm password:	**12345678**
Instance configuration	
DB instance class:	**Burstable classes (including t classes)**
Instance type:	**db.t3.micro**
Storage	
Storage type:	**General-purpose SSD (gp3)**
Allocated storage:	**20GB**
Connectivity	
Compute resource:	**Don't connect to an EC2 compute resource**
VPC:	**grp2-prod-use1-vpc**
DB subnet group:	**grp2-prod-use1-db-subnet-group**
Public access:	**NO**
VPC security group (firewall):	**Choose existing**
Existing VPC security groups:	**grp2-prod-use1-db-sg**
	Remove Default
Monitoring	
Enhanced Monitoring:	**Uncheck Enable Enhanced monitoring**
Additional configuration	
Database options	

Initial database name:	**Proj**

Refer to Appendix: *Fig. 15* to *Fig. 22* **-** for the console screenshot verifying these settings.

4. Click **Create database**

Create Read replica

1. In the AWS Management Console, navigate to **RDS**.
2. In the left navigation pane, choose **Databases**.
3. Select **grp2-prod-mysql-db.**
4. Under **Action** on the top right select **Create read replica**

TABLE 12

READ REPLICA CONFIGURATIONS

Replica source:	**grp2-prod-mysql-db**
DB instance identifier:	**grp2-prod-mysql-db-replica**
Monitoring	
Enhanced Monitoring:	**Uncheck Enable Enhanced monitoring**

5. **Leave everything else as default.**
6. **Refer to Appendix:** *Fig. 23* to *Fig. 24* **-** for the console screenshot verifying these settings.
7. Click **Create read replica.**

Refer to Appendix: *Fig. 26* - for the console screenshot verifying how the final outcome should look like and confirming the instance status is "Available".

Refer to Appendix: *Fig. 27* **-** for the console screenshot verifying the database cannot be publicly accessed.

Refer to Appendix: *Fig. 28* **-** for the console screenshot verifying the database allows access from the authorized grp2-prod-webapp-sg security group.

Refer to Appendix: *Fig. 29* **-** for the console screenshot verifying the successful creation of the Table products in database Proj.

Refer to Appendix: *Fig. 30* - for the console screenshot verifying the successful insertion of records in Table products in database Proj.

6.3 Bastion host administration

1. Create a Security group (grp2-prod-bastion-sg) with an inbound rule to allow anywhere for SSH

2. Create an EC2:

Refer to *Fig. 31 for the SSH into bastion and through bastion to private application* .

TABLE 13

BASTION HOST EC2 CONFIGURATION

The instance name:	**grp2-prod-bastion-ec2**
Application:	**Amazon Linux**
Amazon Machine Image (AMI)	**Amazon Linux 2023 kernel*6.1 AMI**
Architecture:	**64-bit**
Instance type:	**t2.micro**
Key pair:	**vockey**

VPC:	**grp2-prod-use-vpc**
Auto-assign:	**Disable**
Firewall security group:	**grp2-prod-bastion-sg**
Description:	**Allow ssh**
Configure storage:	**1x8 Gib gp3**
Advanced details:	**Detailed CloudWatch monitoring: enabled**

3. Creation of elastic IP:

Refer to *Fig. 33 Elastic IP to show that it is working*

TABLE 14

ELASTIC IP ASSIGNMENT TO BASTION

Public ipv4 address pool:	**Amazon pool of ipv4 address**
Network border group:	**us-east-1**
Resource type:	**instance**
Elastic IP address:	**3.230.188.191**
Instance:	**bastion id (i-0eae864c8eb14d335)**
Private IP:	**10.0.2.50**

6.4 EBS Snapshot

Create tag for volume of EC2 Web Server - This allows the user to easily identify which Volume of EC2 they want to backup later on.

1. Type in **EC2** at Search at the top of the browser
2. In the left navigation pane, click on **Instances** and **tick the empty box** beside "**grp2-prod-webapp-ec2**".
3. Click on the Storage tab then click on volume under **Block devices**. This will bring you to the Volume attached to Web Server **grp2-prod-webapp-ec2.**
4. Tick the empty box then navigate to **Tags** tab, click on **Manage tags, Add tag,** enter **Name** under **Key**, enter **grp2-prod-webapp-data-ebs** under **Value** then click Save. Refer to *fig. 34* for the result.

Create automated snapshots using AWS Backup - Vault creation, EC2 automated snapshots are organized and protected here. It uses an AWS KMS key to ensure that data is encrypted at rest and is physically stored in Amazon S3.

1. Type in **AWS Backup** at search at the top of the browser
2. In the left navigation pane, click on **Vaults** then **Create new vault**.

TABLE 15

Configuration	**Vault**
Vault name	grp2-prod-backup-vault
Vault type	Backup vault
Encryption key	(default) aws/backup

VAULT CONFIGURATION

Create backup plan for automated snapshot creation - Backup plan creation, setting of time schedule and resources to be backed up are configured here.

1. In the left navigation pane, click on **Backup plans** then **Create backup plan**

TABLE 16

BACKUP PLAN CONFIGURATION

Configuration	**Create backup plan**
Backup plan name	grp2-prod-backup-plan
Backup rule name	grp2-prod-daily-backup
Backup vault	grp2-prod-backup-vault
Logically air-gapped vault	No vault selected
Backup frequency	Daily
Backup window (Start time)	04:00 America/New_York (UTC-05:00)
Start within	1 Hour
Complete within	2 Hour
Point-in-time recovery	Unchecked
Cold storage	Unchecked
Total retention period	14 Days
Backup indexes	Unchecked
Copy to destination	No region selected
Tags added to recovery points	Key - Name Value - grp2-auto-ebs-snapshot
Malware protection	Disabled
Advanced backup settings	Windows Vss - Unchecked

	Back up ACLs - Unchecked Back up object tags - Unchecked

Click create plan

Assign resources to backup plan (grp2-prod-backup-plan)

TABLE 17

RESOURCE ASSIGNMENT TO BACKUP PLAN

Resource assignment name	**grp2-prod-web-daily**
Choose an IAM role	LabRole
Include specific resource types	EBS Volume ID - grp2-prod-webapp-data-ebs
Exclude specific resource IDs from the selected resource types	Default
Refine selection using tags	Default

Click assign resources

Refer to *fig. 35* & *fig. 36* For the result of an automated snapshot created.

Refer to *fig. 37* To view snapshot created under Elastic Block Store in EC2

Create automated testing of snapshots created - Automated way to perform test restores of backups to verify integrity and data usability, monitors the restoration time and deletes the test resource once validation is complete to save costs.

1. In the left navigation pane, click on **Restore testing.**
2. Click on **Create restore testing plan.**

TABLE 18

SNAPSHOT TESTING CONFIGURATION

Restore testing plan name	**grp2_prod_snapshot_test**
Test frequency	**Daily**
Start time	**5:00 America/New_York(UTC-05:00)**
Start within	**1 Hour**
Recovery point selection	**Specific vaults - grp2-prod-backup-vault**
Eligible recovery points	**14 Days**
Selection criteria for point-in-time recovery (PITR)-enabled resource types	**Unchecked**
Tags added to restore testing plan	**Nil**

Click create restore testing plan

Assign resources to snapshot testing plan (grp2_prod_snapshot_test)

TABLE 19

ASSIGN RESOURCES TO SNAPSHOT TESTING

Resource assignment name	**grp2_prod_webapp_data_snapshot**
Choose an IAM role	LabRole
Retention period before cleanup	12 Hours
Protected resources	Resource type - EBS
Include specific protected resources of this resource type	Vault - grp2-prod-backup-vault
Protected resources to include	grp2-auto-ebs-snapshot
Restore parameters	Default

Click assign resources

Refer to *Fig. 38* & *fig. 39* For result of automated testing on snapshot.

6.5 S3

Two S3 buckets are to be created. Bucket names follow the naming convention of {groupNo}-{environment}-{instance}-{usage}-s3. All S3 buckets are publicly available as IAM roles cannot be assigned in the Learner Lab.

- **grp2-prod-webapp-webfiles-s3** for containing the webapp source code to be hosted on the webapp EC2 Instance in the private subnet.
- **grp2-prod-webapp-uploads-s3** for file upload and download

TABLE 10

S3 BUCKET CONFIGURATION PARAMETERS

Configuration	**grp2-prod-webapp-webfiles-s3**	**grp2-prod-webapp-uploads-s3**
AWS Region	us-east-1	us-east-1
Bucket Type	General Purpose	General Purpose
Object Ownership	ACLs disabled	ACLs disabled
Block all public access	Unchecked	Unchecked
Bucket versioning	Enabled	Enabled
Encryption Type	SSE-S3	SSE-S3

Bucket key	Enabled	Enabled
Object lock	Disabled	Disabled

Creation of grp2-prod-webapp-webfiles-s3 and grp2-prod-webapp-uploads-s3 S3 buckets

1. Navigate to S3 and click **Create bucket**
2. Configure the security and encryption settings as specified in *table. 10***.**

Additional Configurations for grp2-prod-webapp-webfiles-s3

1. In the left navigation pane, select **General purpose buckets**
2. Select **grp2-prod-webapp-webfiles-s3** and click **Upload**
3. A folder named **egl211** containing all the webpage files is added. Click **Upload**. This folder will automatically have the default encryption settings configured. Refer to *Fig. 43* for the content of the bucket.
4. Select **grp2-prod-webapp-webfiles-s3** and navigate to the **Permissions** tab
5. Under the Bucket policy, click on **Edit**
6. Enter the policy in *fig. 40* and click **Save changes**. This policy will allow the EC2 instance hosting the webapp to fetch website files from the bucket.

Configurations for grp2-prod-webapp-uploads-s3

1. In the left navigation pane, select **General purpose buckets**
2. Click on **grp2-prod-webapp-uploads-s3** and navigate to the **Permissions** tab
3. Under Bucket policy, click on **Edit** and fill up according to *fig. 41*
4. Enter the policy in the figure and click **Save changes**. This policy will allow anyone to list all the files, upload and download from the bucket.
5. Under Cross-origin resource sharing (CORS), click on **Edit**

6. Enter the configuration in the *fig. 42* and click **Save changes**. This policy will allow the uploading function. **AllowedOrigins** is set to **egl211.rirori.xyz** to only allow the webapp to access the S3 bucket.
7. Navigate to the **Objects** tab and click on **Create folder**
    a. Folder name: **uploads**
    b. Server-side encryption: Specify an encryption key
    c. Encryption settings: Use bucket settings for default encryption
8. Click **Create folder**

Content in the grp2-prod-webapp-uploads-s3 is shown in *fig. 44*. The webapp is allowed to list, get and put shown in *fig. 50*

6.6 Additional Configurations

Prefix Group

Following the principle of least privilege, restricting the Load Balancer's security group to only allow Cloudflare IP addresses forces all traffic through Zero Trust authentication. Without this restriction, an attacker could bypass the authentication page entirely by sending requests directly to the Load Balancer's public AWS DNS. By whitelisting only Cloudflare's verified IP ranges, the AWS infrastructure is hidden from the open internet, mitigating direct DDoS attacks and ensuring that every user must first be verified against the **egl211** GitHub organization before reaching your private EC2 instances.

1. Navigate to the **VPC** dashboard and in the left navigation pane, select **Managed prefix list** and click on **Create prefix list**
2. Add the list of cloudflare IP into the prefix list. The list of IP addresses can be retrieved from Cloudflare(2023).
    - Prefix list name: Cloudflare IPv4
    - Max entries: 15

- Address family: IPv4

DNS

A CNAME DNS record is created with Cloudflare to simplify access to the load balancer URL. Route 53 in AWS cannot be used due to restricted permissions.

1. Access the Cloudflare Dashboard at dash.cloudflare.com. The **Domain Management** page should be the default landing page as shown in *fig. 49*.
2. Choose the domain to be used **rirori.xyz** from the list of domains
3. In the left navigation pane, select **DNS > Records**, highlighted in *Fig. 48*.
4. Click **Add record**.
   - Type: CNAME
   - Name: egl211 (as subdomain)
   - Target: grp2-prod-web-alb-1073011350.us-east-1.elb.amazonaws.com (ALB DNS url)
   - Proxy status: Proxied (for security)
   - Click **Save**
5. The webpage through the load balancer can now be accessed at egl211.rirori.xyz as shown in *fig. 47*

Enable authentication to webapp

To secure the webapp, a **Zero Trust** layer was implemented using Cloudflare Access. The application at egl211.rirori.xyz is protected by an identity-aware proxy that requires authentication before traffic is routed to the internal AWS Application Load Balancer. This eliminates the risk of public exposure for the private S3 upload/download functionalities.

1. Access the Cloudflare Zero Trust at one.dash.cloudflare.com
2. In the left navigation pane, select **Access controls > Applications**, shown in *fig. 51*
3. Click **Add an application** and select **Self-hosted**. Refer to *fig. 45*

4. Click **Add public hostname**. Configure according to *table. 11*. An overview of the configuration can be viewed at *fig. 46*

TABLE 11

CLOUDFLARE ADD SELF-HOSTED APPLICATION

Configuration	Value
Application name	GRP2 AWS Portal
Session Duration	No duration, expires immediately (for security)
Input method	Default
Subdomain	egl211
Domain	rirori.xyz

5. Under Access Policies, click **Create new policy**. Perform the configuration according to *table. 12*. The policy created will allow only users within the GitHub organization. Click **Save** after configuring.

TABLE 12

CLOUDFLARE POLICY TO ALLOW GITHUB ORGANISATION ONLY

Configuration	Value
Policy name	GRP2 AWS Portal Auth

Action	Allow
Session Duration	No duration, expires immediately (for security)
Input method	Default
Subdomain	egl211
Domain	rirori.xyz

6. Go back to **Add an application** page and click **Select existing policies**. Select **GRP2 AWS Portal Auth** from the list and click **Confirm**
7. Under **Login methods**, select **Github** and **Save**
8. Refer to *fig. 52 for the configured policy*

6.7 Additional Security Measures

Bastion

Bastion fail2ban IP ban for failure to enter the correct key. The command in *Fig. 1* is configured

```
sudo yum install fail2ban -y

sudo systemctl start fail2ban
sudo systemctl enable fail2ban
```

in the Bastion EC2 to install, start, and enable fail2ban service at startup.

Fig. 1. Fail2ban Installation

The commands in *fig. 2* are used to set up fail2ban for SSH connections to the bastion EC2.

```
sudo nano /etc/fail2ban/jail.local

[sshd]
enabled = true
port = ssh
logpath = /var/log/fail2ban.log
maxretry = 3
bantime = 600

sudo systemctl restart fail2ban
sudo systemctl status fail2ban

sudo fail2ban-client status
sudo fail2ban-client status sshd
```

Fig. 2. Fail2ban Configurations for SSH

Refer to *Fig. 32 for the IP ban example.*

SSH into private instance from Bastion EC2 through Agent Forwarding

The ssh command used to connect to the Bastion EC2 has the flag “-A” to allow agent forwarding. This allows the use of local SSH key sets to authenticate the EC2 in private subnets. Doing so will eliminate the need to transfer and store the private key onto the Bastion, enhancing security as the sensitive private key never leaves the local machine.

Individual Reflection

[TEAM MEMBER 1]

In this project, I believe I performed well in designing and setting up the database according to the system's needs and requirements. I carefully considered factors such as performance, scalability, reliability, and security during the implementation. One key achievement was configuring a read replica to support the main database. The read replica helps handle read queries, reducing the load on the primary database and improving overall performance. Additionally, it enhances high availability and provides an extra layer of backup, ensuring better fault tolerance and system reliability. I also ensured that the database was not publicly accessible by placing it in a private subnet and configuring security groups to allow access only from the web application. This strengthened the overall security of the system and protected sensitive data.

However, one area I did not perform well in was my limited familiarity with SQL commands. At times, I struggled to recall certain queries and had to rely on documentation or online references, which slowed down my development process. Furthermore, due to account limitations, I was unable to fully explore and implement more advanced database features that could further enhance performance and security.

If I were to do this project again, I would first improve my proficiency in SQL so that I can design queries more efficiently and manage the database with greater confidence. A stronger foundation in SQL would help me optimize queries, structure tables more effectively, and troubleshoot issues more systematically.

In addition, I would implement advanced AWS features such as Amazon RDS Proxy, AWS Secrets Manager, and Blue/Green deployment for RDS, using an account configured with the appropriate IAM role policies. By integrating AWS Secrets Manager, I would avoid hardcoding database usernames and passwords in the application code. Instead, the application would securely retrieve credentials at runtime, improving security and reducing the risk of credential exposure.Rather than connecting directly to the RDS endpoint, I would configure the application

to connect through RDS Proxy. This would enhance connection pooling, improve scalability, and increase overall system reliability, especially under high traffic conditions.Furthermore, by implementing Blue/Green deployment for RDS, I would be able to test new code and database changes in the Green environment without affecting the main production database (Blue). Once testing is successful, I could perform a controlled switchover, minimising downtime and reducing deployment risks. This approach would significantly improve system stability, security, and maintainability.

[TEAM MEMBER 2]

During this project, I configured automated snapshot backup using AWS Backup to ensure data protection and support business continuity. Compared to manual snapshot creation, AWS Backup provides these advantages:

Centralized management - AWS Backup provides a centralized dashboard to monitor and manage backup and restore activities, e.g. successful/unsuccessful Backup jobs, successful/unsuccessful Restore jobs, providing convenience for users and centralized visibility.

Automation - Allows users to create a routine for automated snapshot creation, instead of manually creating snapshots, this reduces efforts and provides consistency as users might forget to create a snapshot which can put data at risk.

Cost efficiency - AWS Backup has a lifecycle and retention feature which deletes the snapshot after a specified period of time, this helps to save costs as older snapshots will be deleted, making sure that you only pay for what you require. Compared to taking a manual snapshot and forgetting to delete it which can accumulate charges.

Reduced human error - Manual snapshot creations are prone to errors, such as deleting the wrong snapshot, creating a snapshot of the wrong volume or missing the retention date. With AWS Backup, users will only need to configure backup policies once, eliminating the need for manual snapshot creation and reducing operational risks.

Restore testing - Periodically verifies data integrity without manual effort. The service selects the most recent recovery point, restores it into an isolated environment and ensures that the data is usable and uncorrupted. After the specified time frame of 12 hours, the resource will be deleted automatically to minimize costs. This feature reduces the need to manually test if backup snapshots are usable providing convenience and efficiency.

Another point I want to highlight is why I chose AWS Backup instead of using Lambda + EventBridge. I am aware of the advantages that Lambda provides such as flexibility to code which resource it wants to back up and where it backs up to, however this approach takes much more effort as it requires custom code, more prone to mistakes, code has to be maintained, monitor failure, if the script has a bug, the business loses data and it does not have a centralized view like AWS Backup. I chose AWS Backup because the requirement asks for Business Continuity, AWS Backup has a fully Managed Governance Service that provides automated backup scheduling, automated restore testing, centralized monitoring and reduced operational overhead. AWS Backup offers reliability, governance and ease of use, making it more suitable for enterprise-level data protection compared to custom-scripted solutions.

[TEAM MEMBER 3]

Throughout this project, one of the key strengths was successfully implementing a multi-tier cloud architecture that decoupled layers, allowing for independent scaling and ensuring high availability for application and database tiers. I was able to deploy an Application Load Balancer across 2 public subnets while applying the principle of least privilege by restricting ALB to only permit Cloudflare IPv4 in AWS managed prefix list to ensure restricted access from the public. Additionally, I ensure that web applications only allow inbound HTTP traffic from Application Load Balancer security group and SSH access only from Bastion Host security group, demonstrating strong security measures rather than relying on default settings set by AWS.

Beyond Security controls, I also configured Auto scaling group spanning across 2 availability zones to deploy EC2 Web applications on private subnets, triggering scalability after receiving

1000 requests per target instance. I have also implemented a minimum of 2 EC2 web instances to be running simultaneously for failover to achieve high availability to ensure that if one instance fails, the application is able to run without any significant downtime.Additionally, i have also implemented HTTPS by attaching an SSL certificate to the ALB listener to establish a secure and encrypted communication.

Although the deployment of ALB and ASG was functional, one area I could have improved on was the scaling policy. I set the threshold of 1000 requests per target based on assumption instead of data-driven analysis. In a more professional setting, Scaling policies were implemented based on traffic patterns and load testing results before determining appropriate scaling policies, for example, Amazon.com has seasonal traffic peaks in November during events like Black Friday and Cyber Monday, so fixed capacity would leave about 76 percent of resources idle most of the year; this shows that scaling is configured based on data-driven analysis. However in this project, the scaling metric was configured based on google searches and personal assumption with little benchmarking. I strongly believe that a more strategic approach can be implemented like manual testing to decide on the metric value to ensure that the scaling of the EC2 is aligned with realistic workload rather than just pure assumption.

If I were to redo this project I would implement a more data-driven scaling policy instead of setting a scaling value threshold based on assumption. I would first perform load testing on the EC2 instance to determine how many requests the web application EC2 can handle before it begins to underperform and then configure the scaling threshold based on the EC2 instance capability. This ensures that the scaling decision on the ASG is based on data rather than estimation.

Additionally, implementing a CDN layer using Amazon CloudFront will help significantly improve performance and scalability. In this project, every web request is forwarded to ALB and EC2. Hence being implementing AWS CloudFront, commonly accessed content can be stored at the nearest edge location so that when content is accessed again, it is delivered from the nearest edge location instead of origin server. Reducing traffic to the EC2 instance and improving responsiveness and website latency.

Lastly, For ASG Enabling scale-in protection for selected instances would prevent critical EC2 instances from being terminated during scale-in events. Even though in my architecture, currently a minimum of 2 instances is running for high availability however,during scaling events ASG could terminate a healthy but important instance as without protection, it chooses which instance to terminate automatically. Hence enabling scale-in protection helps prevent risk of unnecessary termination and improve stability during traffic fluctuations.

[TEAM MEMBER 4]

In this project, I implemented a bastion host and learned how it plays a critical role in securing cloud infrastructure. Initially, we configured an Elastic IP for the bastion host to allow external administrative access and from a security perspective, I understood that SSH access should ideally only be allowed from a specific administrator public IP address. However, we did not own a fixed public IP in this project, thus SSH inbound access had to be allowed from anywhere. This made me realise the importance of balancing accessibility with security risks in real-world environments.

While configuring the bastion, I saw the significance of restricting SSH access and why limiting exposure is an essential to reduce potential attack surfaces. I learned that using smaller subnets helps to minimise attack as it reduces opportunities for attackers to scan or exploit unused addresses. Another important lesson was disabling password authentication and enforcing SSH key-based authentication. This significantly strengthens security because key-based authentication is far more difficult to brute-force compared to traditional passwords.

I also gained a deeper understanding of proper network architecture design. The bastion host must be placed in a public subnet so administrators can access it from the Internet, in the meantime, application and database servers should remain in private subnets without public IP addresses. This is to ensure that sensitive resources are not available to the internet. SSH access to private servers should only be permitted through the private server’s security group, creating a secure access path.

In addition, I learned the importance of monitoring and logging. Enabling AWS CloudWatch allows administrators like us to detect suspicious activities and monitor access patterns to see if there is anything off. This project also introduced me to more advanced security practices such as using AWS Systems Manager Session Manager as an alternative to traditional bastion hosts. By doing so, organisations can eliminate public SSH exposure entirely and further strengthen their security posture, but it does not have the necessary tools to edit files inside the server or let older machines have access to the service.

While researching, I found how Fail2ban works, as it helps to protect the bastion from brute force attacks. When an attacker obtains the bastion host's public IP address and tries to access SSH with incorrect credentials multiple times, Fail2ban will automatically detect all failed login attempts and after three unsuccessful logins, the attacker's IP address will be temporarily banned from accessing the SSH service.

Overall, this project helped me to truly understand the security principles behind bastion host deployment. I now better appreciate how careful planning, restricted access, monitoring, and modern management tools work together to build a secure and resilient cloud environment.

ERI WANG

My implementations in the project were centered on building a secure and highly available web tier that leveraged cloud-native automation and identity-based security. One of my primary successes was the integration of Cloudflare Zero Trust to secure my web application and S3 assets. By configuring user authentication before access to S3 upload/download page, I was able to transition from a potentially vulnerable public storage model to a secure, authenticated environment without adding complex authentication code to the backend. This ensured that while the browser could perform high-performance GET and POST actions via CORS, the underlying S3 objects remained private and protected from unauthorized access.

Another area where I excelled was in the automation of the web tier deployment. By developing a robust User Data script within the EC2 Launch Template, I ensured that every instance

provisioned by the Auto Scaling Group was truly "self-healing." This script automatically installed the necessary PHP environment, configured Apache, and synchronized the latest application files from my S3 bucket. This automation allowed the infrastructure to scale horizontally while maintaining consistent code versions across all nodes, fulfilling the requirements for high availability and operational excellence.

A major technical hurdle I never fully resolved within the application code was session persistence across the load balancer. Because the application uses local file-based sessions, users frequently encounter a "Login" redirect when attempting CRUD operations like adding or editing records. This occurred because the Application Load Balancer (ALB) would route the initial login to one instance and the subsequent CRUD request to another instance that lacked the local session file. Troubleshooting this "session drift" highlighted a gap in my initial architectural planning regarding statelessness.

Beyond the technical challenges, I found the coordination of the shared group report to be unexpectedly difficult. Managing a collaborative document across multiple contributors required constant oversight to ensure technical consistency and a unified voice. Specifically, maintaining a standardized figure and table numbering system proved to be a major administrative burden. As sections were added or rearranged, cross-references to specific diagrams and data tables often broke, even with the assistance of an extension that links Table and Figure number to reference. This experience highlighted that clear documentation standards are just as critical to project success as the underlying code.

If I were to do the project again, I would implement a centralized session management system for the webapp. Since Amazon ElastiCache could not be used in the learner lab environment, a database-backed session handler would be implemented in the RDS. Adopting a truly stateless architecture would have permanently solved the session issues I encountered with the ALB. By storing session data in a central "Source of Truth," any instance in the Auto Scaling Group could authenticate a user's request, significantly improving the user experience and the overall reliability of the multi-instance environment.

I would also refine the User Data script to include better error handling and logging. During the project, it was often difficult to debug why an instance failed to synchronize files or configure Apache correctly. By adding logic to output script progress to a custom log file (e.g., /var/log/user-data.log), I would have been able to troubleshoot deployment issues much faster, ensuring a more reliable and observable "self-healing" infrastructure.

In my capacity as the Group Leader, I oversaw the end-to-end integration of the VPC, web tier, and storage components, ensuring that each module aligned with the project's security and availability mandates. This role provided deep insights into the complexities of distributed systems management and the critical importance of clear architectural documentation. Moving from manual configurations to automated deployments through the ALB and User Data taught me that consistency is the foundation of security. I now have a much stronger grasp of how to build resilient, identity-aware cloud applications for the modern web.

Recommendations

IPv6 Implementation

Implementing IPv6 in AWS helps avoid Public IPv4 charges,ensuring and ensures long-term scalability. Unlike IPv4 architecture like this project, it has limited address space and requires NAT gateways to access the internet as private IPv4 addresses are not routable on the public internet and requires the NAT Gateway to translate private IP addresses into a public IP before sending traffic out, which induces more cost.

By implementing IPv6, Private EC2 instances can directly access the internet without NAT as each instance receives a globally unique public IPv6 address, reducing operational costs.

Although IPv6 assigns globally routable addresses to instances, security is maintained by using an Egress-Only Internet Gateway and restrictive security group rules. This ensures that private EC2 in this project can access the internet while preventing unsolicited inbound IPv6 connections. Hence By implementing dual-stack architecture, VPC reduces reliance on NAT Gateways which is costly and IPv4 exhaustion which is highly recommended for long-term scalability.

Stricter IAM

Although the LabInstance IAM role was attached to the EC2 instances to enable secure access to S3 without storing access keys in the application code, the IAM policy attached to this role could not be fully customized due to AWS Learner Lab restrictions. As a result, the S3 bucket policy was configured using "Principal": "*" which allows any EC2 instance to call s3:GetObject as long as they can get hold of the bucket name which poses security risk. If restrictions for IAM role were lifted, public access would be disabled and access would be restricted strictly to the EC2 IAM role ARN and permissions will follow a stricter least privilege principle could be implemented like allowing access to specific S3 buckets and granting the only required actions like s3:GetObject, s3:PutObject. Applying such fine-grained permissions would strengthen overall security of the system by preventing unnecessary access rights.

AWS system manager session manager (bastion) - [TEAM MEMBER 4]

Compared to traditional bastion host access, AWS Systems Manager Session Manager provides a more secure method as it eliminates the need for public SSH exposure and uses IAM authentication with full logging.

RDS Proxy(Database) – [TEAM MEMBER 1]

It is recommended to implement RDS Proxy to improve database performance and scalability, as it efficiently manages and pools connections between the application and the database while providing faster failover during maintenance or unexpected outages. Additionally, RDS Proxy uses its own endpoint rather than directly exposing the database endpoint, which enhances security by reducing direct access to the database.

Secret manager(Database) - [TEAM MEMBER 1]

It is recommended to use AWS Secrets Manager to securely store and manage database credentials, eliminating the need to hardcode passwords in application code. Secrets Manager also supports automatic credential rotation and integrates with RDS and RDS Proxy, which improves security and simplifies credential management while reducing the risk of unauthorized access.

Blue/Green RDS version(Database) - [TEAM MEMBER 1]

It is recommended to implement **RDS Blue/Green Deployment** to perform database updates and schema changes safely. This approach allows testing in a separate, identical environment before switching traffic to the production database, minimizing downtime and providing a quick rollback option if any issues occur, thereby reducing risk and ensuring continuous availability.

AWS Backup - [TEAM MEMBER 2]

Creation of vault lock - Compliance mode, snapshots under this policy cannot be deleted by any users until their lifecycle completes, this ensures that even if a hacker gets Root credentials, they are unable to delete our backups which is enforced by AWS platform.

Enable malware protection - With provided IAM roles, I would enable this feature to provide an additional layer of security to scan backup snapshots and ensure the data is safe and usable and ensure that infected data is not restored.

Cross-Account Copy - Protect against ransomware, automatically copy snapshots to a separate backup account providing an air-gap where the hacker is unable to touch backups stored in the separate account even if they have full access to the production account.

S3 - ERI WANG

To achieve the requirement of public upload and download directly from the client's browser to the S3 bucket, a CORS (Cross-Origin Resource Sharing) policy is required. This prevents browsers from blocking the GET, POST and PUT request when the domain of the webapp which was configured to use (egl211.rirori.xyz) is different from the S3 endpoint. For security, Cloudflare Access is configured before the S3 upload/download page to only allow authenticated users to perform the actions.

To enhance continuity, S3 bucket versioning is enabled on the two S3 buckets used in the project. With bucket versioning enabled, accidental deletion or update of any file can be rolled back immediately without requiring the back up and restoration. This mechanism ensures high data durability without the latency or complexity associated with traditional backup restoration processes.

As the size of the uploads bucket scales, cost for storage will increase. To save cost, previous versions of a file can be deleted after a set amount of time (e.g. 14 days), achievable with bucket versioning enabled. This will clean up , debloat and keep the bucket lean. In addition, objects that have not been accessed can be moved to the Infrequent Access tier or lower with S3 Intelligent-Tiering. This automatic process can reduce at least 40% of storage cost for infrequent accessed objects.

Conclusion

To Conclude, This project has demonstrated the design and implemented a public facing 2-tier web applications with application tier and a database tier to reduce attack surface by placing application and database in private subnets. The overall architecture follows cloud best practices in terms of security, scalability, and high availability.

The VPC was carefully designed using appropriate CIDR allocation and subnet segmentation across 2 Availability Zones. The use of “Rule of Thirds” where subnet sizing is available IP space is divided in thirds 33% for current requirements, 33% for planned growth (next 2-3 years), 33% for unexpected expansion and new services. Public and Private subnets are separated using subnet tables, with internet gateway mapped to public subnets to provide internet access for public resources and NAT gateway mapped to private subnets to allow resources in a private subnet to securely connect to the internet while remaining isolated from the internet, preventing direct access.

Security was implemented based on the principle of least privilege. Security Groups were configured to act as virtual firewalls, controlling inbound and outbound traffic for associated VPC resources which includes restricting web server access to the Application Load Balancer, allowing only cloudflare IPv4 prefix list to access Application Load balancer. While limiting SSH access to private instances only through the bastion host and from the host to bastion. The use of bastion also meant that we would be able to install more tools to help harden the bastion host by using tools like fail2ban, where it would deny any ip that failed to authenticate correctly and properly.

To implement application tier scalability and high availability, an Application Load Balancer is deployed across 2 Availability zones, and an Auto Scaling Group was implemented to automatically launch or terminate EC2 instances based on demand. It is configured with a minimum of 2 instances and a maximum of 6 instances running for high availability. The system is designed to automatically scale-out when the average request count per target exceeds 1000, ensuring that the system maintains performance during traffic spikes. ELB health checks are

enabled ensuring traffic is only routed to healthy operational instances and Cloudwatch monitoring is used to track metrics and trigger scaling events allowing dynamic scaling.

The database was carefully designed and configured to ensure strong security and reliable access control. Network access was restricted so that only the web application tier security group is allowed to connect to the database through MySQL port 3306, preventing unauthorized systems from establishing direct connections. In addition, the database was configured without public accessibility, ensuring it remains isolated within the private network environment and reducing exposure to external threats. This layered security approach strengthens the overall protection of the system by limiting attack surfaces and enforcing strict communication rules between application and database components.

To ensure reliability and performance, the database was deployed using a Multi-AZ configuration, which provides high availability by maintaining a standby instance in a separate Availability Zone for automatic failover during outages or maintenance events. Furthermore, configuring a read replica enhances scalability by offloading read-heavy workloads from the primary database instance, allowing the system to handle higher query volumes more efficiently. Read replicas also improve application performance by reducing latency for read operations and can support disaster recovery strategies by providing additional copies of data that can be promoted if required.

Automated snapshots using AWS Backup were implemented to fulfil the Business Continuity framework. By moving away from manual snapshots, we have established a consistent and audit-ready architecture that focuses on data availability and eliminates risks of human error and recovery uncertainty inherent in custom scripting. AWS Backup is equipped with a centralized dashboard, offering full visibility into backup health and job status, while the integration of Automated Restore Testing ensures that our recovery points are not just stored, but are tested and ensured that they are functioning properly and ready for deployment at any moment. While the current foundation is operating effectively, to further improve it we can implement AWS Backup Vault Lock for immutability and Cross-Account backup for air-gapped security to significantly reduce the risk of ransomware or account compromise. By adding on these features, the

organization's critical data remains protected, verified, highly available, and cost-optimized, whilst adhering to modern cloud resilience and governance best practices.

Monitoring tools like CloudWatch and alerting tools SNS are enabled and implemented to help monitor AWS resources and applications in real time, send automatic email notifications for auto scaling and instance related events. Providing administrators with a monitoring and observability service for AWS resources and applications ensuring faster incident response.

The implementation of the S3 storage tier successfully fulfills the project's requirements for high availability, security and business continuity. By integrating Cloudflare Zero Trust as an authentication layer, the architecture transitioned from a vulnerable public access model to a secure, identity-aware environment, ensuring that only authenticated users can access sensitive application files. The deployment of CORS (Cross-Origin Resource Sharing) policies optimized performance by enabling direct client-side interactions with the buckets, while S3 Bucket Versioning and Intelligent-Tiering established a robust framework for business continuity and long-term cost management. Ultimately, these configurations demonstrate a mature cloud storage strategy that balances seamless user accessibility with enterprise-grade data protection.

Overall, this project demonstrates how multiple AWS services can be implemented to host a 2-tier web applications with an application tier and a database tier by following industry best practices while still providing room for future scalability of the cloud environment.

References

SubnetCalculator Team. (2025, January 22). *AWS VPC subnetting best practices: Reserved IPS and CIDR Planning Guide*. Subnet Calculator for Developers. https://www.subnetcalculator.dev/blog/aws-vpc-subnetting-best-practices

WEI Xin. (2025, December 5). *Effective ECS scaling solutions for various patterns | One Career Tech blog*. note (ノート). https://note.com/dev_onecareer/n/n4b81fd7ed0c1

Cloudflare (Ed.). (2023, September 28). *Ip Ranges*. Connect, Protect and Build Everywhere. https://www.cloudflare.com/ips/

Tencent Cloud (Ed.). (2025, September 12). Using OpenSSL to Generate Self-Signed Certificates. https://www.tencentcloud.com/document/product/1145/65945

Ellingwood, J., & Garnett, A. (2022, August 2). *How fail2ban works to protect services on a linux server*. DigitalOcean. https://www.digitalocean.com/community/tutorials/how-fail2ban-works-to-protect-services-on-a-linux-server

Appendices

4.1 VPC

Fig. 3. Verification of VPC creation

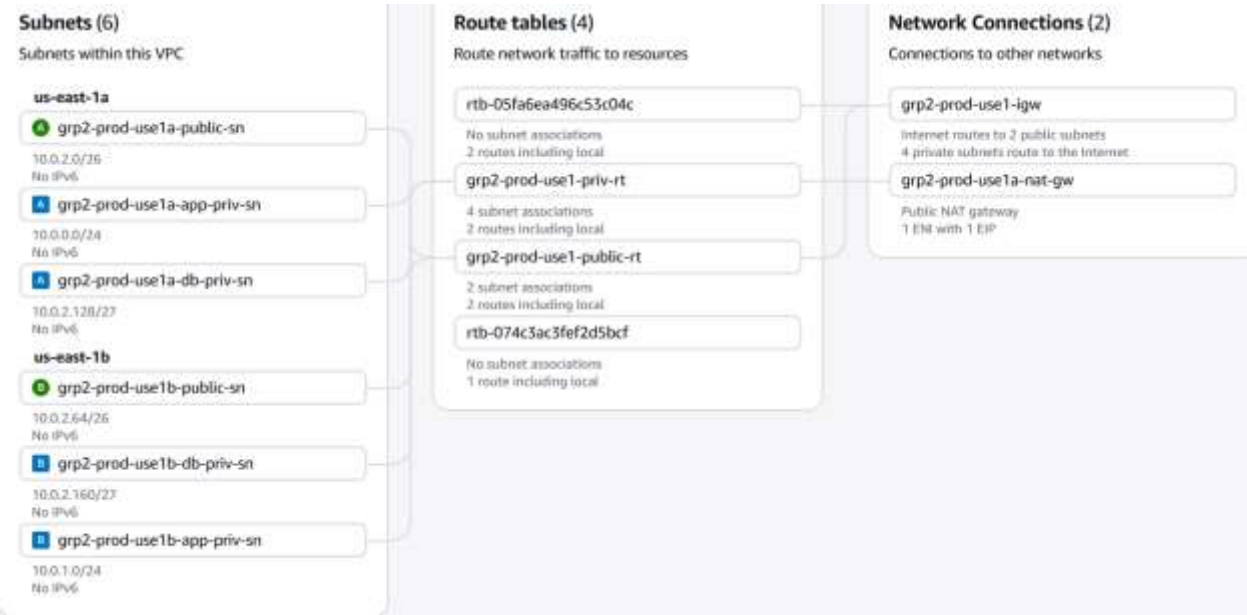

Fig. 4. VPC Resource Map

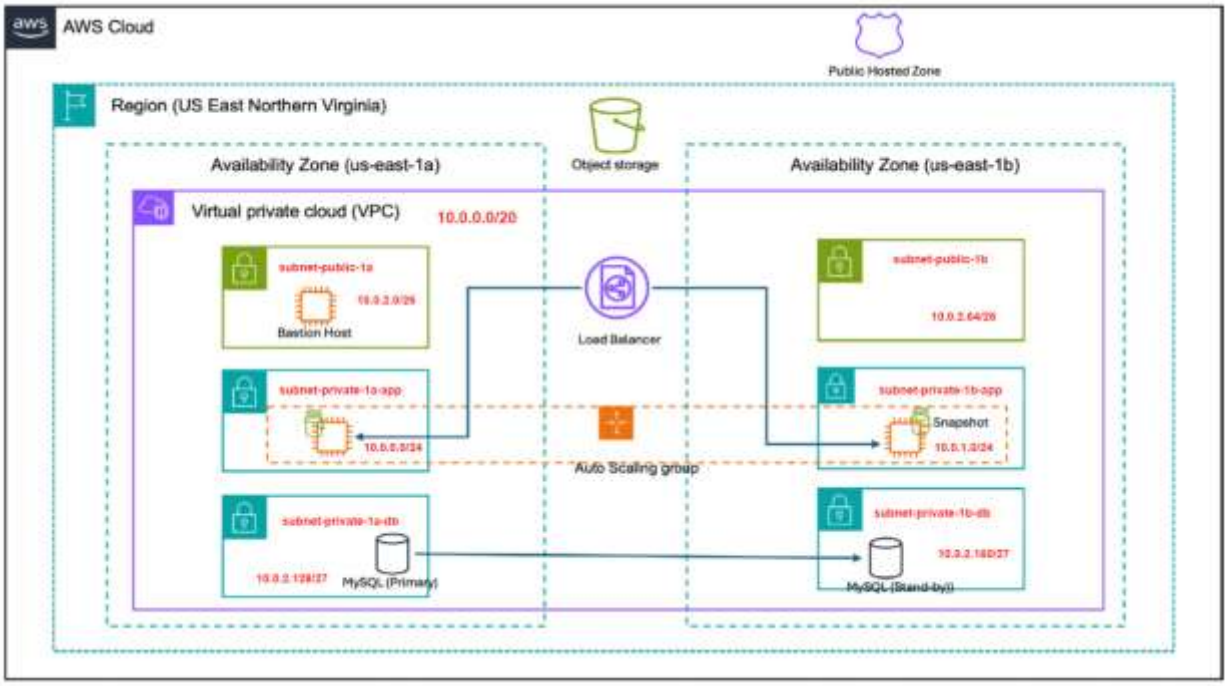

Fig. 5. Conceptual 2-tier application implementation architecture diagram

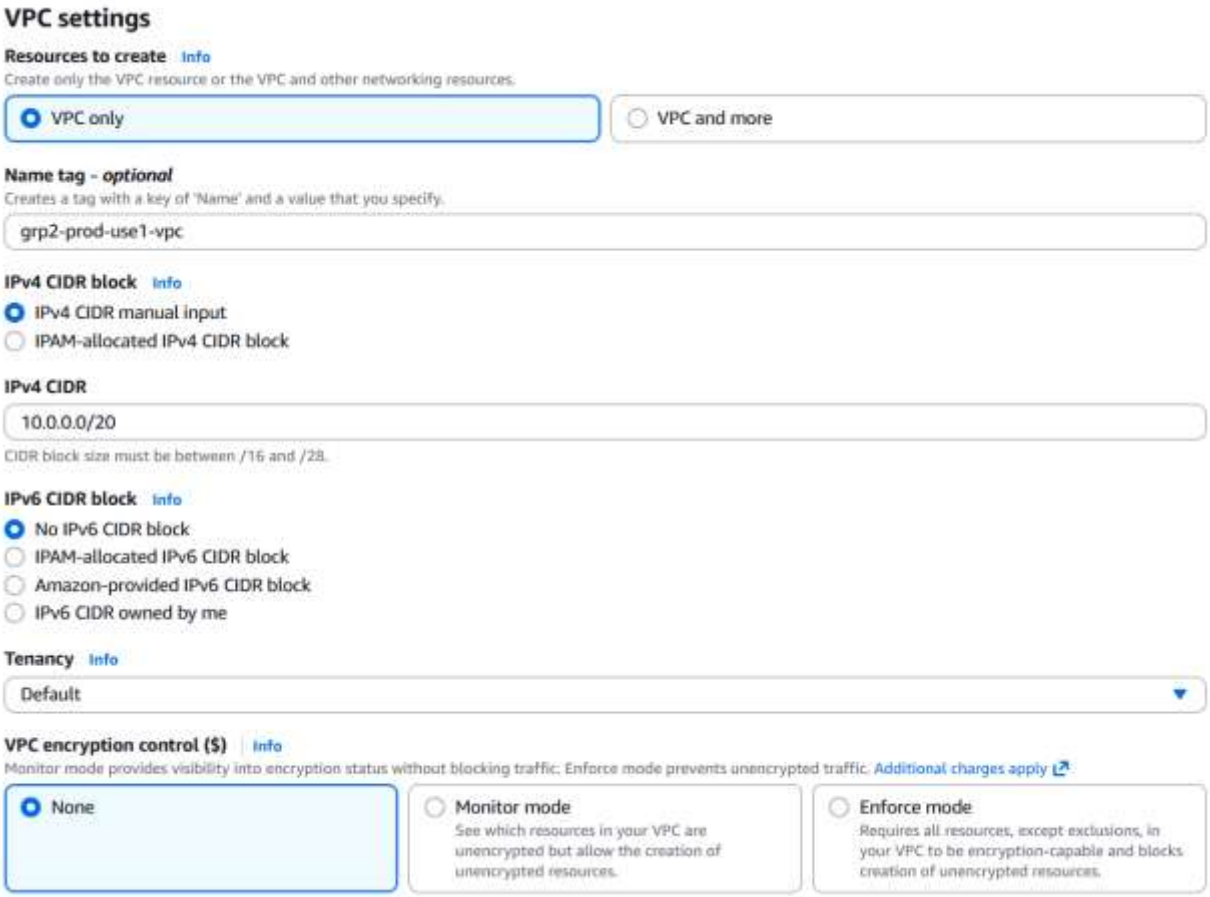

Fig. 6. VPC Creation

4.2.1 Load Balancer - [TEAM MEMBER 3]

```
#!/bin/bash
# 1. Update and install web stack and cron
dnf install -y httpd php php-mysqli mariadb105 cronie cronie-anacron

# 2. Start and enable Apache and cron
systemctl start httpd
systemctl enable httpd

sudo systemctl start crond.service
sudo systemctl enable crond.service

# 3. Initial pull from S3 (Ensures instance is healthy for ALB)
# Note: Ensure LabInstanceProfile is attached to this Launch Template
/usr/bin/aws s3 sync s3://grp2-prod-webapp-webfiles-s3/egl211/
/var/www/html/ --delete

# 4. Set correct permissions for Apache
chown -R apache:apache /var/www/html
chmod -R 755 /var/www/html

# 5. Set up the Weekly Auto-Fetch (Sunday at midnight)
echo "0 0 * * 0 /usr/bin/aws s3 sync s3://grp2-prod-webapp-webfiles-s3/egl211/
/var/www/html/ --delete && chown -R apache:apache /var/www/html" |
crontab -
```

Fig. 7. User data script

Fig. 8. SSL Certificate issued

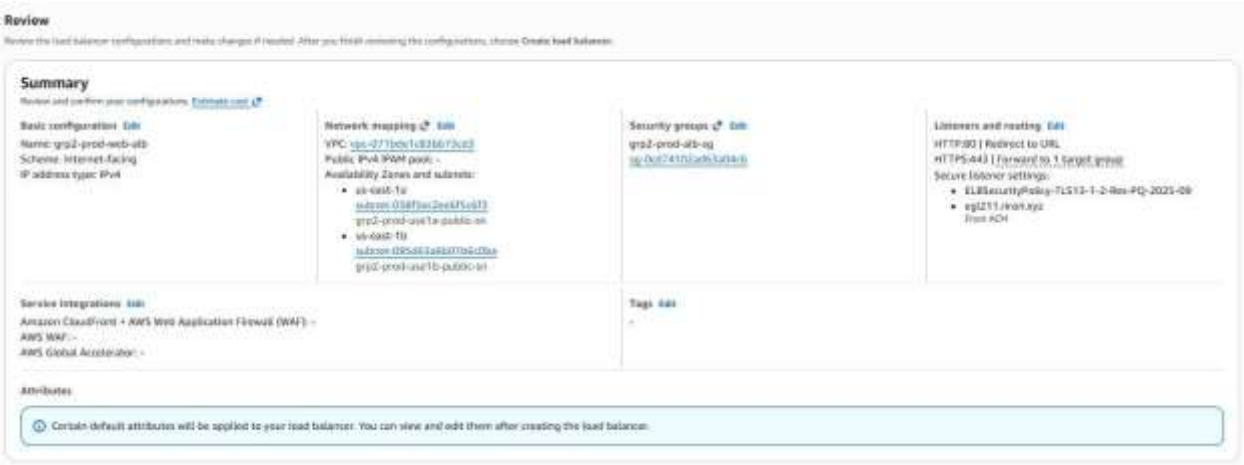

Fig. 9. ALB deployment

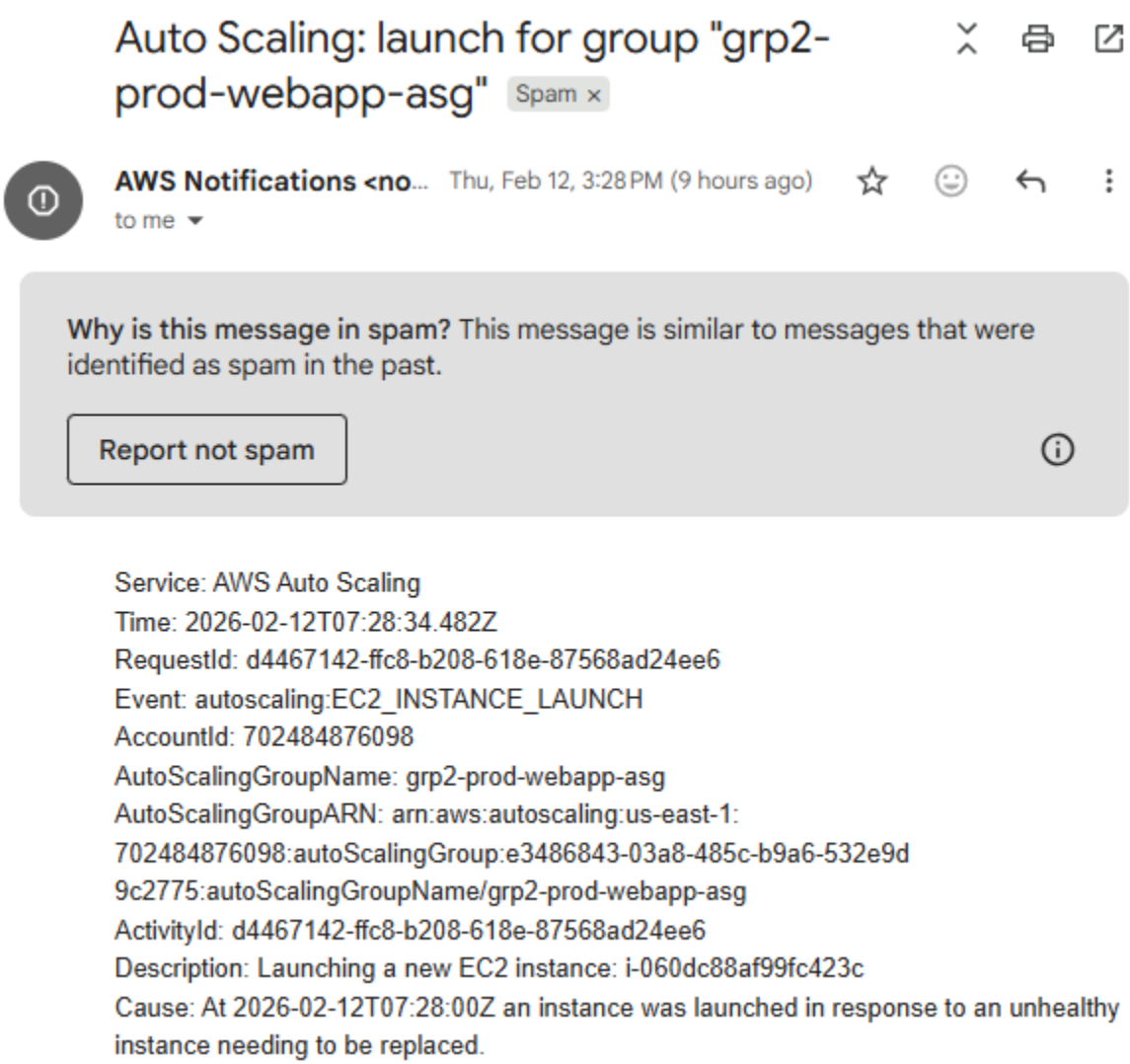

Fig. 10. SNS Via Gmail

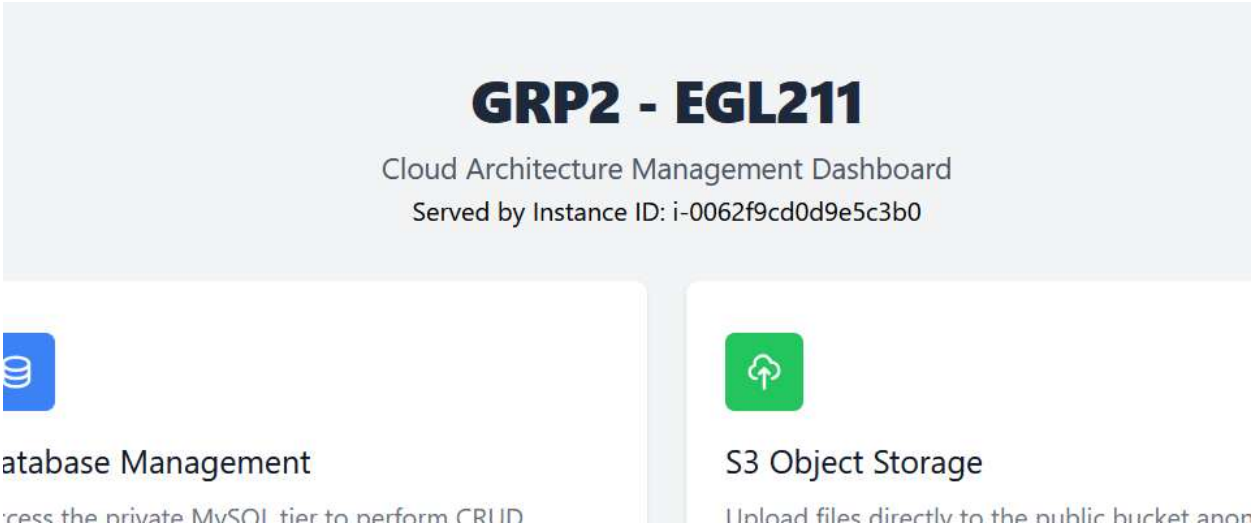

Fig. 11. Instance ID: i-0062f9cd0d9e5c3b0 hosting the web application prior to termination

Fig. 12. Instance ID: i-017fa7c0265c11258 Hosting Web application following ASG failover

4.2.2 Database - [TEAM MEMBER 1]

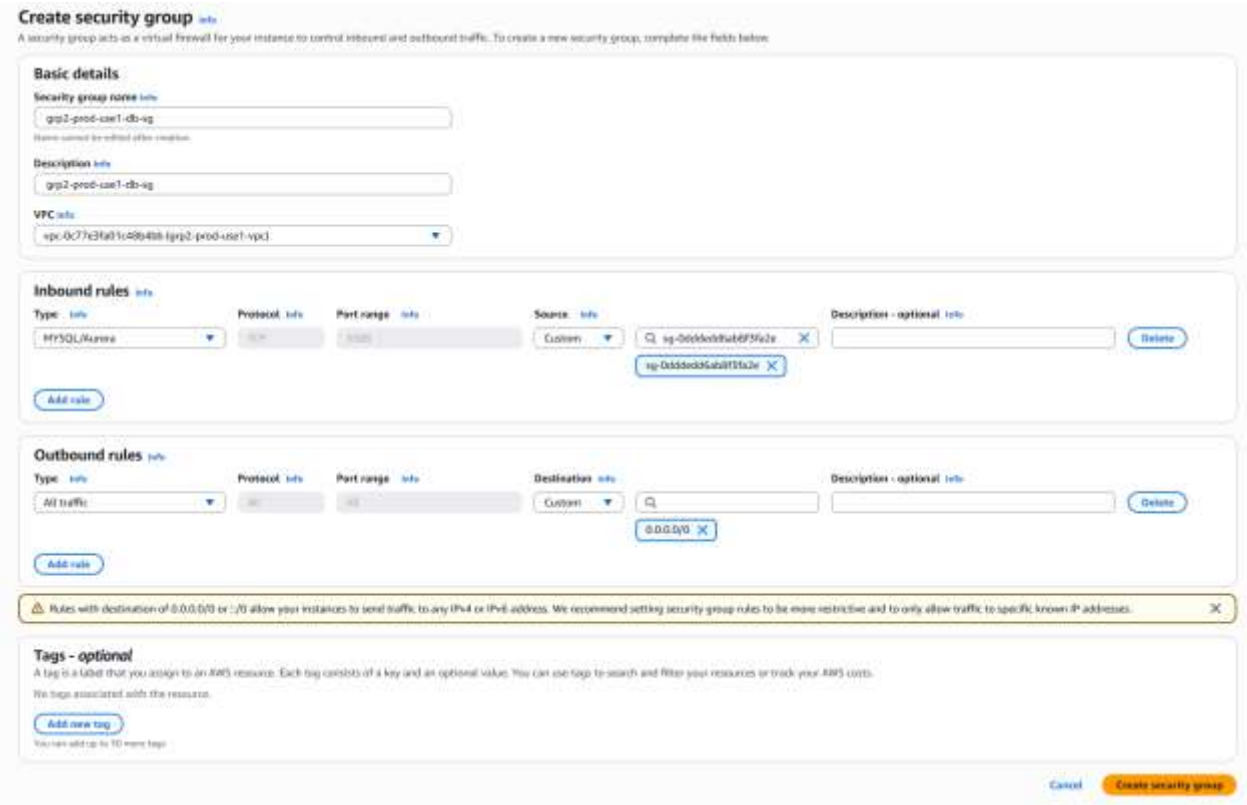

Fig. 13. security group creation

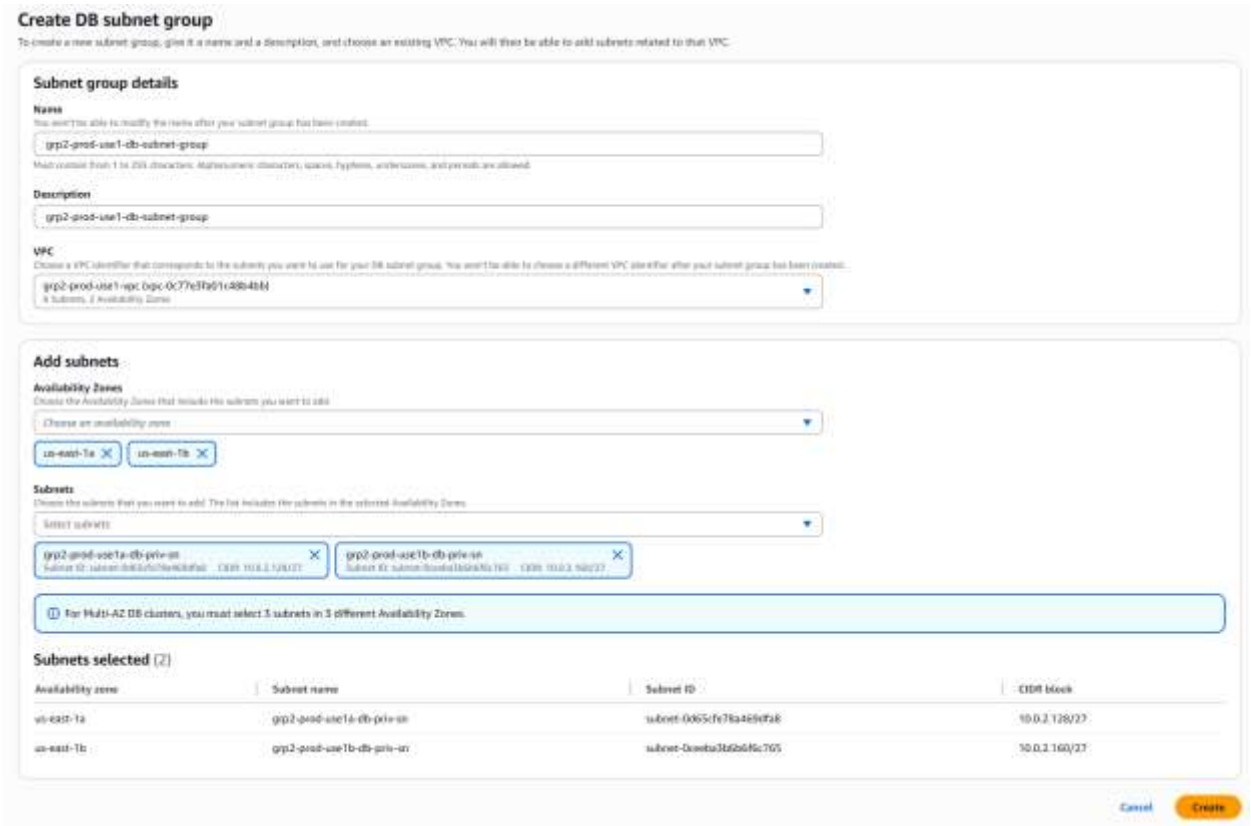

Fig. 14. DB subnet group creation

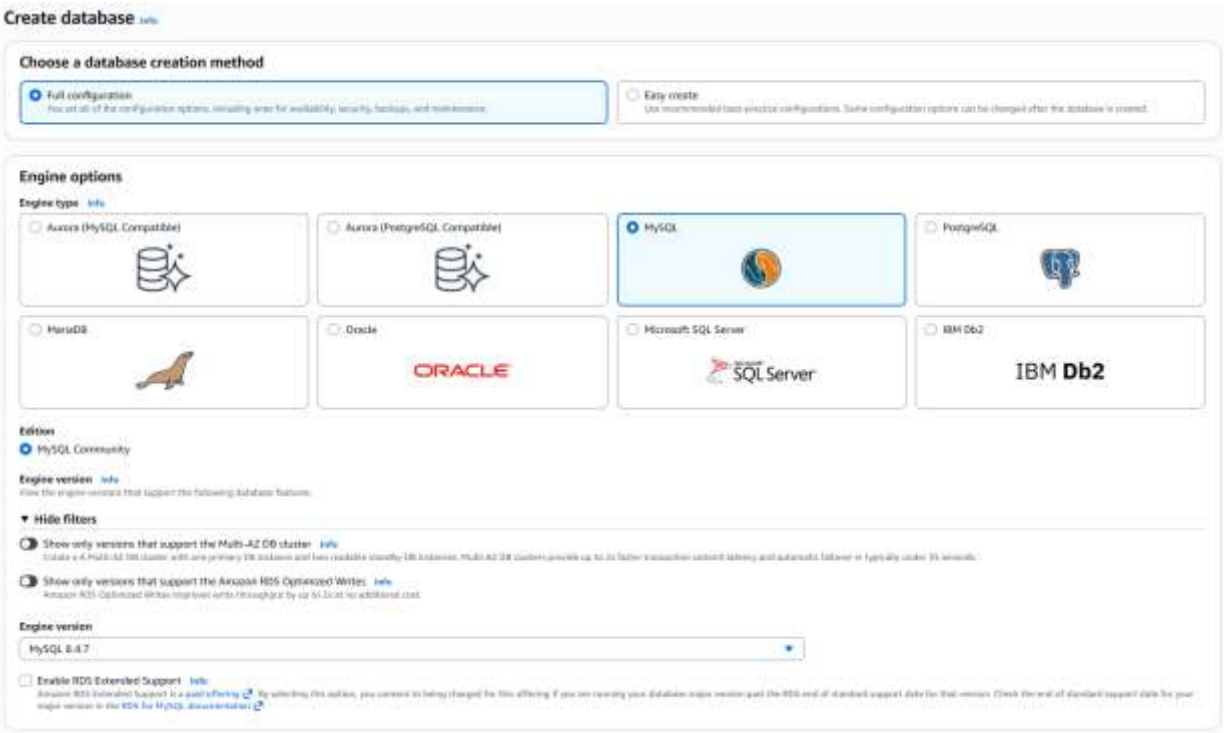

Fig. 15. Database creation.1

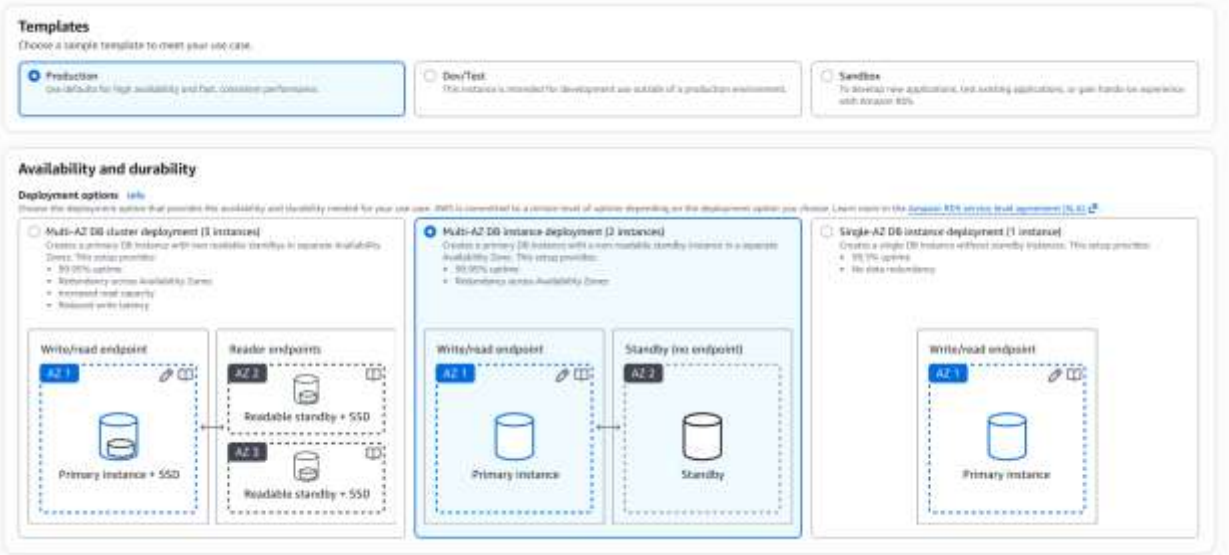

Fig. 16. Database creation.2

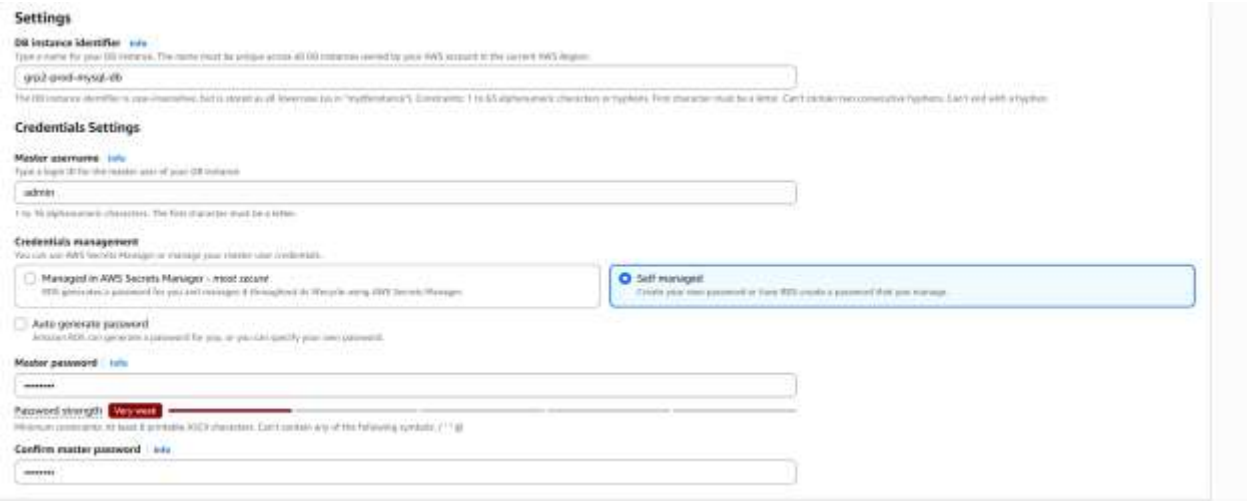

Fig. 17. Database creation.3

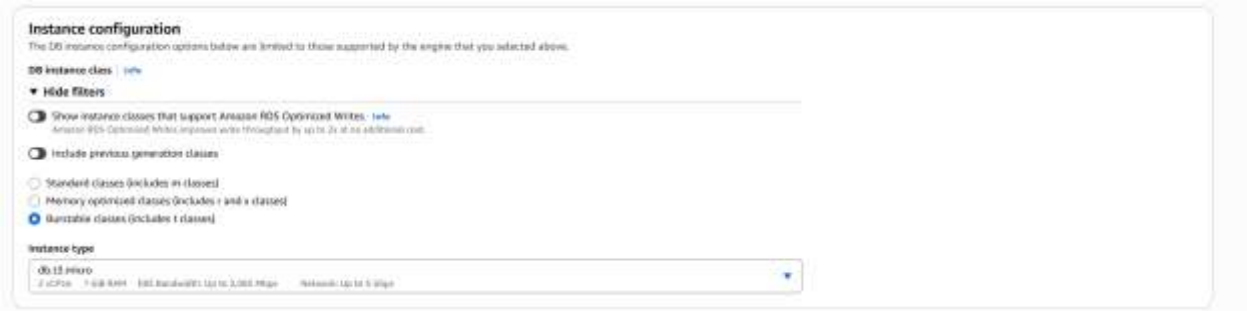

Fig. 18. Database creation.4

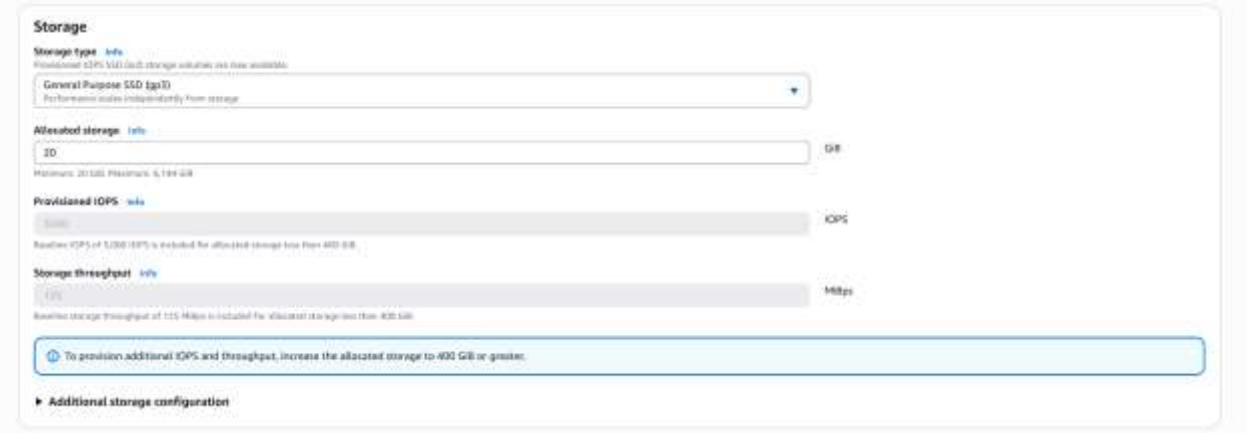

Fig. 19. Database creation.5

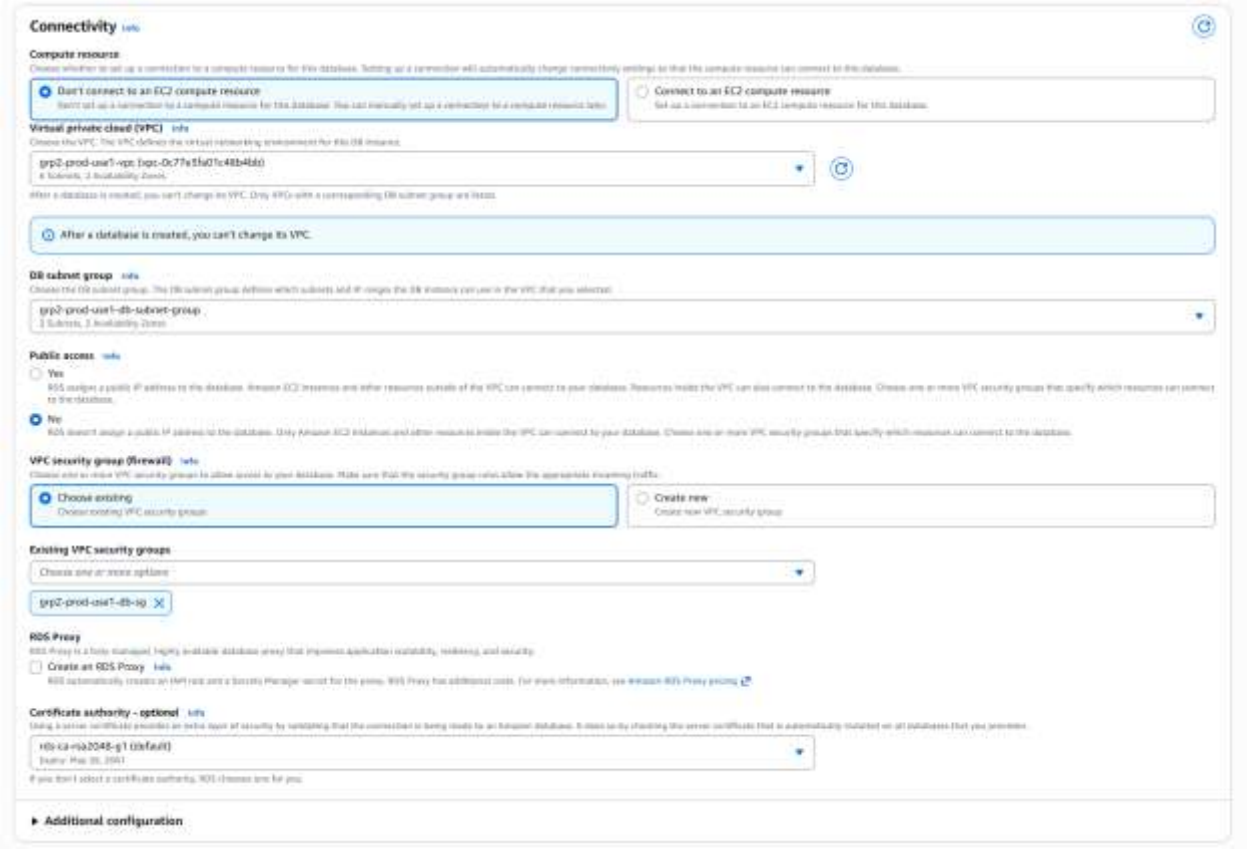

Fig. 20. Database creation.6

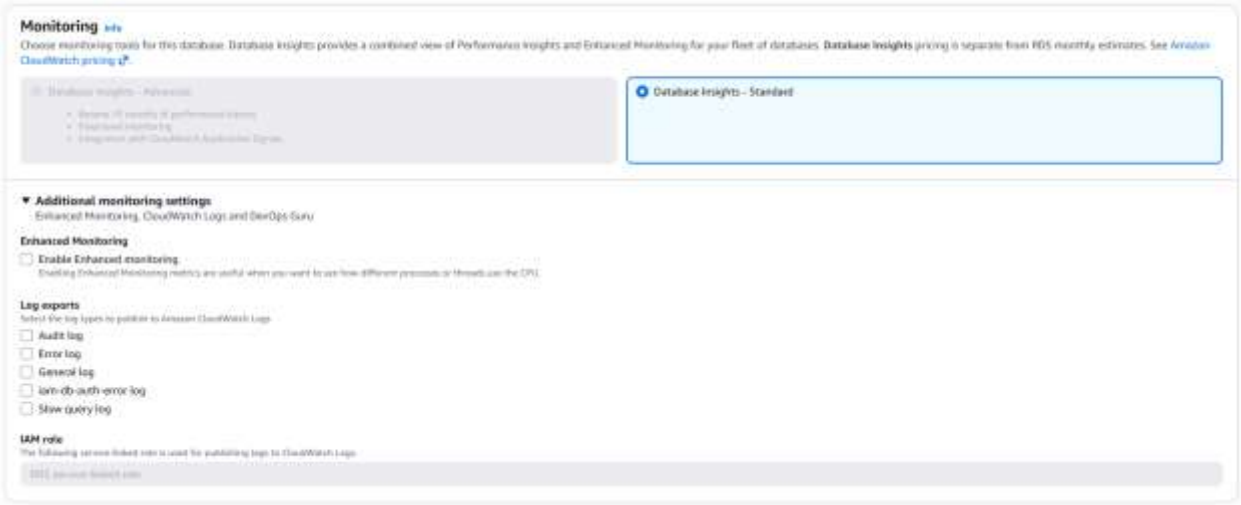

Fig. 21.. Database creation.7

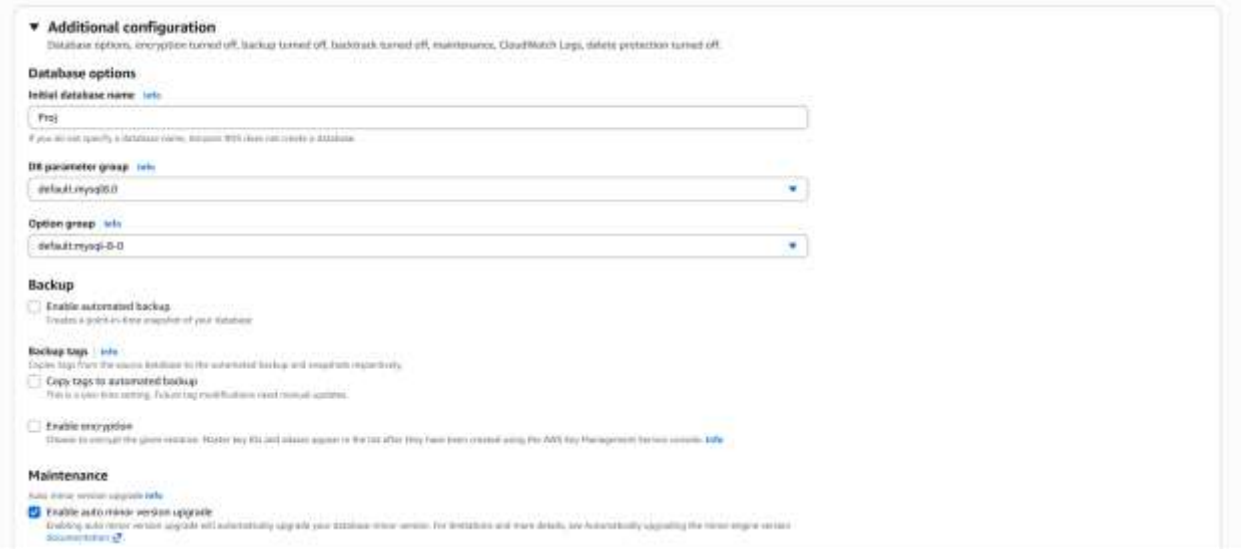

Fig. 22. Database creation.8

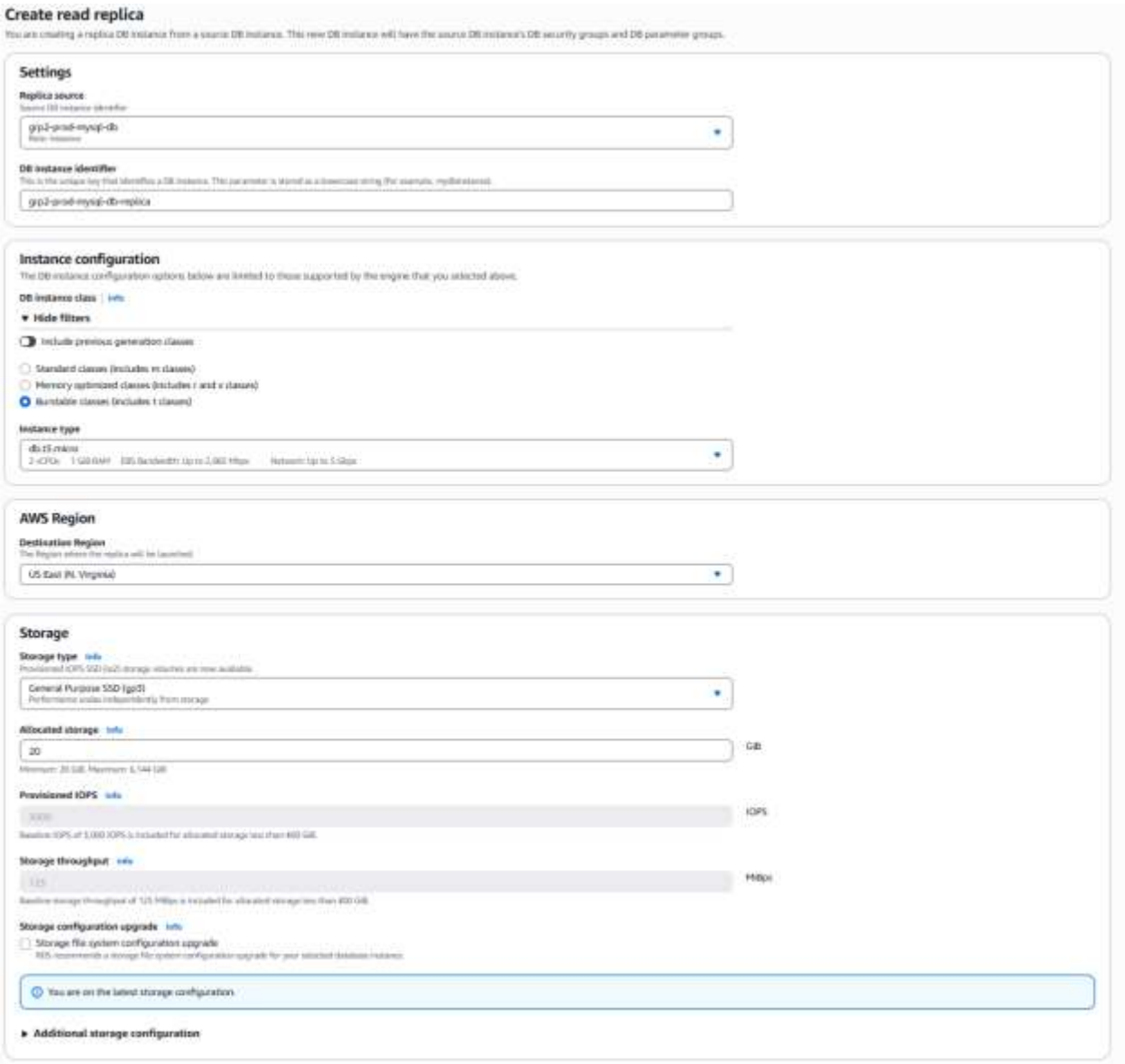

Fig. 23. Read replica configuration.1

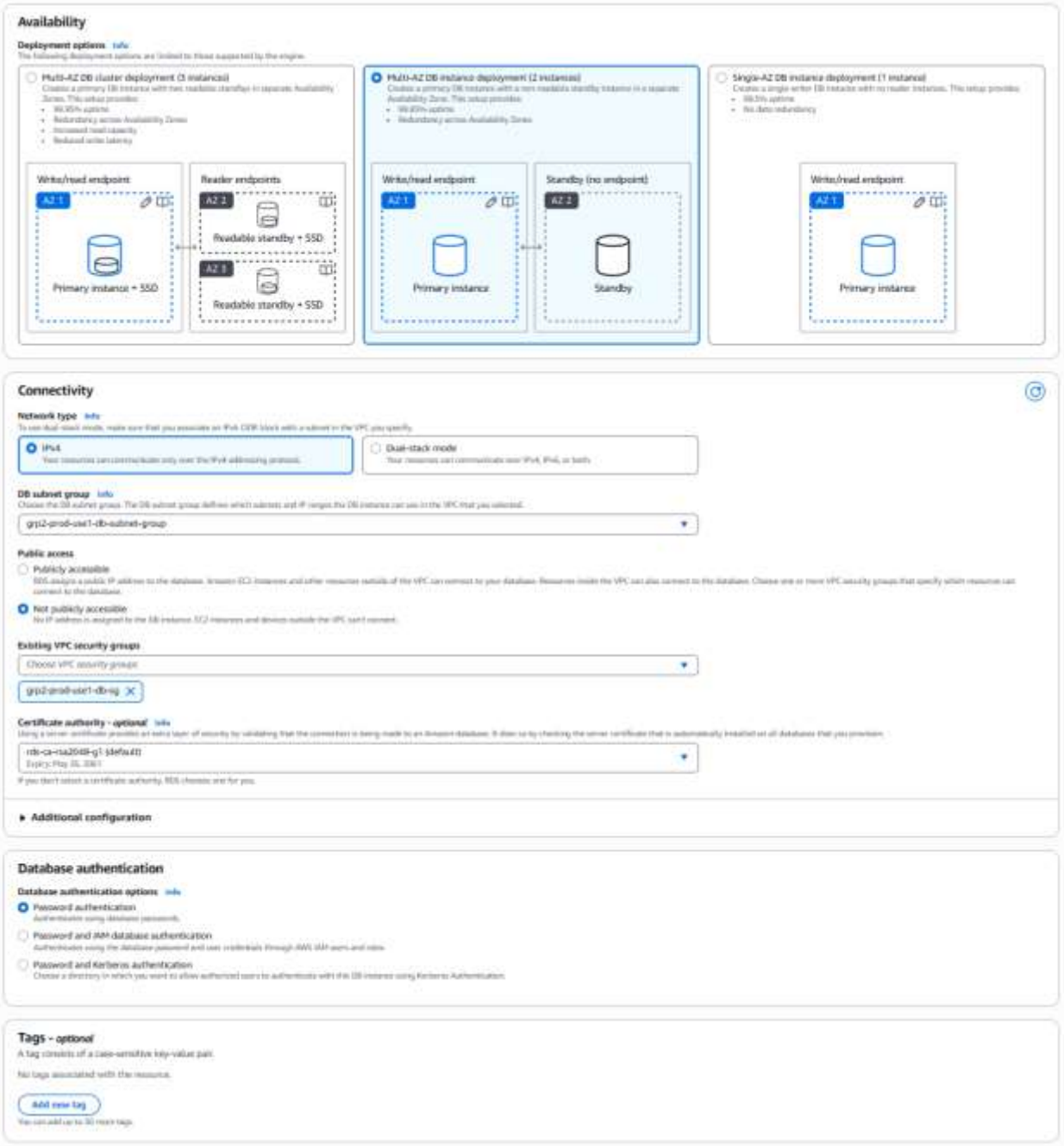

Fig. 24. Read replica configuration.2

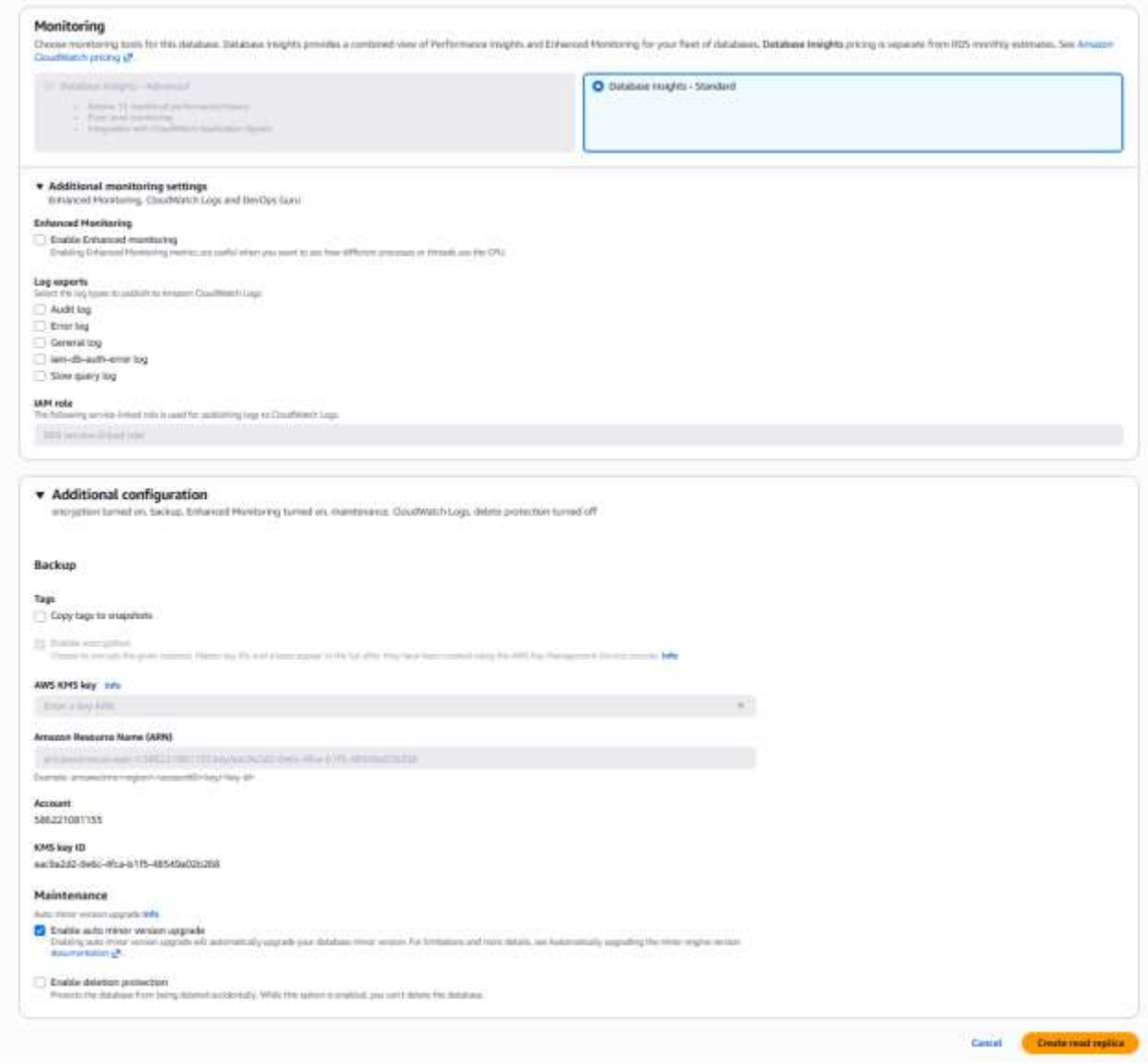

Fig. 25. Read replica configuration.3

Fig. 26. Final Databases

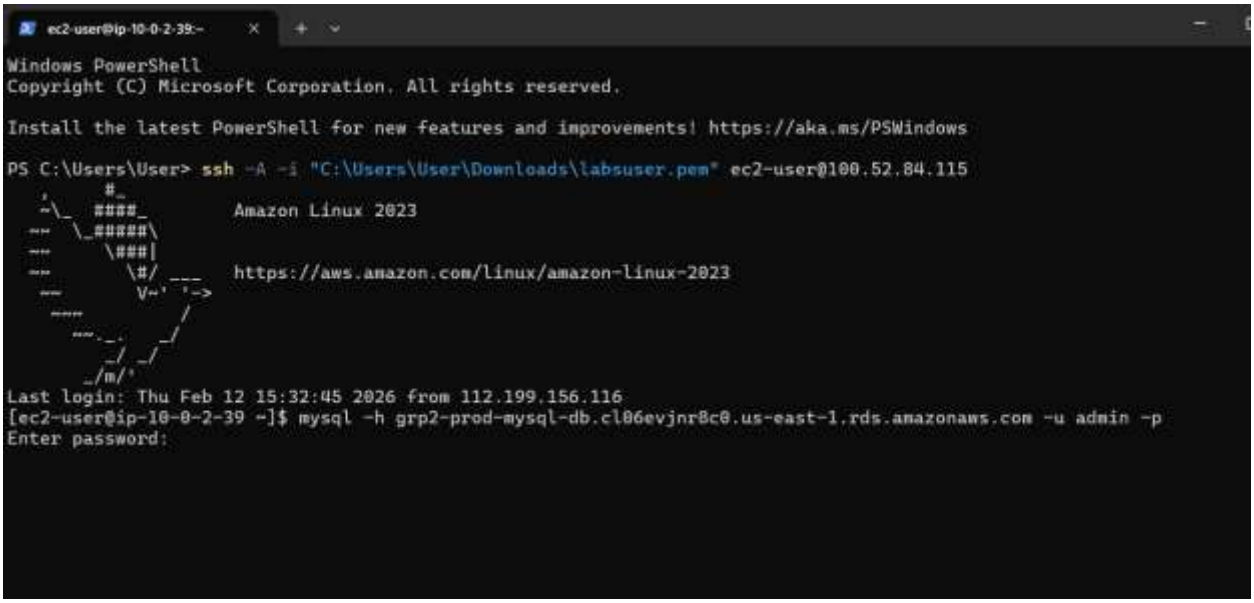

Fig. 27. Bastion into db

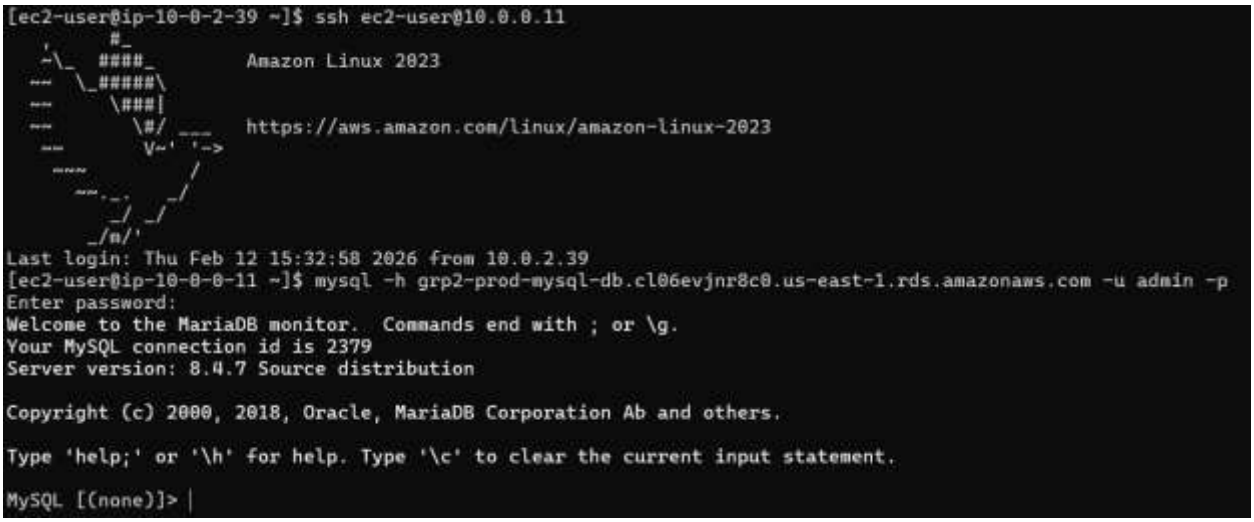

Fig. 28. Webapp into db

```
MySQL [(none)]> USE Proj;
Database changed
MySQL [Proj]> CREATE TABLE products (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(255),
    ->     description TEXT,
    ->     price DECIMAL(10, 2),
    ->     stock_quantity INT
    -> );
Query OK, 0 rows affected (0.101 sec)

MySQL [Proj]>
```

Fig. 29. Add table in db

```
MySQL [Proj]> INSERT INTO products(name,description,price,stock_quantity) VALUES ('HP','HP desktop',1200,50);
ucts(name,description,price,stock_quantity) VALUES ('HP','HP printer',200,100);
INSERT INTO products(name,description,price,stock_quantity) VALUES ('DELL','Dell notebook',1800,15);
INSERT INTO products(name,description,price,stock_quantity) VALUES ('ASUS','ASUS notebook',1500,100);Query OK, 1 row aff
ected (0.016 sec)

MySQL [Proj]> INSERT INTO products(name,description,price,stock_quantity) VALUES ('HP','HP printer',200,100);
Query OK, 1 row affected (0.014 sec)

MySQL [Proj]> INSERT INTO products(name,description,price,stock_quantity) VALUES ('DELL','Dell notebook',1800,15);
Query OK, 1 row affected (0.010 sec)

MySQL [Proj]> INSERT INTO products(name,description,price,stock_quantity) VALUES ('ASUS','ASUS notebook',1500,100);
Query OK, 1 row affected (0.031 sec)

MySQL [Proj]> SELECT * FROM products;
+----+------+---------------+---------+----------------+
| id | name | description   | price   | stock_quantity |
+----+------+---------------+---------+----------------+
|  1 | HP   | HP desktop    | 1200.00 |             50 |
|  2 | HP   | HP printer    |  200.00 |            100 |
|  3 | DELL | Dell notebook | 1800.00 |             15 |
|  4 | ASUS | ASUS notebook | 1500.00 |            100 |
+----+------+---------------+---------+----------------+
4 rows in set (0.002 sec)
```

Fig. 30. Insert record in table

4.3 Bastion Host - [TEAM MEMBER 4]

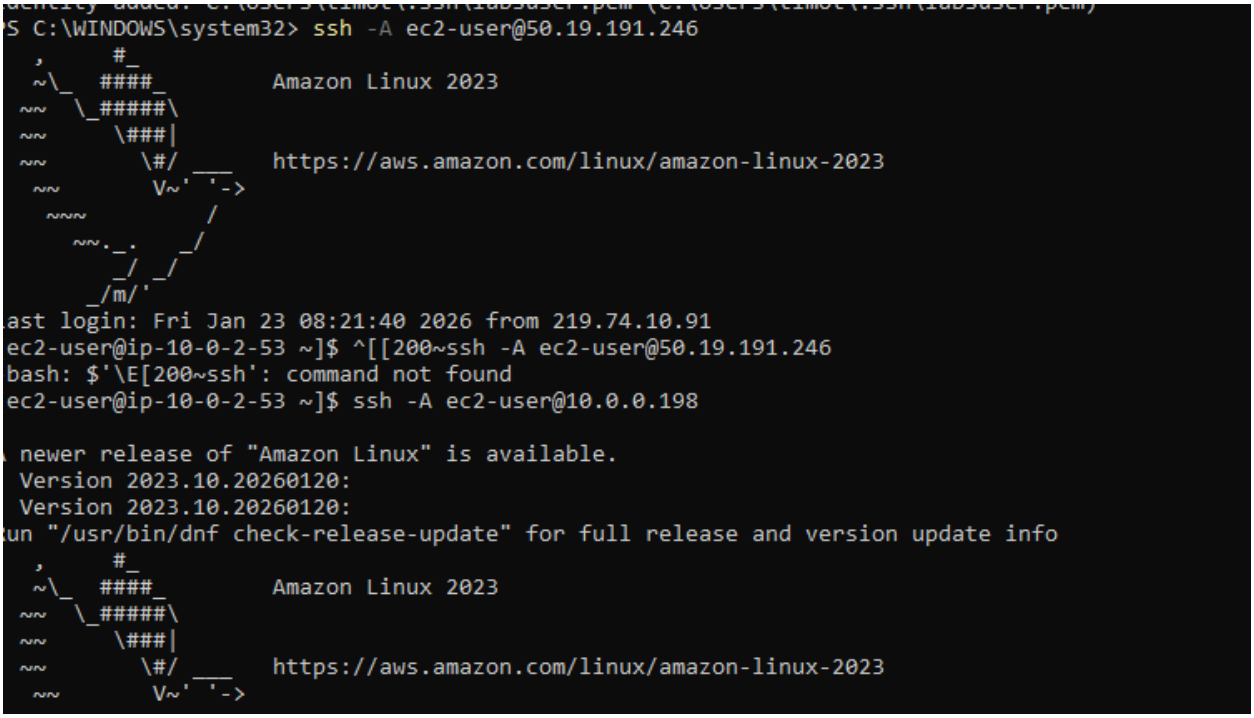

Fig. 31. SSH into bastion and through bastion to private application

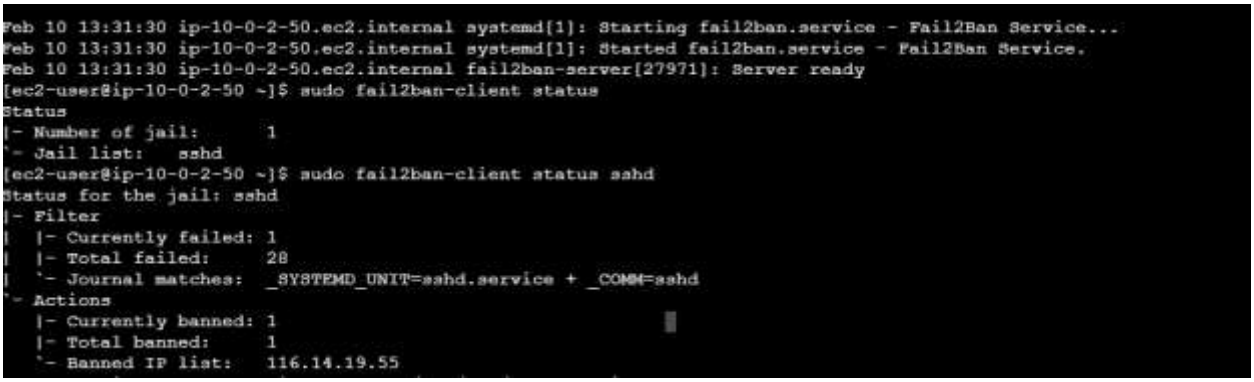

Fig. 32. for the IP ban example

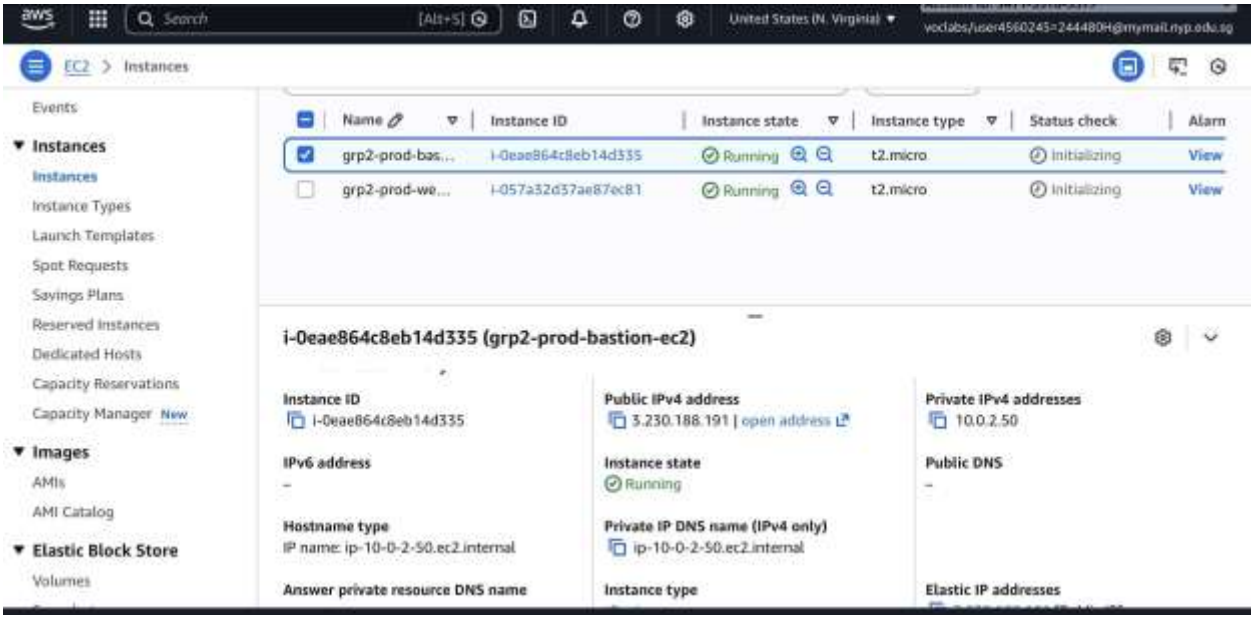

Fig. 33. Elastic IP

4.4 EBS Snapshot - [TEAM MEMBER 2]

Fig. 34. Assign tag to EC2 Volume

Fig. 35. Automated snapshot creation result

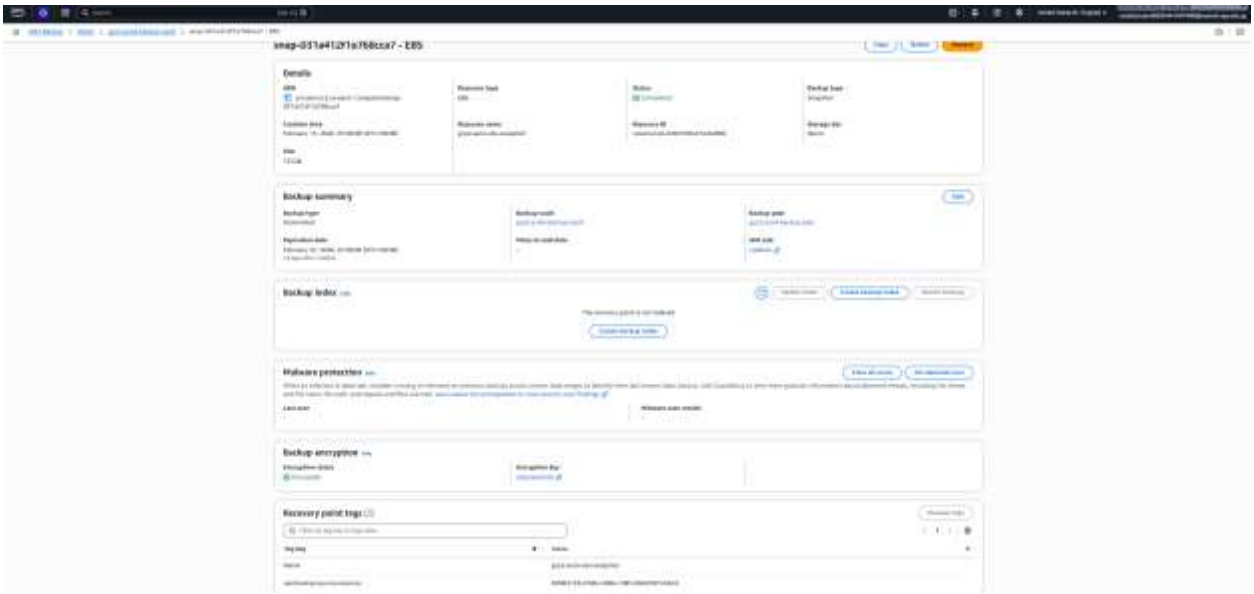

Fig. 36. Details of automated snapshot in grp2-prod-backup-vault

Fig. 37. Automated snapshot can be found under EBS snapshots

Fig. 38. Automated testing result

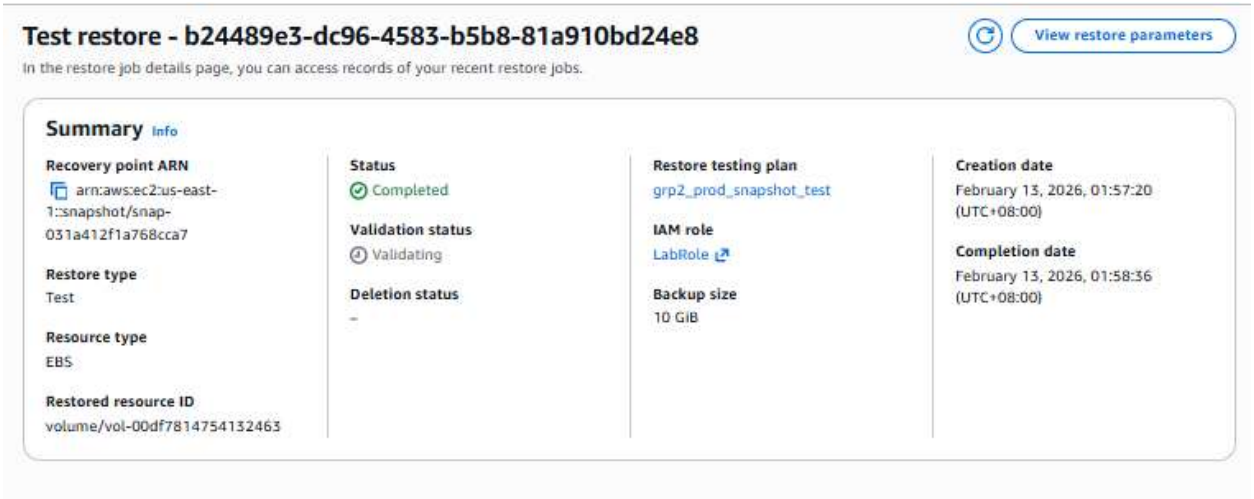

Fig. 39. Details of testing result

4.5 S3 & Webapp – ERI WANG

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEC2Get",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": [
          "arn:aws:s3:::grp2-prod-webapp-webfiles-s3",
          "arn:aws:s3:::grp2-prod-webapp-webfiles-s3/*"
          ]
    }
  ]
}
```

Fig. 40. Bucket Policy for grp2-prod-webapp-webfiles-s3

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPublicAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
          "arn:aws:s3:::grp2-prod-webapp-uploads-s3",
          "arn:aws:s3:::grp2-prod-webapp-uploads-s3/*"
          ]
    }
  ]
}
```

Fig. 41. Bucket Policy for grp2-prod-webapp-uploads-s3

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "GET",
            "POST",
            "PUT"
        ],
        "AllowedOrigins": [
            "https://egl211.rirori.xyz"
        ],
        "ExposeHeaders": [
            "ETag"
        ],
        "MaxAgeSeconds": 3000
    }
]
```

Fig. 42. CORS policy to enable cross-origin browser uploads from the Load Balancer to S3

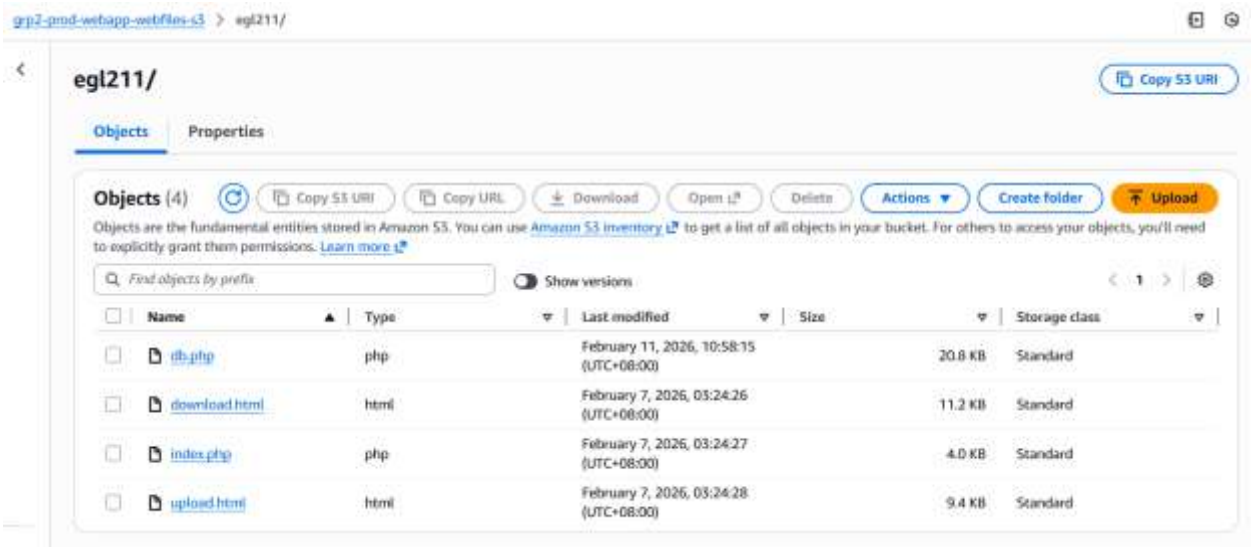

Fig. 43. Content of grp2-prod-webapp-webfiles-s3

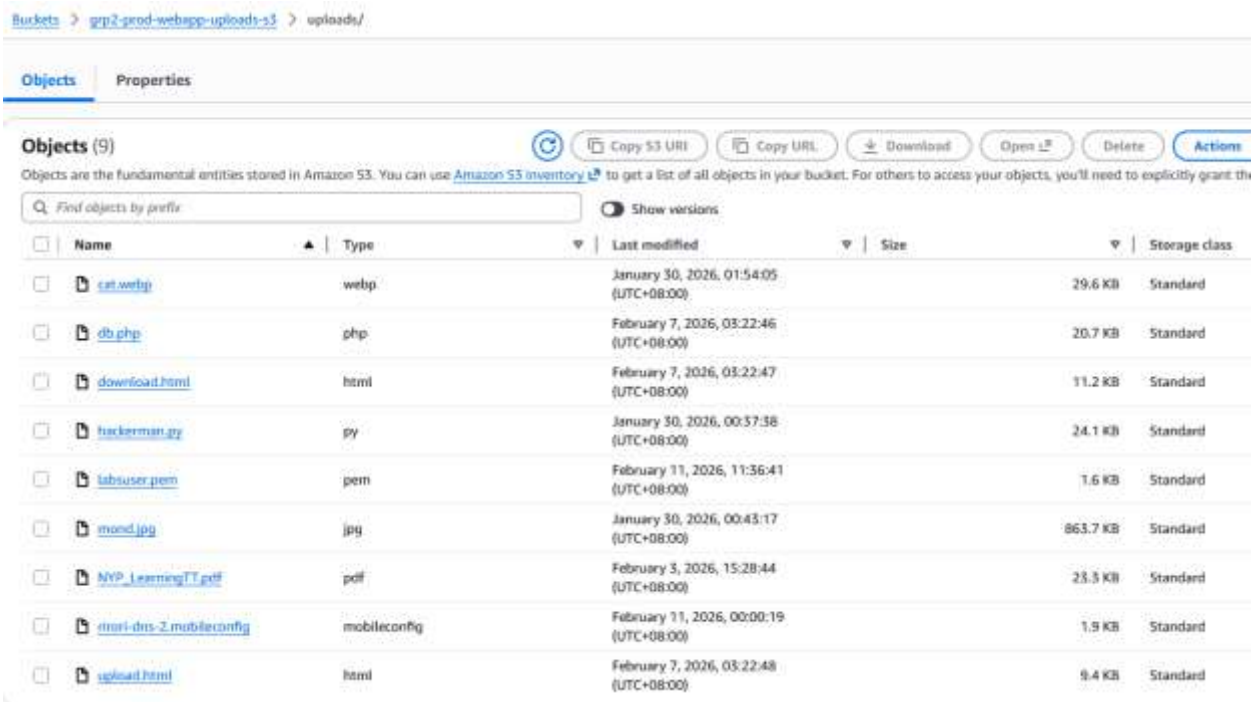

Fig. 44. Content of grp2-prod-webapp-uploads-s3

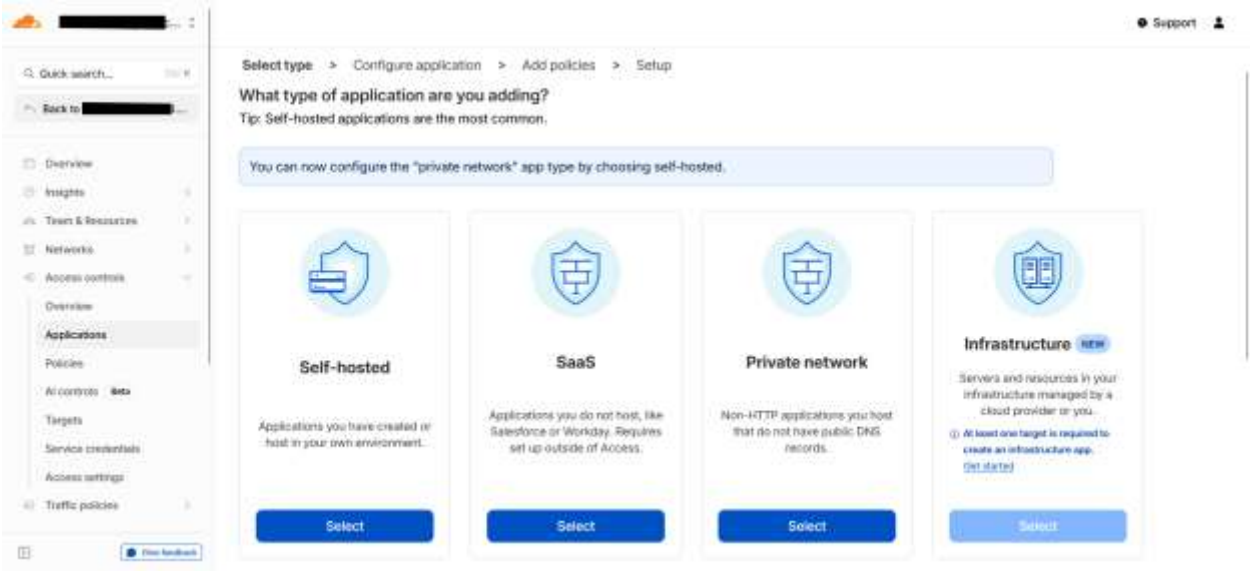

Fig. 45. Cloudflare Zero Trust Add Application page

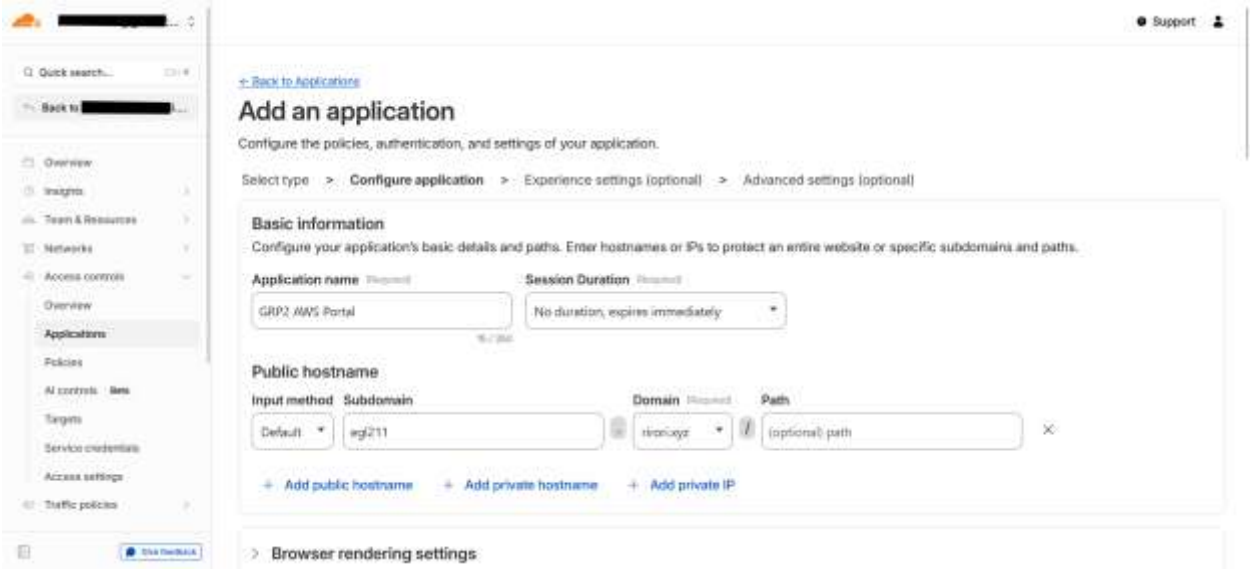

Fig. 46. Cloudflare Application configurations

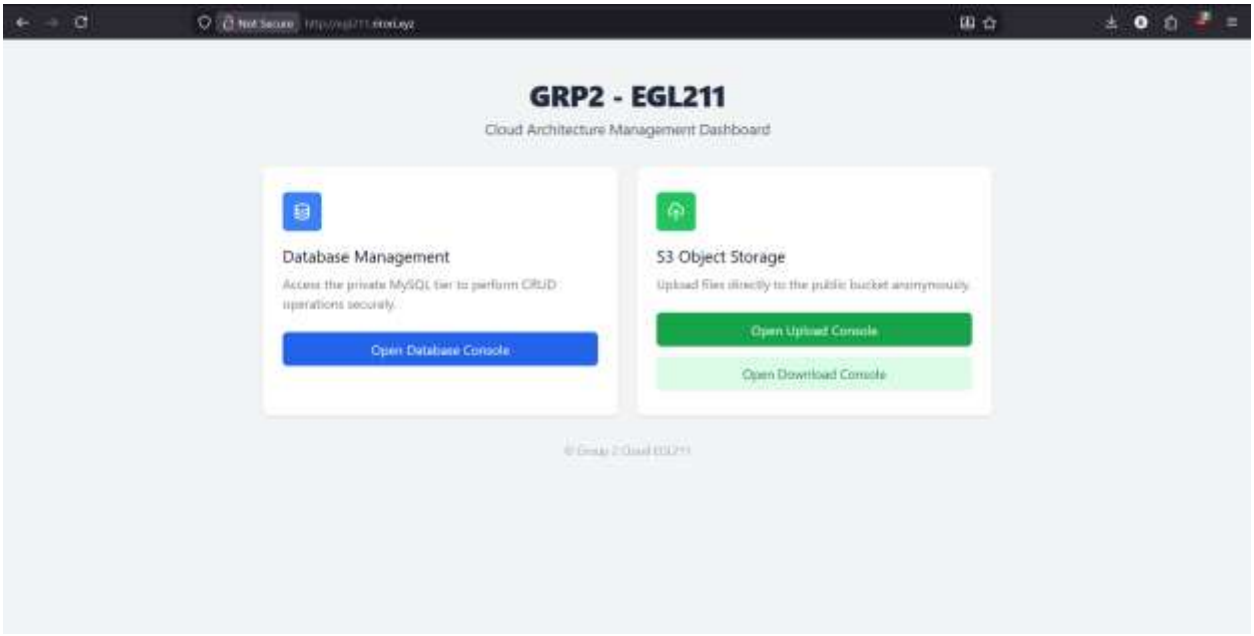

Fig. 47. Accessing webpage through custom domain

Fig. 48. Cloudflare domain dashboard

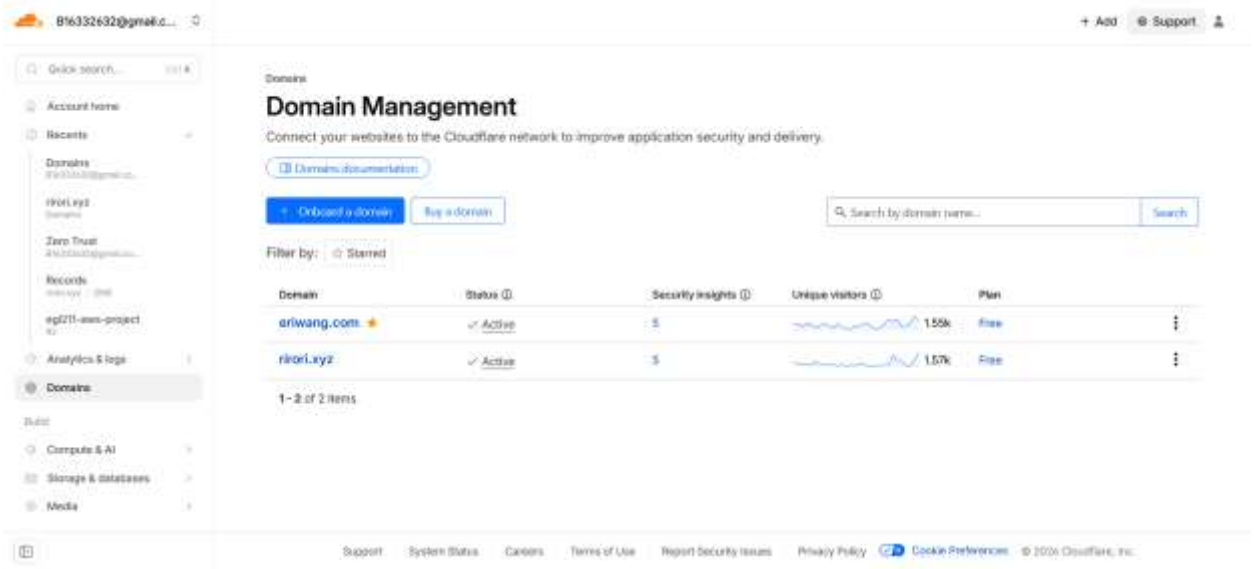

Fig. 49. Cloudflare Dashboard

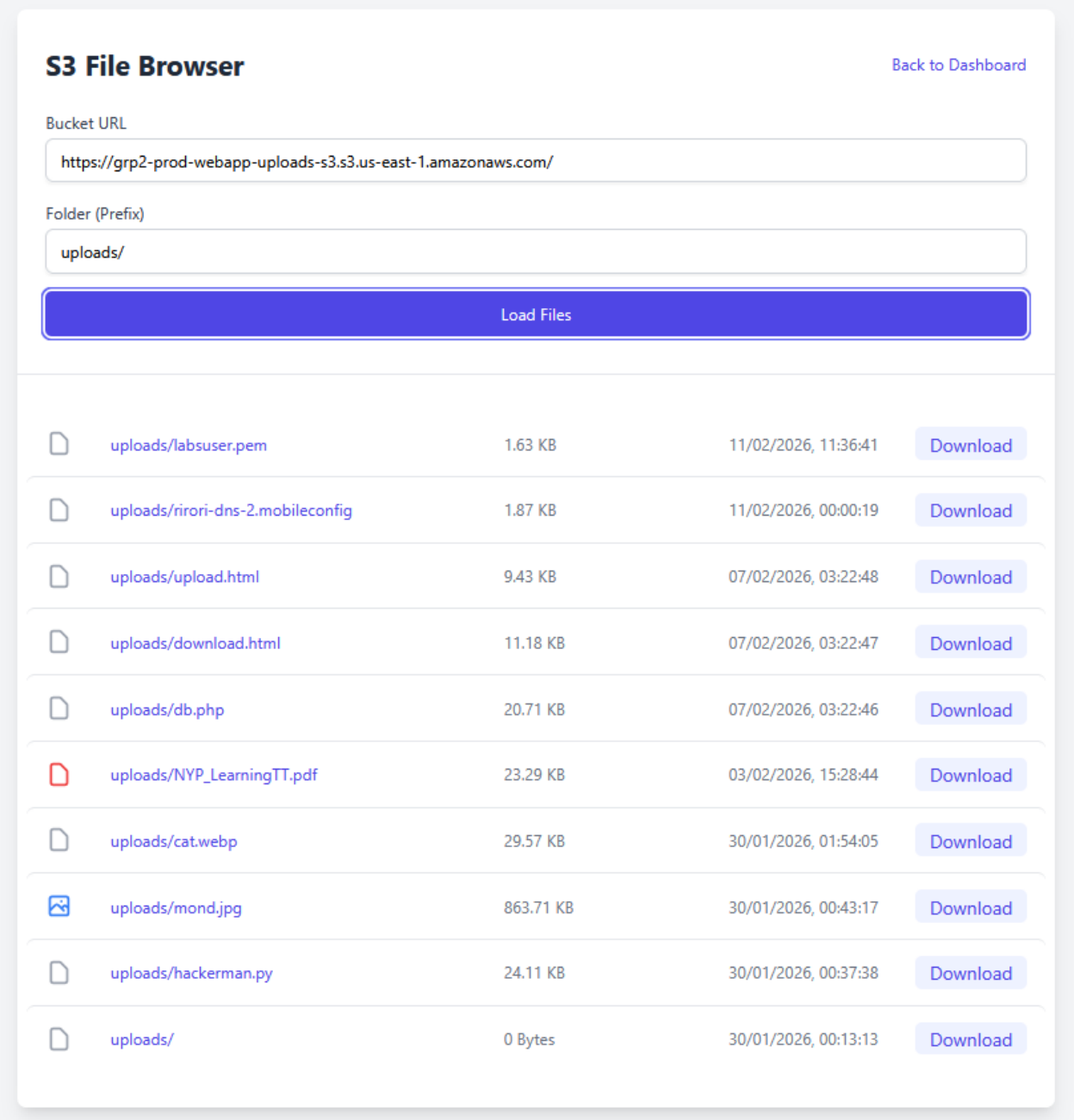

Fig. 50. Webapp loading content from grp2-prod-webapp-uploads-s3

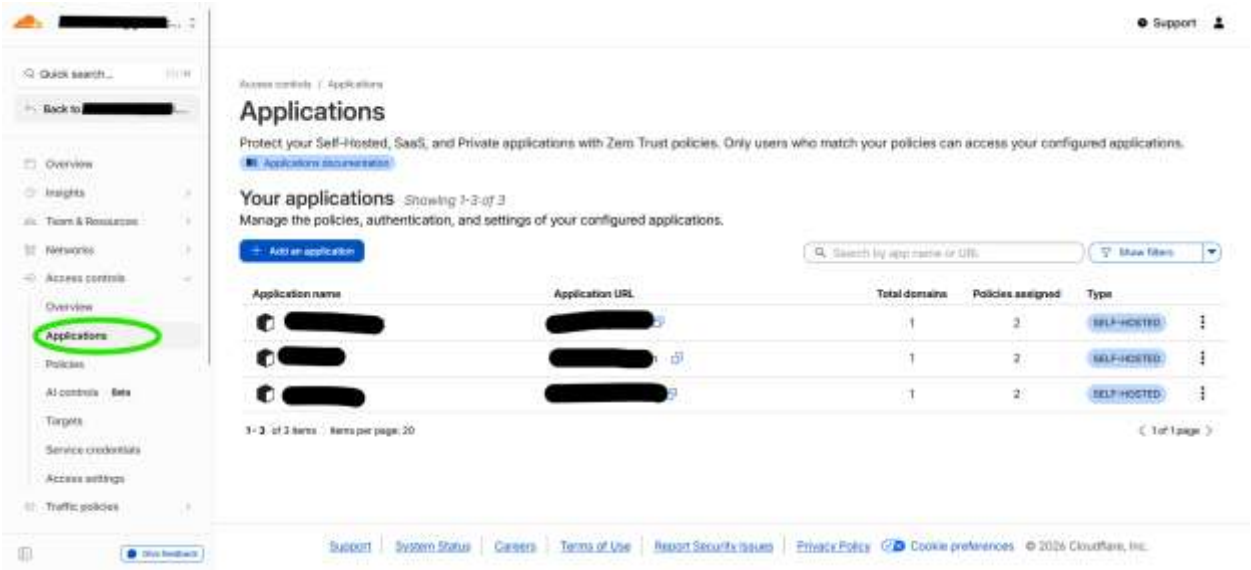

Fig. 51. Cloudflare Zero Trust Applications page

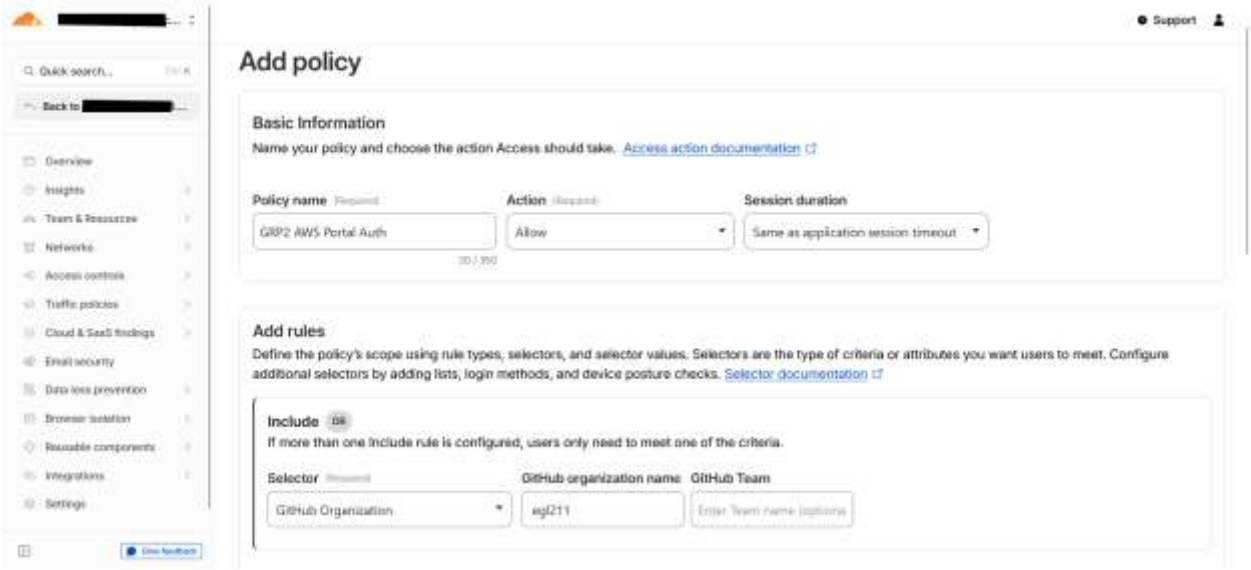

Fig. 52. Cloudflare Policy configuration