Kimai Infrastructure Migration

Date : 11-07-2025

Github Repository

https://github.com/subadevanc/Kimai-TechForce/

Presented by

Sophiya S
23347
B.E. CSE (Cyber Security)

Phase 1

Design and

Planning

Task 1.1 – High-Level Design (HLD)

Project Overview

- **The project migrates the Kimai timesheet app to a secure, multi-cloud setup.**
- **It replaces manual hosting with automated, containerized deployment.**
- **Docker + Apache are used to run the app in isolated containers.**
- **Jenkins CI/CD pipeline automates code build, test, and deployment.**
- **AWS hosts the core infrastructure: app, Jenkins, Bastion, and database.**

- **GCP handles monitoring and metrics via Grafana + Prometheus.**
- **Terraform provisions all cloud infrastructure as code.**
- **Security is enforced using IAM roles and firewall rules.**
- **Monitoring provides real-time health checks and usage stats.**
- **The new setup ensures better scalability, performance, and automation.**

Problem Statement

- On-prem Kimai lacks auto-scaling and redundancy.
- No CI/CD leads to manual deployments.
- Poor visibility into system health.
- No centralized monitoring or alerts.
- Risk of data loss or breach in local setup.
- High maintenance and operational cost.

No version-controlled infrastructure.

Problem Objectives

- Deploy Kimai in a cloud-native Docker container.
- Set up automated CI/CD pipeline using Jenkins.
- Provision infrastructure using Terraform.
- Enable real-time monitoring and logging.
- Implement secure, scalable environment.
- Use only free-tier or cost-effective resources.

Ensure backups and security for data.

Cloud Storage

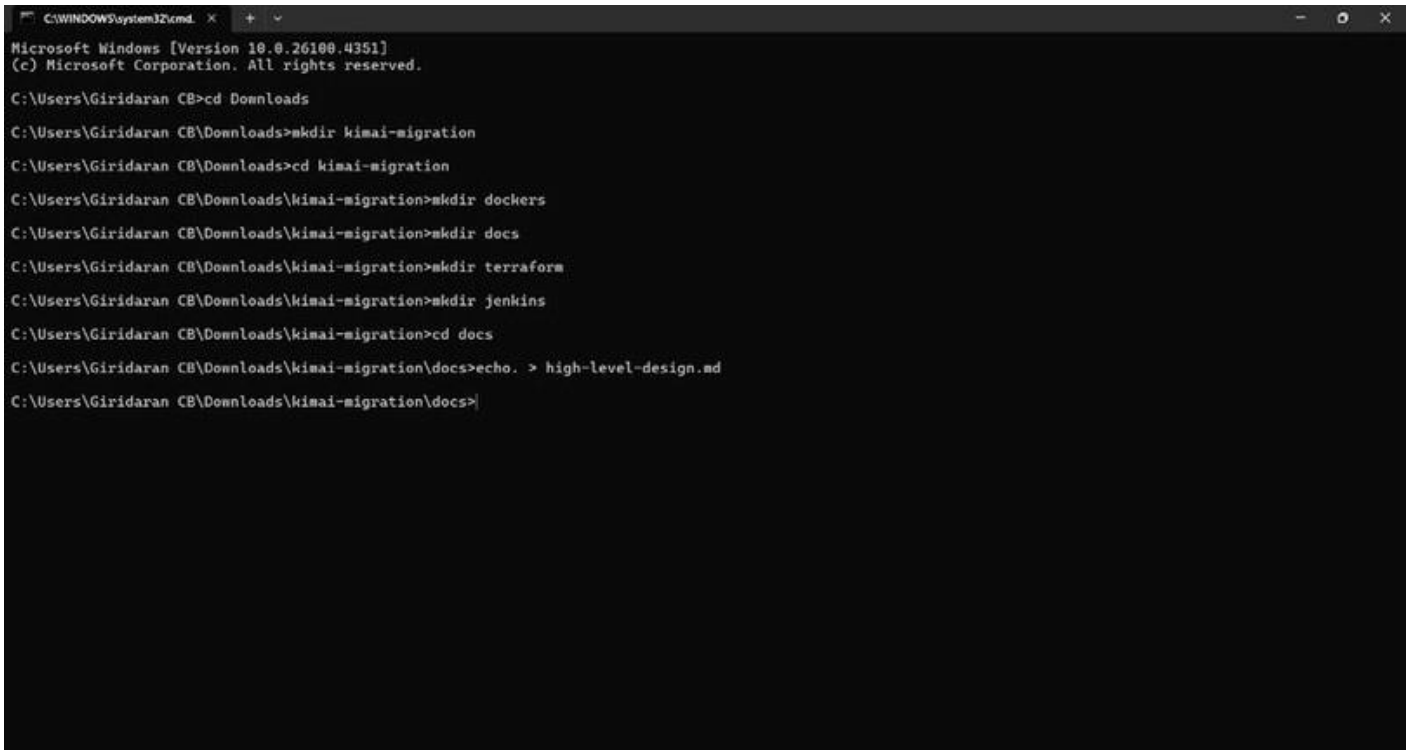
- Deploy Kimai in a cloud-native Docker container.
- Set up automated CI/CD pipeline using Jenkins.
- Provision infrastructure using Terraform.
- Enable real-time monitoring and logging.
- Implement secure, scalable environment.
- Use only free-tier or cost-effective resources.
- Ensure backups and security for data.

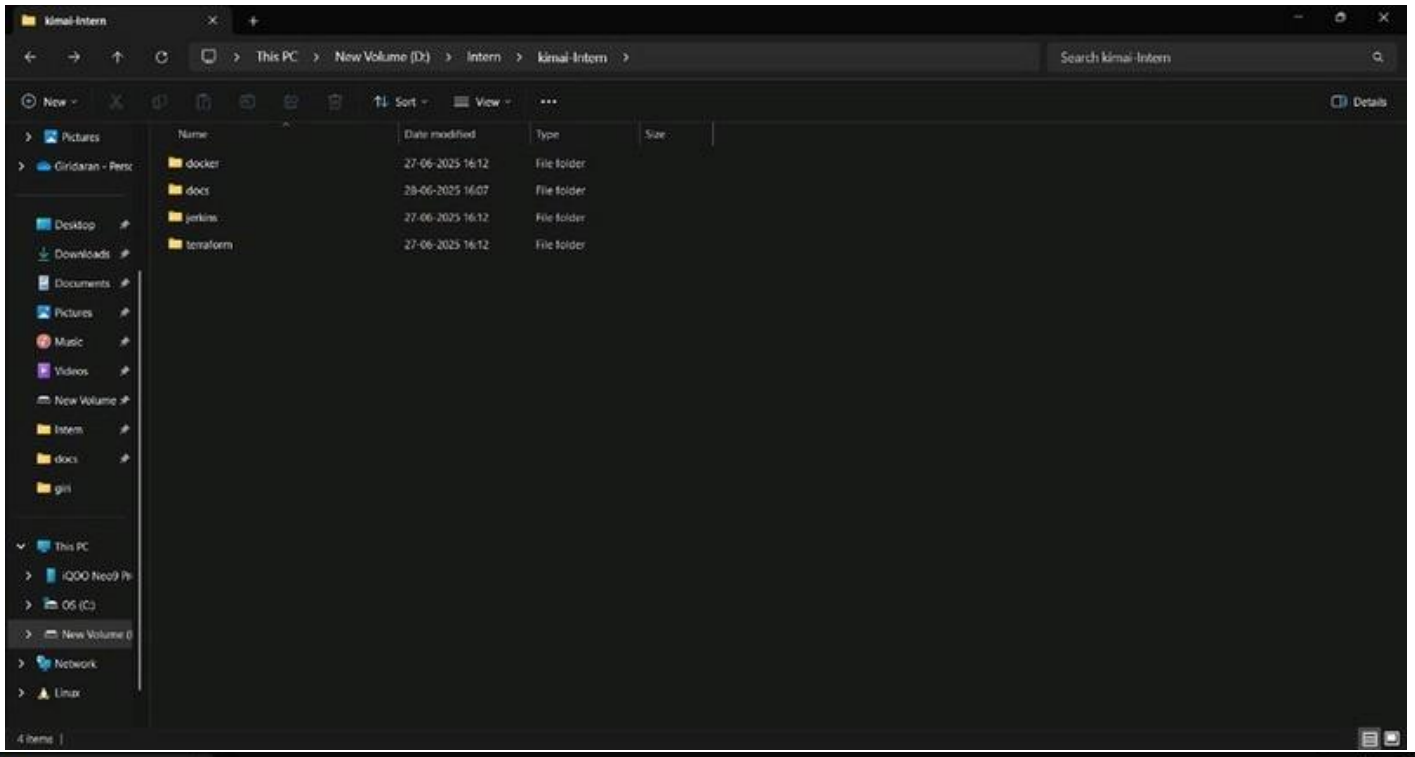
Technology Stack

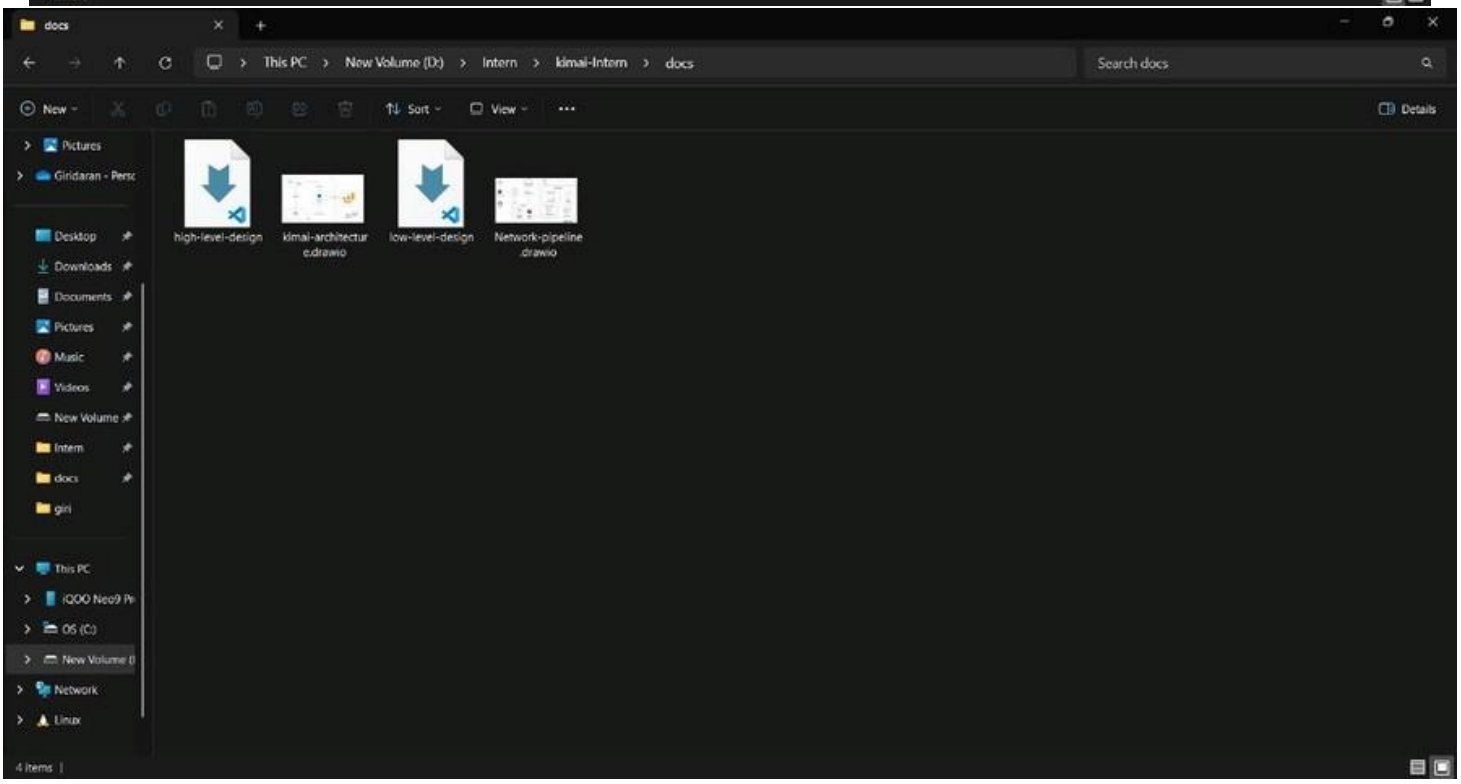
- Docker for app containerization.
- Apache + PHP for Kimai runtime.
- Jenkins for continuous integration/deployment.
- Terraform for infrastructure provisioning.
- AWS EC2, RDS, IAM, SGs for core hosting.

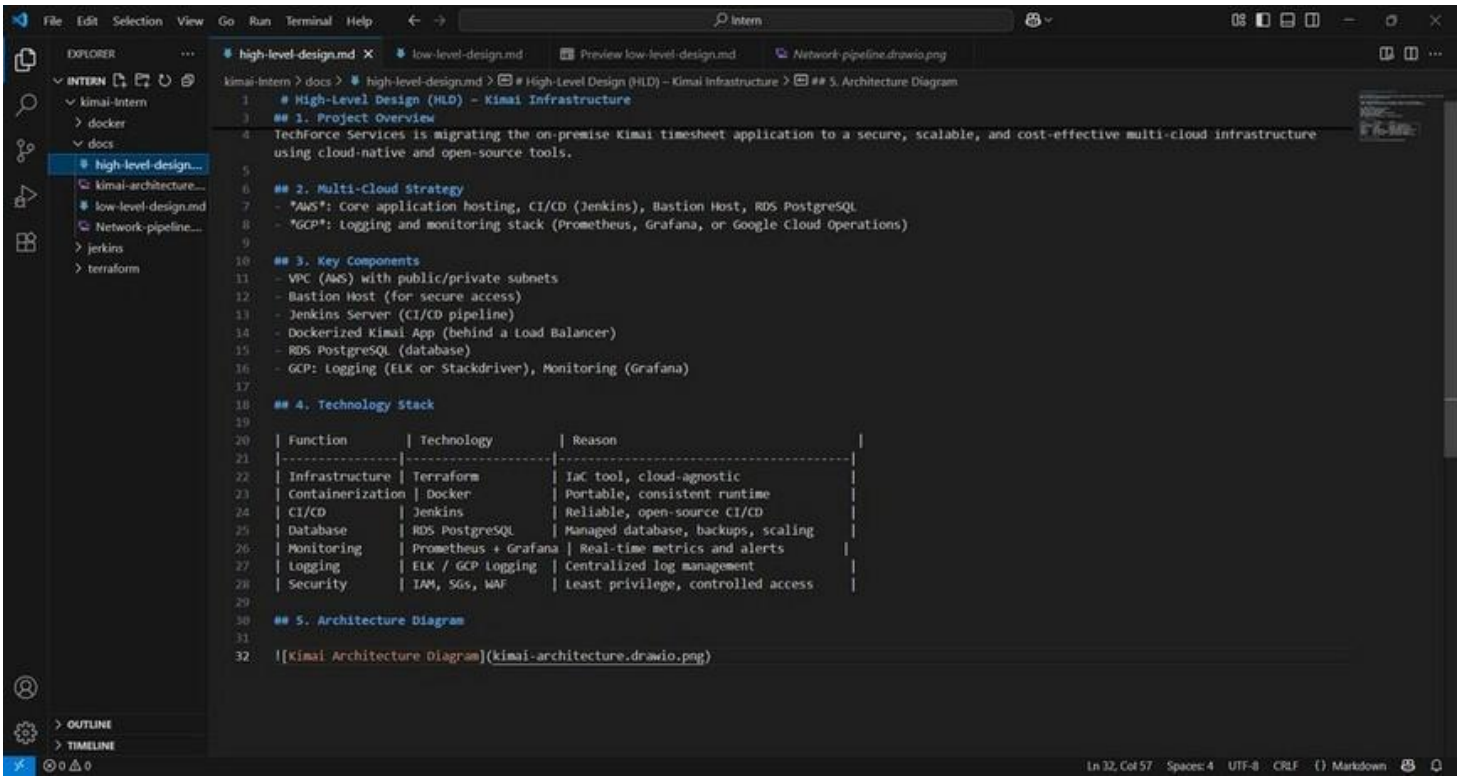
GCP Compute Engine for monitoring.

PostgreSQL for backend database.


C:\WINDOWS\system32\cmd.
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Giridaran CB>cd Downloads
C:\Users\Giridaran CB\Downloads>mkdir kimai-migration
C:\Users\Giridaran CB\Downloads>cd kimai-migration
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir dockers
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir docs
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir terraform
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir jenkins
C:\Users\Giridaran CB\Downloads\kimai-migration>cd docs
C:\Users\Giridaran CB\Downloads\kimai-migration\docs>echo. > high-level-design.md
C:\Users\Giridaran CB\Downloads\kimai-migration\docs>

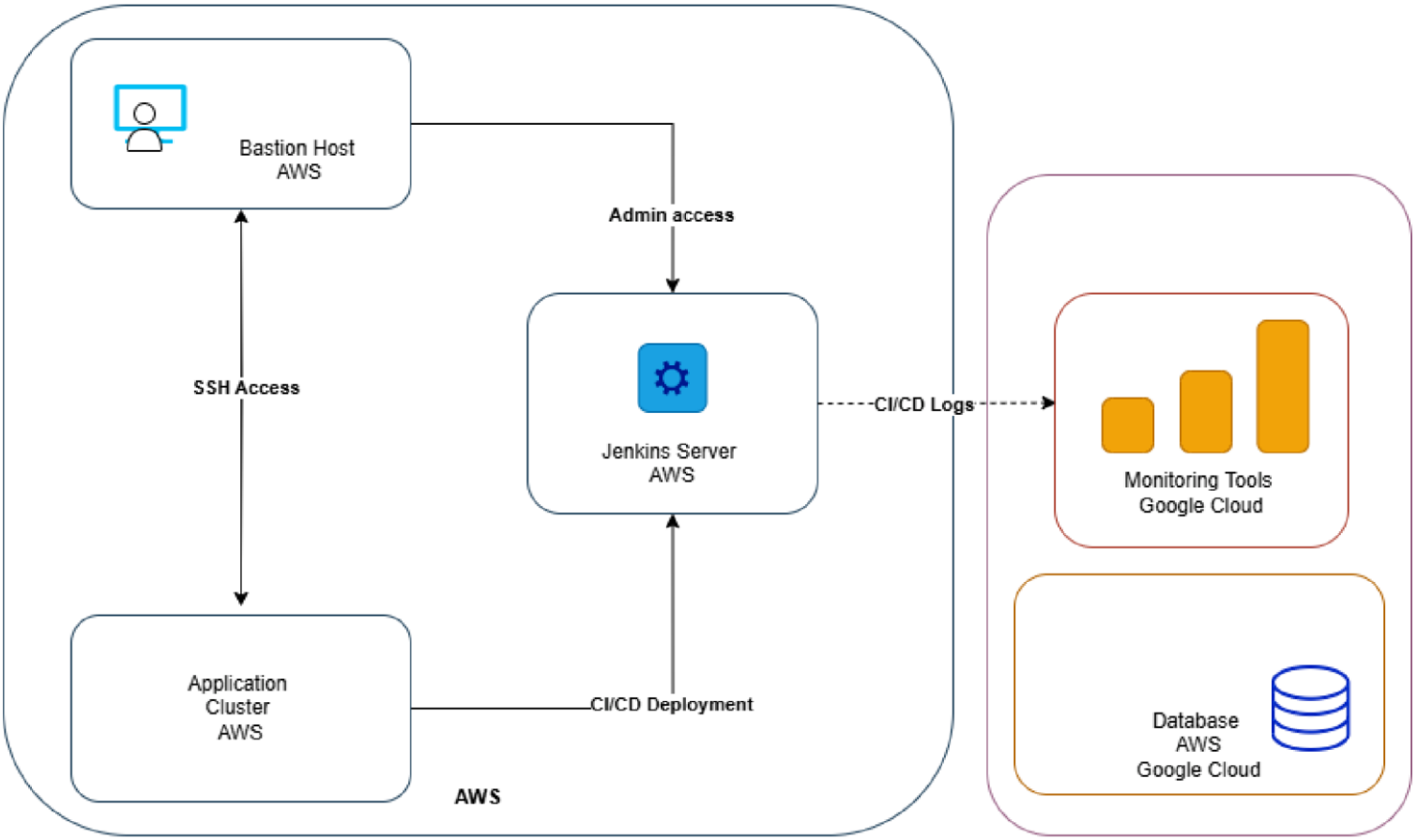

kimai-Intern
This PC
New Volume (D:)
Intern
kimai-Intern
Search kimai-Intern
New
Sort
View
Details
Name
Date modified
Type
Size
docker
27-06-2025 16:12
File folder
docs
28-06-2025 16:07
File folder
jenkins
27-06-2025 16:12
File folder
terraform
27-06-2025 16:12
File folder
Pictures
Giridaran - Persc
Desktop
Downloads
Documents
Pictures
Music
Videos
New Volume
Intern
docs
giri
This PC
iQOO Neo9 Pr
OS (C:)
New Volume (
Network
Linux
4 items


docs
This PC
New Volume (D:)
Intern
kimai-Intern
docs
Search docs
New
Sort
View
Details
high-level-design
kimai-architecture.drawio
low-level-design
Network-pipeline.drawio
Pictures
Giridaran - Persc
Desktop
Downloads
Documents
Pictures
Music
Videos
New Volume
Intern
docs
giri
This PC
iQOO Neo9 Pr
OS (C:)
New Volume (
Network
Linux
4 items


File Edit Selection View Go Run Terminal Help
Intern
EXPLORER
high-level-design.md
low-level-design.md
Preview low-level-design.md
Network-pipeline.drawio.png
kimai-intern > docs > high-level-design.md > # High-Level Design (HLD) – Kimai Infrastructure > ## 5. Architecture Diagram
INTERN
kimai-intern
docker
docs
high-level-design....
kimai-architecture....
low-level-design.md
Network-pipeline....
jenkins
terraform
OUTLINE
TIMELINE
High-Level Design (HLD) – Kimai Infrastructure
1. Project Overview
TechForce Services is migrating the on-premise Kimai timesheet application to a secure, scalable, and cost-effective multi-cloud infrastructure using cloud-native and open-source tools.
2. Multi-Cloud Strategy
- *AWS*: Core application hosting, CI/CD (Jenkins), Bastion Host, RDS PostgreSQL
- *GCP*: Logging and monitoring stack (Prometheus, Grafana, or Google Cloud Operations)
3. Key Components
- VPC (AWS) with public/private subnets
- Bastion Host (for secure access)
- Jenkins Server (CI/CD pipeline)
- Dockerized Kimai App (behind a Load Balancer)
- RDS PostgreSQL (database)
- GCP: Logging (ELK or Stackdriver), Monitoring (Grafana)
4. Technology Stack
Function	Technology	Reason
Infrastructure	Terraform	IaC tool, cloud-agnostic
Containerization	Docker	Portable, consistent runtime
CI/CD	Jenkins	Reliable, open-source CI/CD
Database	RDS PostgreSQL	Managed database, backups, scaling
Monitoring	Prometheus + Grafana	Real-time metrics and alerts
Logging	ELK / GCP Logging	Centralized log management
Security	IAM, SGs, WAF	Least privilege, controlled access
5. Architecture Diagram
![Kimai Architecture Diagram](kimai-architecture.drawio.png)
Ln 32, Col 57 Spaces: 4 UTF-8 CRLF Markdown

Infrastructure Diagram

High-Level Design



Phase - 1

Task 1.2

Low-Level Design (LLD)

Network Design – AWS

- **VPC CIDR: 10.0.0.0/16**
- **Public Subnet: Jenkins, Bastion**
- **Private Subnet: Kimai App, RDS**
- **Internet Gateway for public subnet**
- **NAT Gateway for private instances**
- **Custom Route Tables per subnet**
- **Security Groups for access contro**

Network Design – GCP

- **GCP VPC CIDR: 10.1.0.0/16**
- **Private subnet for monitoring stack**
- **Grafana and Prometheus installed**
- **Firewall allows metrics from AWS app subnet**
- **Logs and metrics are pulled securely**

- **No public access to monitoring instance**

Compute Design

- **EC2 t2.micro for App, Jenkins, Bastion.**
- **GCP e2-micro for Monitoring Node.**
- **Free-tier eligible instances to reducecost.**
- **Auto Scaling Group for App Server.**
- **Min: 1, Max: 3 instances.**
- **Trigger: CPU > 70% for 5 minutes.**

Storage Design

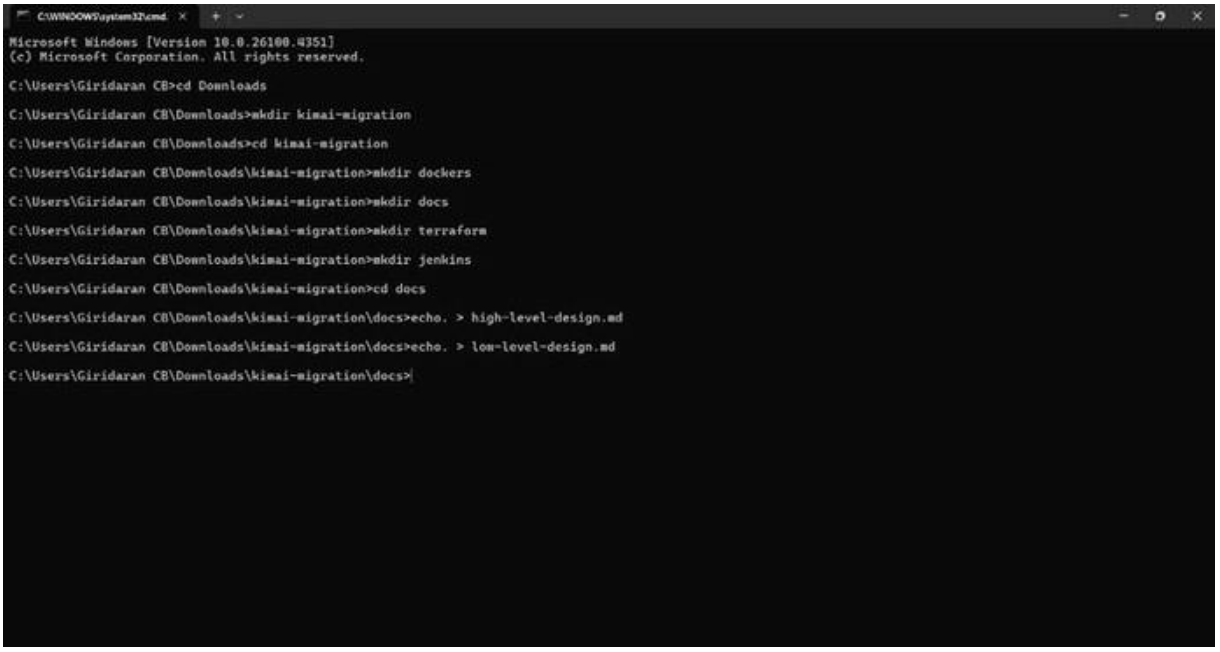
- **EC2 Volumes: 20 GB gp2 (SSD).**
- **RDS PostgreSQL: db.t3.micro instance.**
- **20 GB allocated storage.**
- **Backups enabled with 7-day retention.**

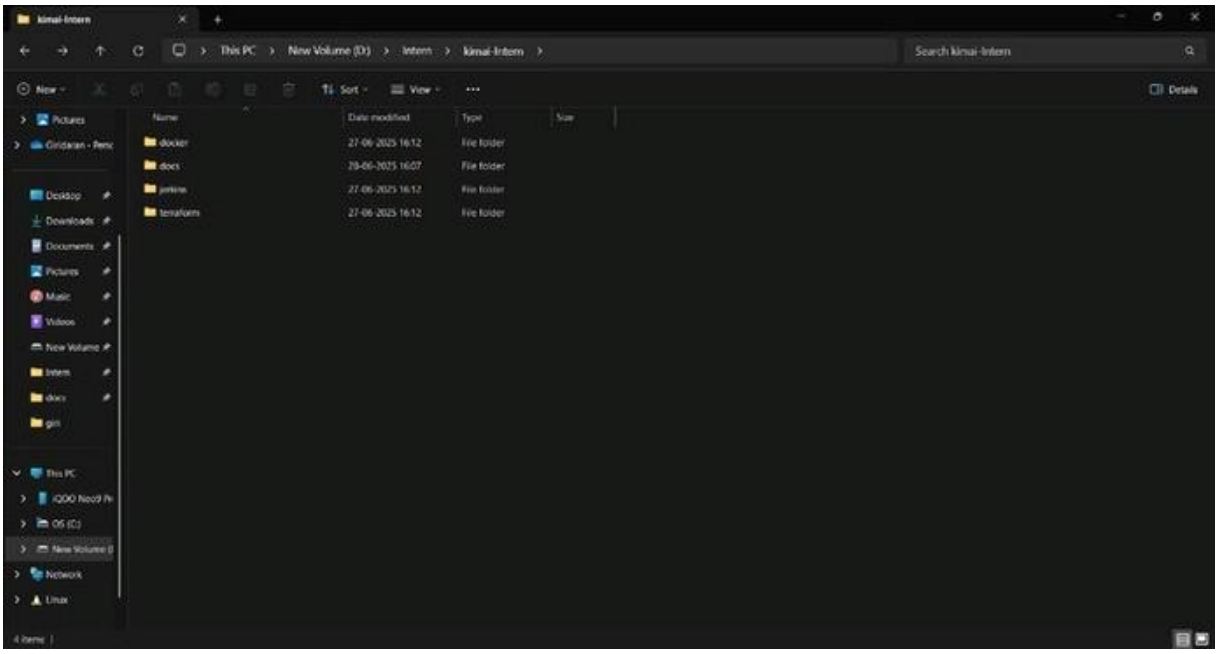
- **Encrypted at rest.**

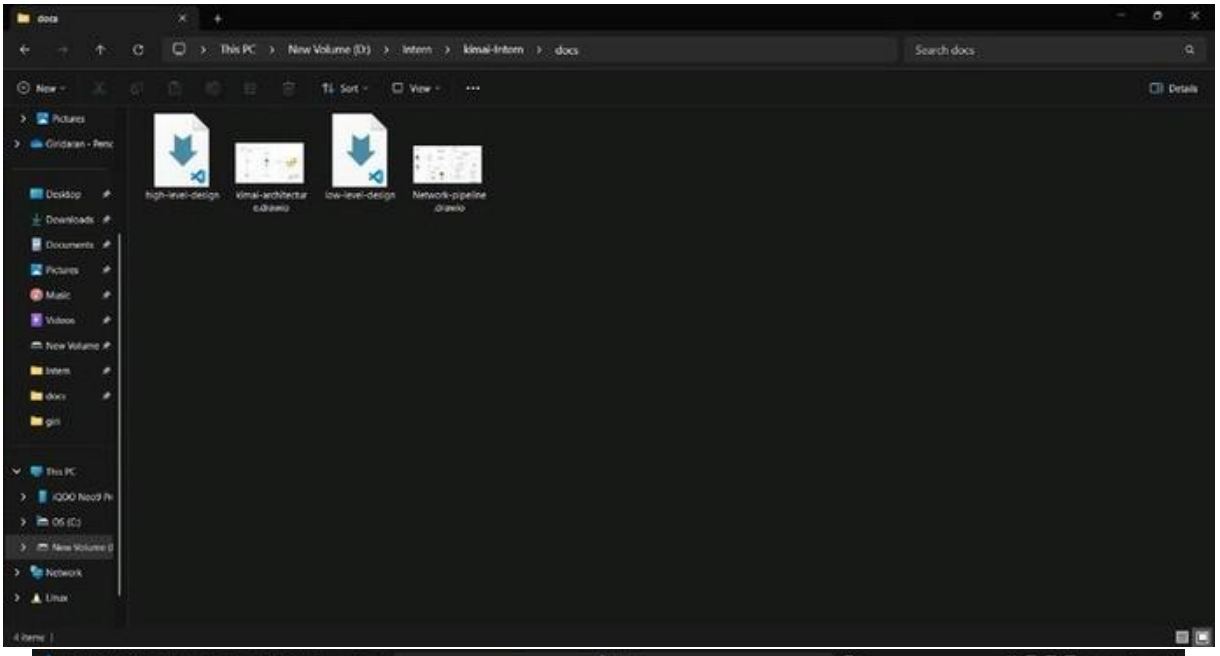
- **Database not publicly accessible.**

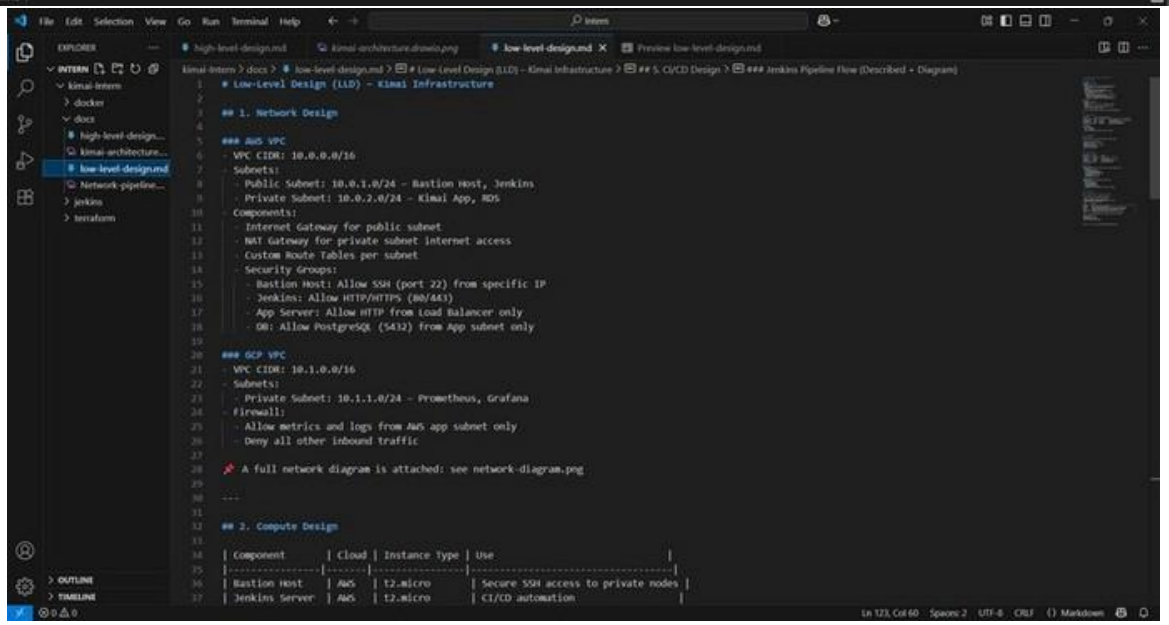
CI/CD Pipeline Design

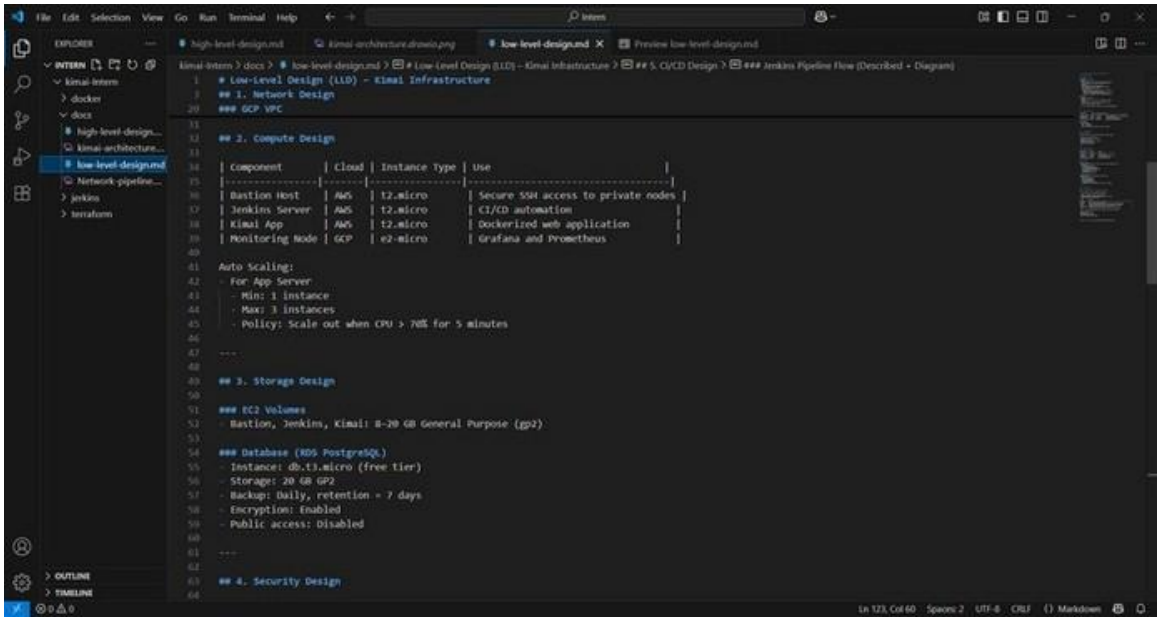
- **GitHub push triggers Jenkins job Jenkins**
- **stages:**
- **Checkout**
- **Docker Build**
- **Simulated Test**
- **Push Image**
- **Deploy to EC2**
- **Docker image hosted on DockerHub or ECR**


C:\WINDOWS\system32\cmd.
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Giridaran CB>cd Downloads
C:\Users\Giridaran CB\Downloads>mkdir kimai-migration
C:\Users\Giridaran CB\Downloads>cd kimai-migration
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir dockers
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir docs
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir terraform
C:\Users\Giridaran CB\Downloads\kimai-migration>mkdir jenkins
C:\Users\Giridaran CB\Downloads\kimai-migration>cd docs
C:\Users\Giridaran CB\Downloads\kimai-migration\docs>echo. > high-level-design.md
C:\Users\Giridaran CB\Downloads\kimai-migration\docs>echo. > low-level-design.md
C:\Users\Giridaran CB\Downloads\kimai-migration\docs>

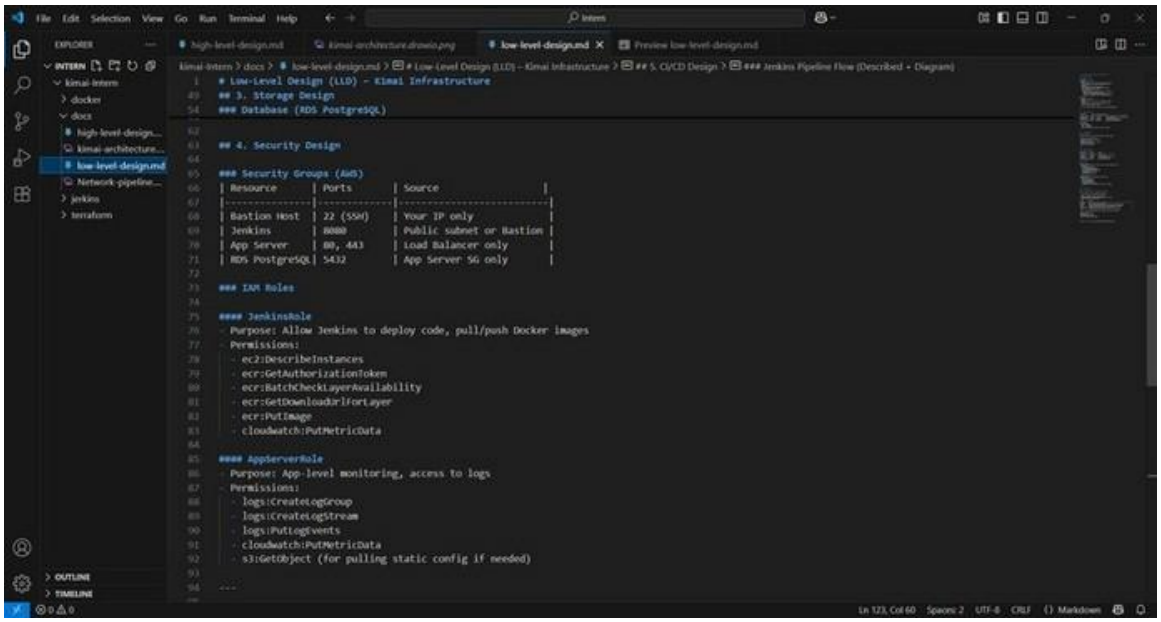

kimai-Intern
This PC
New Volume (D:)
Intern
kimai-Intern
Search kimai-Intern
New
Sort
View
Details
Name
Date modified
Type
Size
docker
27-06-2025 16:12
File folder
docs
28-06-2025 16:07
File folder
jenkins
27-06-2025 16:12
File folder
terraform
27-06-2025 16:12
File folder
Pictures
Desktop
Downloads
Documents
Pictures
Music
Videos
New Volume
Intern
docs
giri
This PC
OS (C:)
Network
Linux
4 items


docs
This PC
New Volume (D:)
intern
kimai-intern
docs
Search docs
New
Sort
View
Details
high-level-design
kimai-architectur e.drawio
low-level-design
Network-pipeline .drawio
4 items

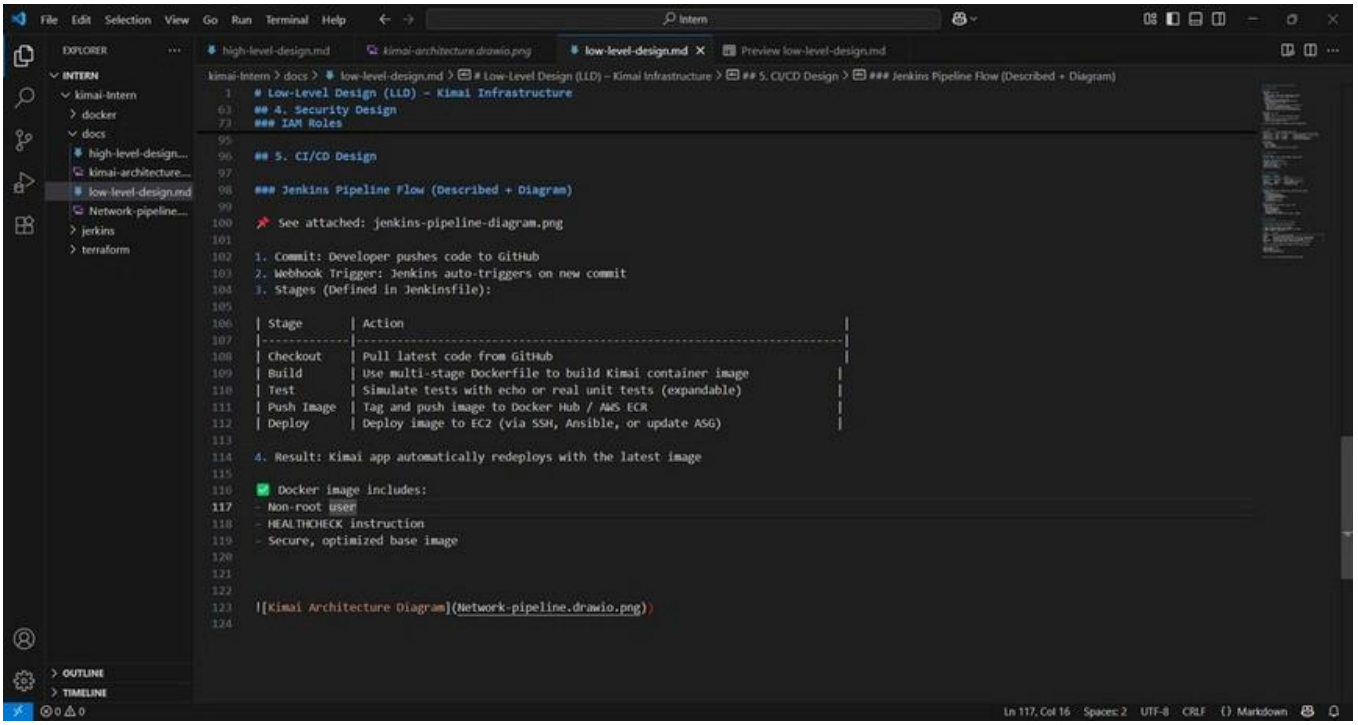

File Edit Selection View Go Run Terminal Help
intern
EXPLORER
high-level-design.md
kimai-architecture.drawio.png
low-level-design.md
Preview low-level-design.md
kimai-intern > docs > low-level-design.md > # Low-Level Design (LLD) - Kimai Infrastructure > ## 5. CI/CD Design > ### Jenkins Pipeline Flow (Described + Diagram)
Low-Level Design (LLD) - Kimai Infrastructure
1. Network Design
AWS VPC
- VPC CIDR: 10.0.0.0/16
- Subnets:
- Public Subnet: 10.0.1.0/24 - Bastion Host, Jenkins
- Private Subnet: 10.0.2.0/24 - Kimai App, RDS
Components:
- Internet Gateway for public subnet
- NAT Gateway for private subnet internet access
- Custom Route Tables per subnet
- Security Groups:
- Bastion Host: Allow SSH (port 22) from specific IP
- Jenkins: Allow HTTP/HTTPS (80/443)
- App Server: Allow HTTP from Load Balancer only
- DB: Allow PostgreSQL (5432) from App subnet only
GCP VPC
- VPC CIDR: 10.1.0.0/16
- Subnets:
- Private Subnet: 10.1.1.0/24 - Prometheus, Grafana
- Firewall:
- Allow metrics and logs from AWS app subnet only
- Deny all other inbound traffic
A full network diagram is attached: see network-diagram.png
---
2. Compute Design
Component	Cloud	Instance Type	Use
Bastion Host	AWS	t2.micro	Secure SSH access to private nodes
Jenkins Server	AWS	t2.micro	CI/CD automation
OUTLINE
TIMELINE
Ln 123, Col 60 Spaces: 2 UTF-8 CRLF Markdown



```
# Low-Level Design (LLD) – Kimai Infrastructure
## 1. Network Design
### GCP VPC

## 2. Compute Design

| Component       | Cloud | Instance Type | Use                                  |
|-----------------|-------|---------------|--------------------------------------|
| Bastion Host    | AWS   | t2.micro      | Secure SSH access to private nodes |
| Jenkins Server  | AWS   | t2.micro      | CI/CD automation                     |
| Kimai App       | AWS   | t2.micro      | Dockerized web application           |
| Monitoring Node | GCP   | e2-micro      | Grafana and Prometheus               |

Auto Scaling:
- For App Server
  - Min: 1 instance
  - Max: 3 instances
  - Policy: Scale out when CPU > 70% for 5 minutes

---

## 3. Storage Design

### EC2 Volumes
- Bastion, Jenkins, Kimai: 8–20 GB General Purpose (gp2)

### Database (RDS PostgreSQL)
- Instance: db.t3.micro (free tier)
- Storage: 20 GB GP2
- Backup: Daily, retention = 7 days
- Encryption: Enabled
- Public access: Disabled

---

## 4. Security Design
```



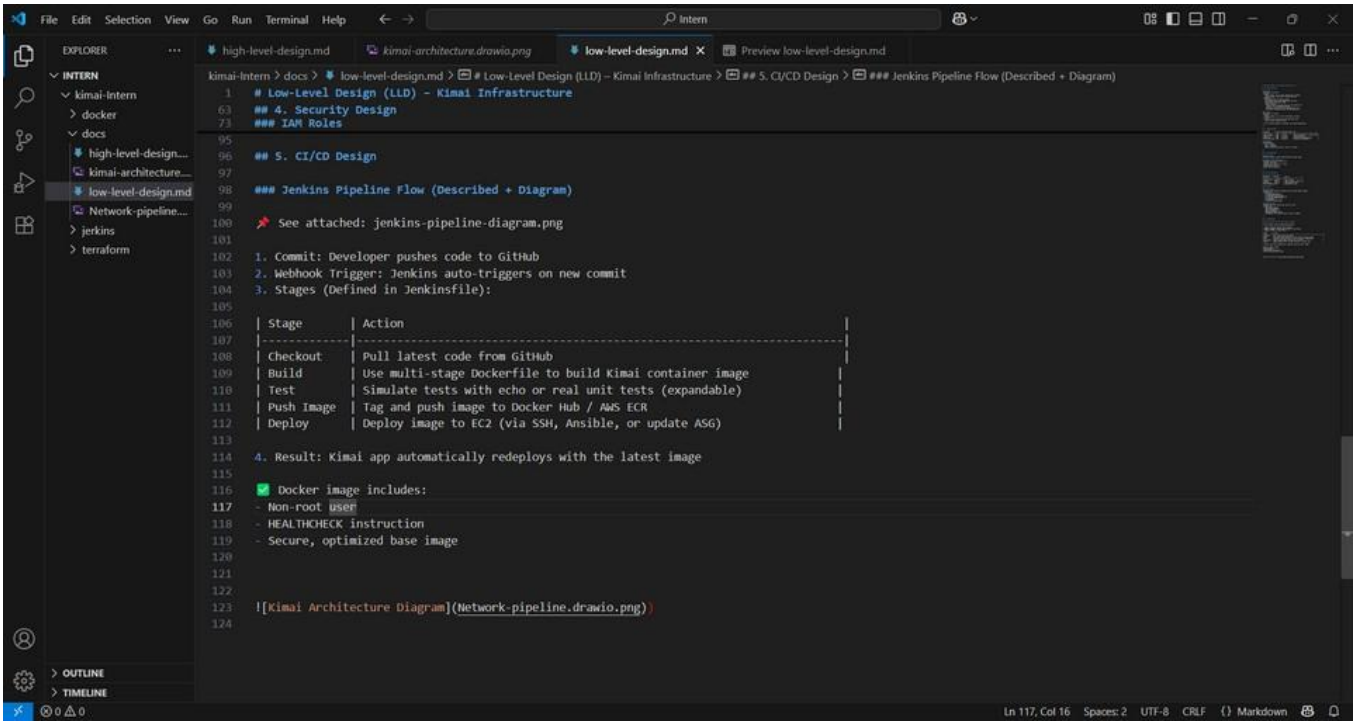
```
# Low-Level Design (LLD) – Kimai Infrastructure
## 3. Storage Design
### Database (RDS PostgreSQL)


## 4. Security Design

### Security Groups (AWS)
| Resource      | Ports       | Source                  |
|---------------|-------------|-------------------------|
| Bastion Host  | 22 (SSH)    | Your IP only            |
| Jenkins       | 8080        | Public subnet or Bastion |
| App Server    | 80, 443     | Load Balancer only      |
| RDS PostgreSQL| 5432        | App Server SG only      |

### IAM Roles

#### JenkinsRole
- Purpose: Allow Jenkins to deploy code, pull/push Docker images
- Permissions:
  - ec2:DescribeInstances
  - ecr:GetAuthorizationToken
  - ecr:BatchCheckLayerAvailability
  - ecr:GetDownloadUrlForLayer
  - ecr:PutImage
  - cloudwatch:PutMetricData

#### AppserverRole
- Purpose: App-level monitoring, access to logs
- Permissions:
  - logs:CreateLogGroup
  - logs:CreateLogStream
  - logs:PutLogEvents
  - cloudwatch:PutMetricData
  - s3:GetObject (for pulling static config if needed)

---
```



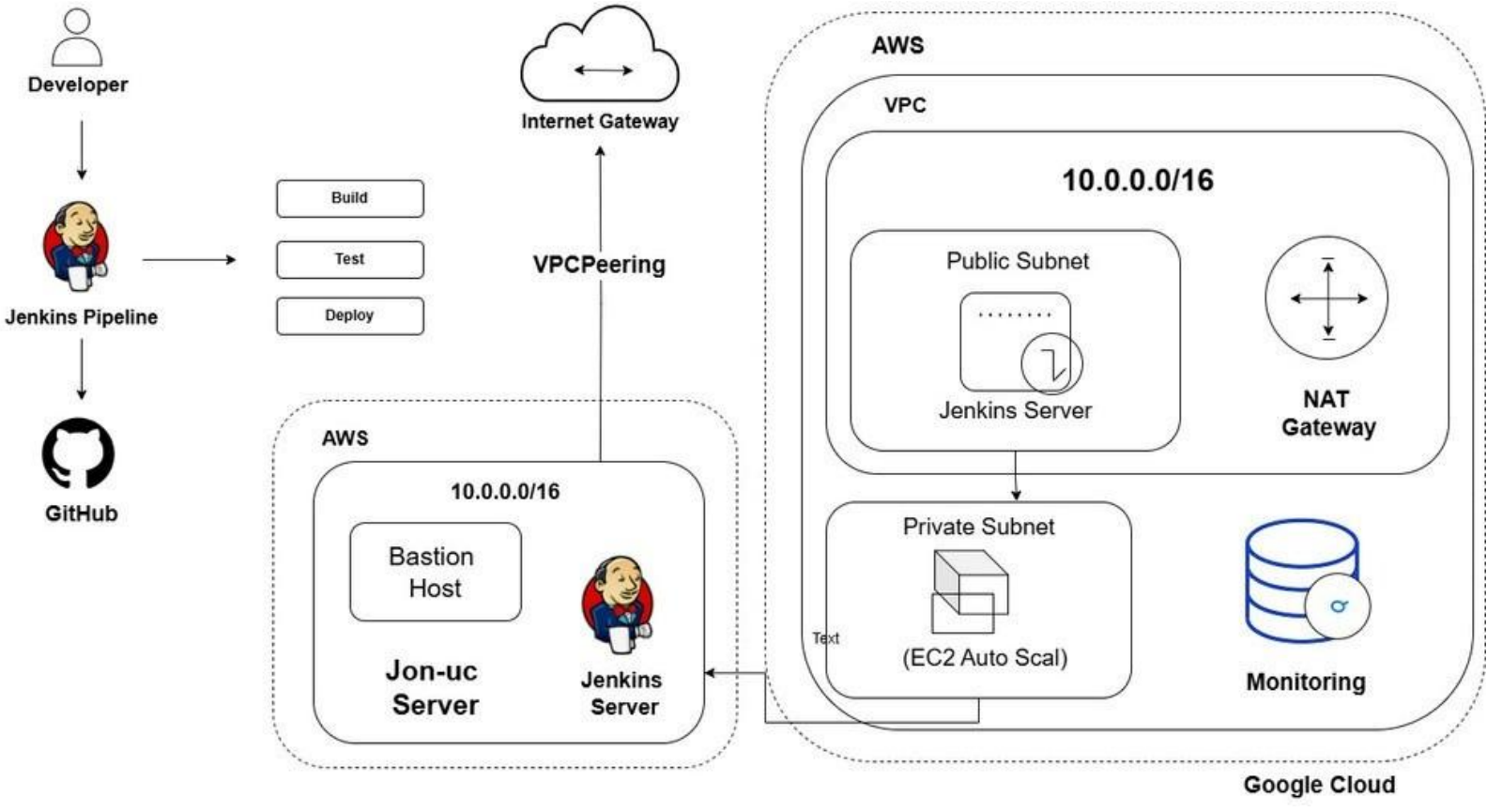
Infrastructure

Diagram

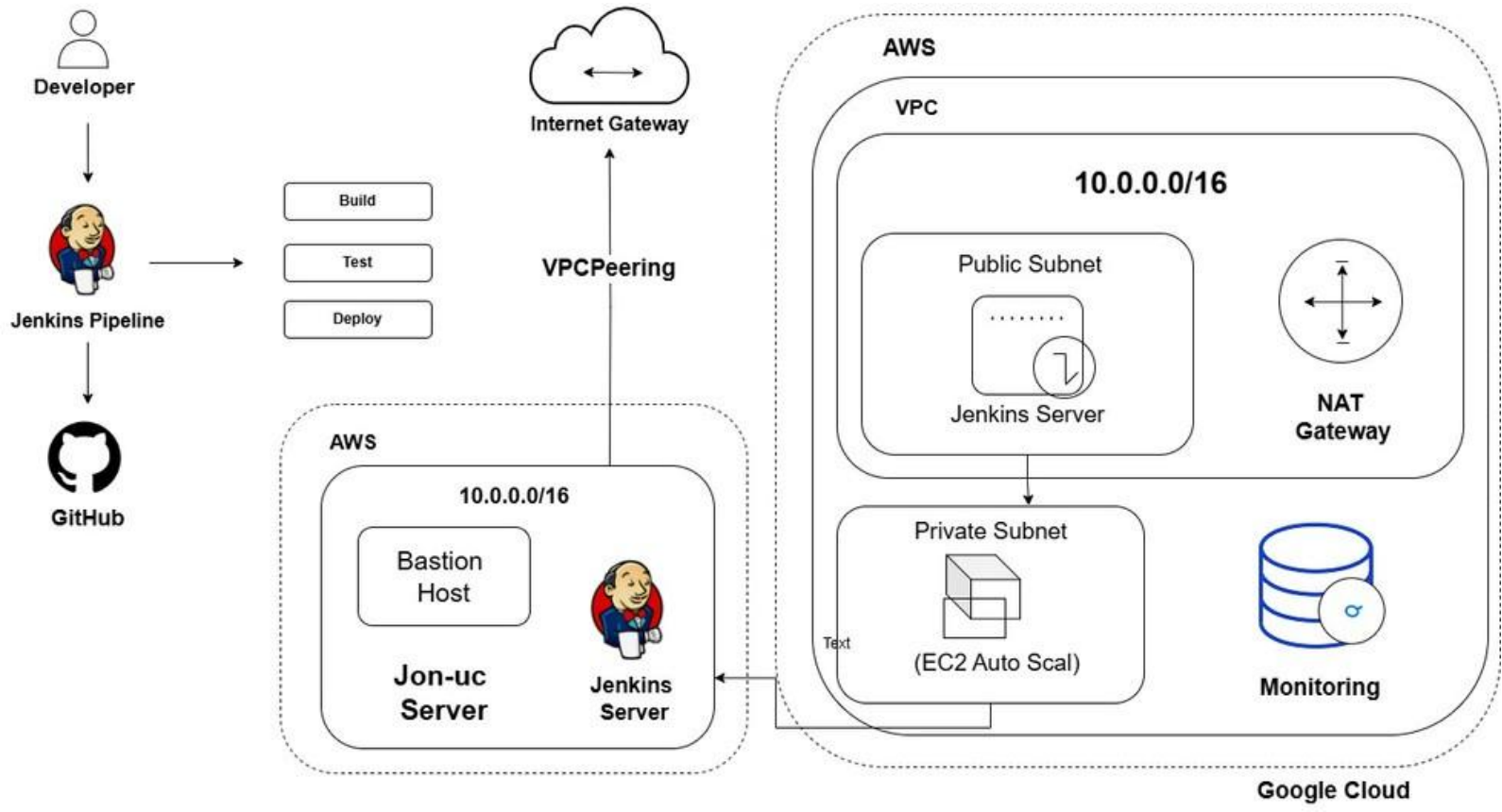
Low-Level Design



```
# Low-Level Design (LLD) - Kimai Infrastructure
## 4. Security Design
### IAM Roles

## 5. CI/CD Design

### Jenkins Pipeline Flow (Described + Diagram)

📌 See attached: jenkins-pipeline-diagram.png

1. Commit: Developer pushes code to GitHub
2. Webhook Trigger: Jenkins auto-triggers on new commit
3. Stages (Defined in Jenkinsfile):

| Stage       | Action                                                                 |
|-------------|------------------------------------------------------------------------|
| Checkout    | Pull latest code from GitHub                                           |
| Build       | Use multi-stage Dockerfile to build Kimai container image              |
| Test        | Simulate tests with echo or real unit tests (expandable)               |
| Push Image  | Tag and push image to Docker Hub / AWS ECR                             |
| Deploy      | Deploy image to EC2 (via SSH, Ansible, or update ASG)                  |

4. Result: Kimai app automatically redeploys with the latest image

✅ Docker image includes:
- Non-root user
- HEALTHCHECK instruction
- Secure, optimized base image



![Kimai Architecture Diagram](Network-pipeline.drawio.png)
```



Infrastructure Diagram
Low-Level Design



Phase 2: Infrastructure as Code (IaC)

Objective:

To write modular, reusable, and version-controlled Terraform code to automate the provisioning of the infrastructure designed in Phase 1.

Tools Used:

Terraform v1.12.2 Infrastructure as Code
AWS EC2 Testing and deployment environment
Visual Studio Code / Notepad Writing .tf files
AWS CLI Configure credentials for Terraform

Folder Structure:

```
techforce-infra-migration/
└── terraform/
    ├── main.tf
    ├── variables.tf
    ├── outputs.tf
    └── modules/
        ├── network/
        |   ├── main.tf
        |   ├── variables.tf
        |   └── outputs.tf
        ├── compute/
        |   ├── main.tf
        |   ├── variables.tf
        |   └── outputs.tf
        └── security/
            ├── main.tf
            ├── variables.tf
            └── outputs.tf
```

Modules Developed: A.

Network Module:

Files created:

main.tf: VPC, public and private subnets resources variables.tf: VPC

CIDR, public/private subnet CIDRs outputs.tf: Outputs VPC ID and

subnet IDs

B. Compute Module:

Files created:

main.tf: EC2 instance resources (bastion and Jenkins)

variables.tf: AMI ID, instance type, key pair, subnet IDs, security

groups outputs.tf: Outputs instance IDs and public IPs

C. Security Module:

Files created:

main.tf: Security group rules for bastion host and Jenkins

variables.tf: Input variables for security configurations outputs.tf:

Outputs security group IDs

Backend Configuration:

Configured in terraform/main.tf:

```
terraform {   backend
"s3" {
   bucket        = "my-terraform-state-bucket-012"
   key             = "terraform/state"      region
= "eu-north-1"
```

```
    dynamodb_table = "terraform-lock-table"
     }
    }
```

Purpose:

To store Terraform state file remotely and enable state locking for team collaboration.

AWS Credentials Setup:

Configured using:

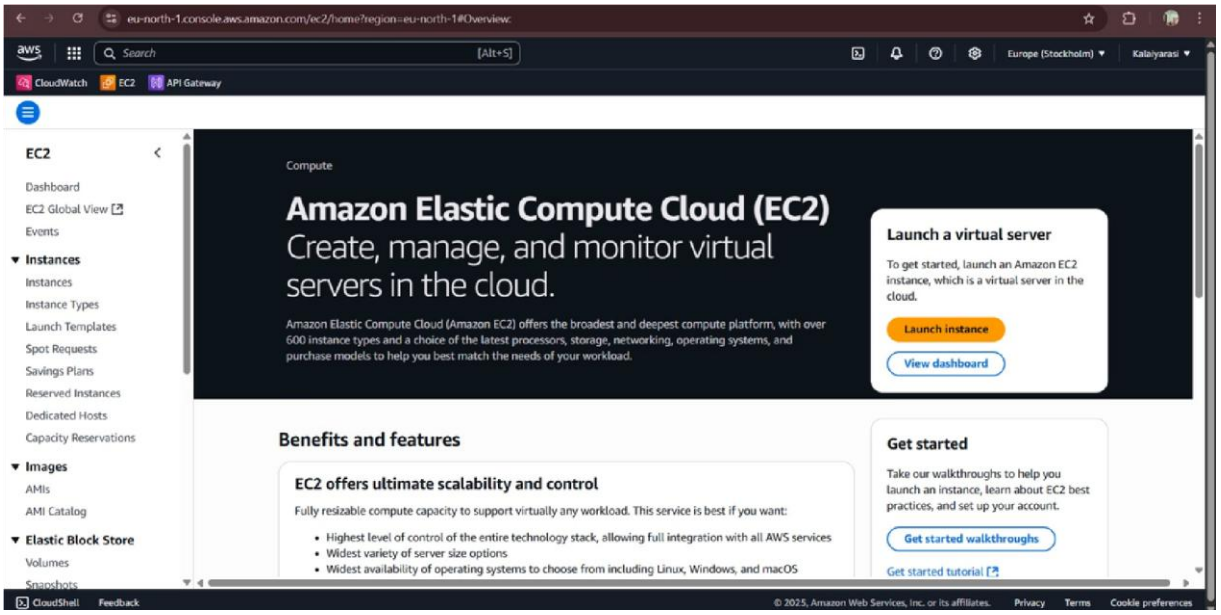
aws configure

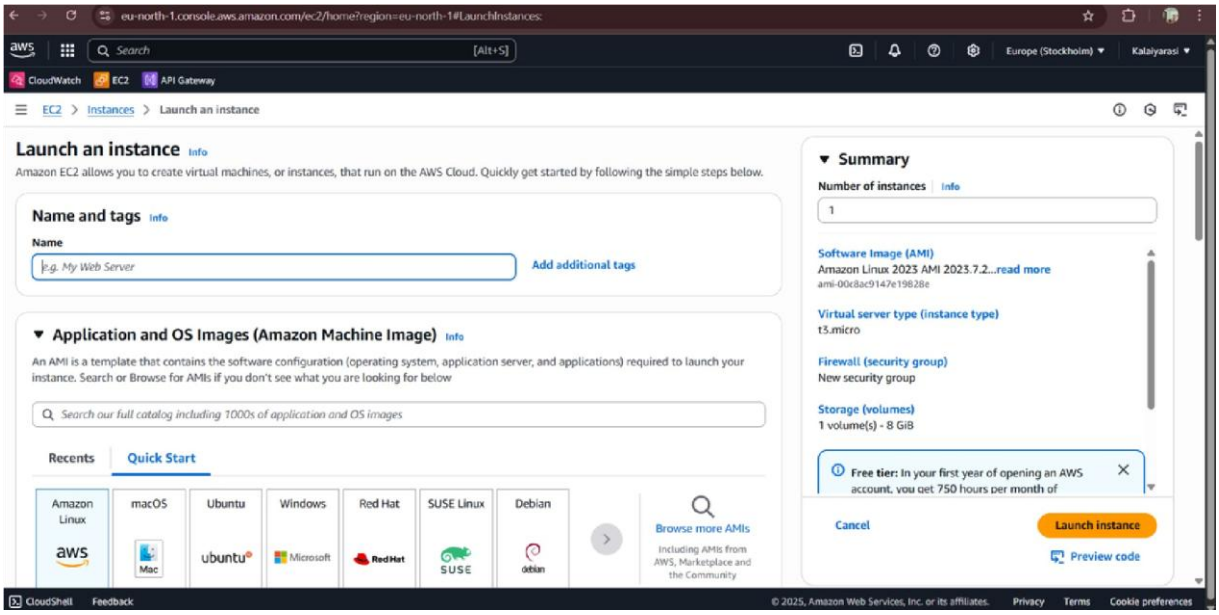
Access Key ID

Secret Access Key

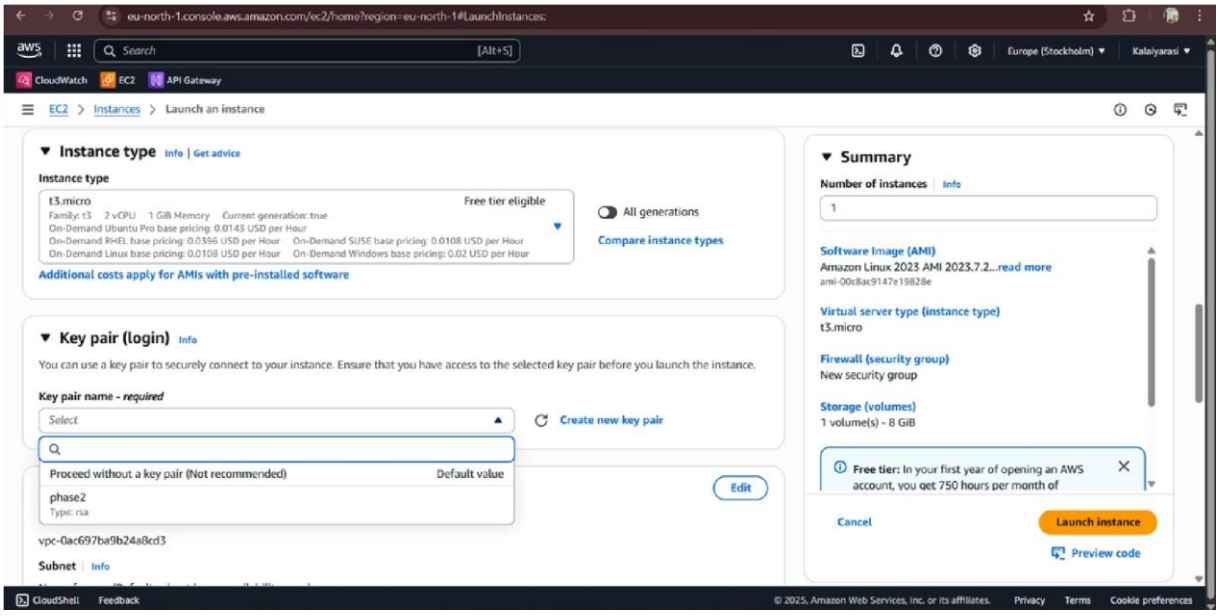
Default region name

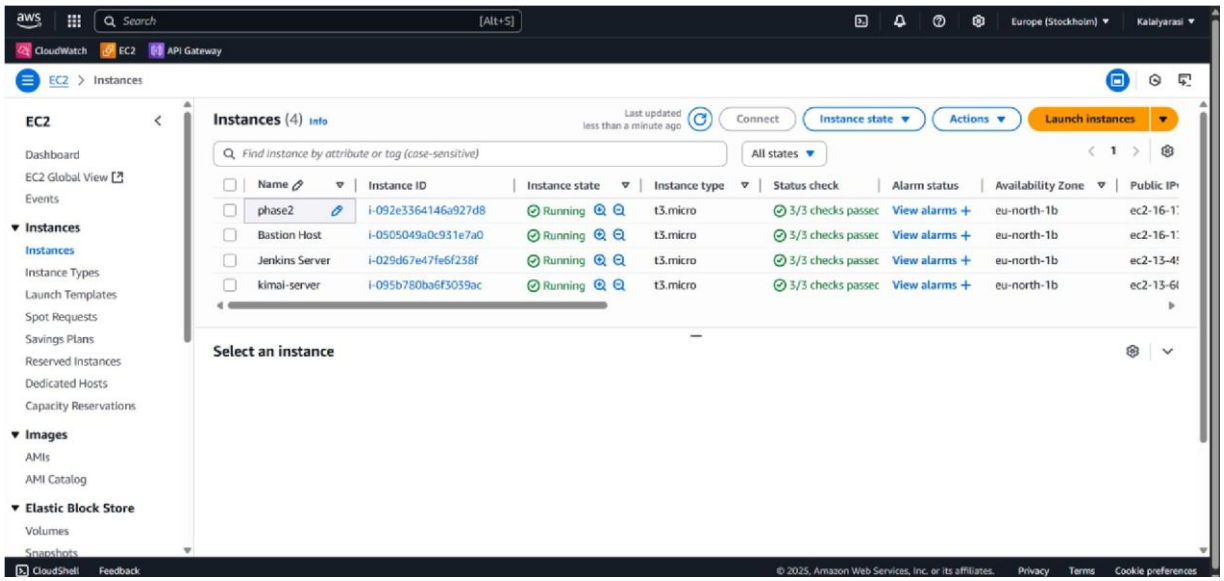
Default output format

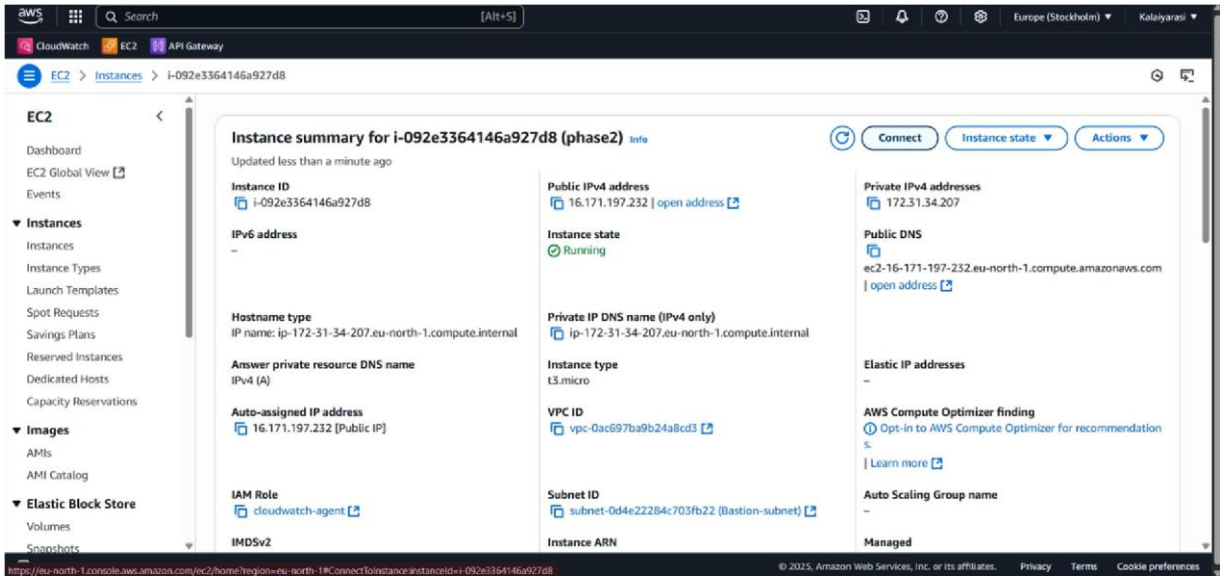


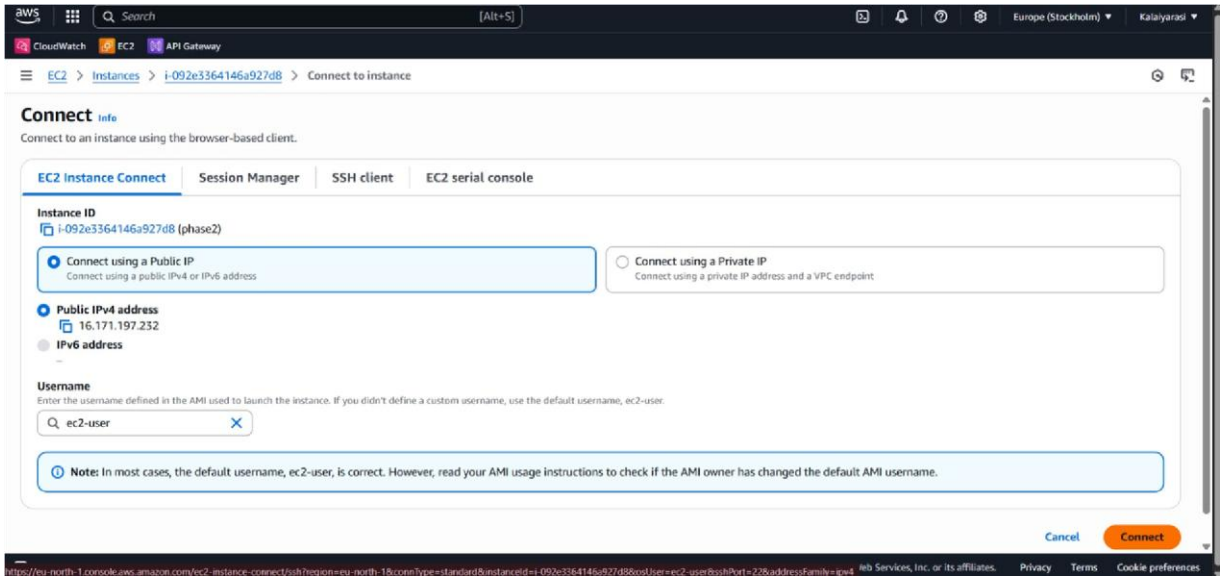


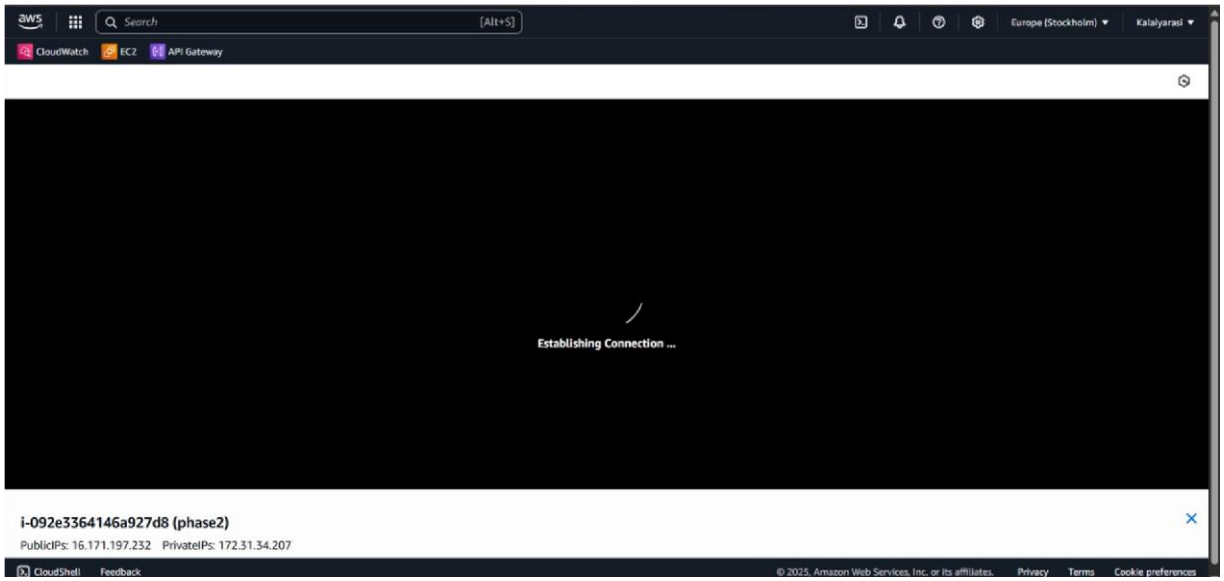
After creating instance's name and tag we are supposed to create a new key pair for the instance created.

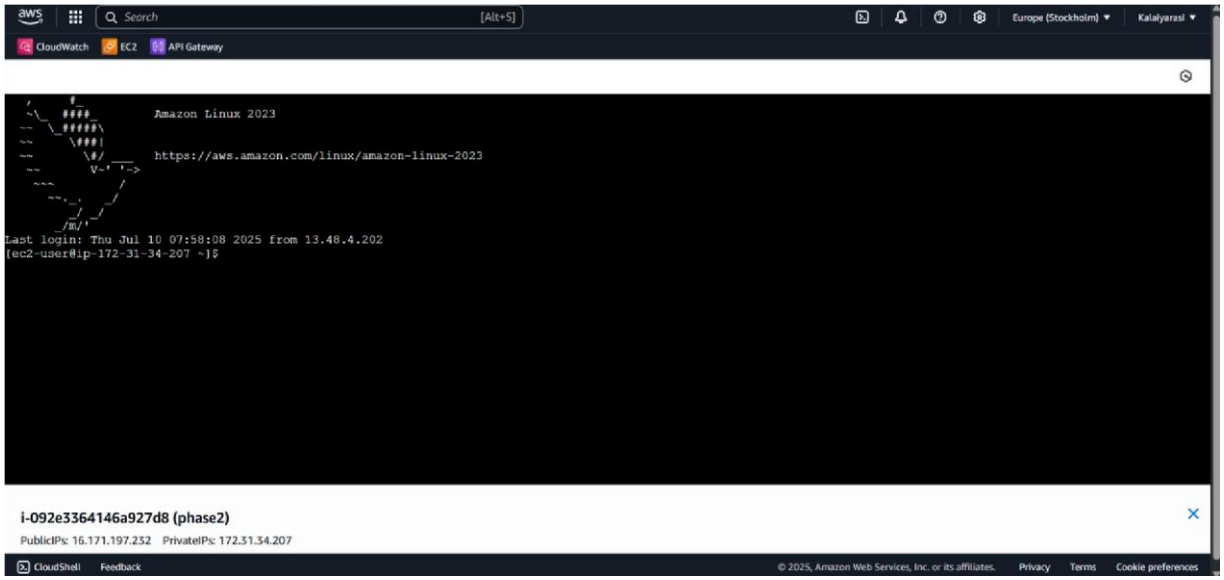



Search [Alt+S]
Europe (Stockholm)
Kalaiyarasi
CloudWatch
EC2
API Gateway
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Instances (4) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IP
phase2
i-092e3364146a927d8
Running
t3.micro
3/3 checks passed
View alarms
eu-north-1b
Bastion Host
i-0505049a0c931e7a0
Jenkins Server
i-029d67e47fe6f238f
kimai-server
i-095b780ba6f3039ac
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Europe (Stockholm)
Kalaiyarasi
CloudWatch
EC2
API Gateway
EC2 > Instances > i-092e3364146a927d8
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Instance summary for i-092e3364146a927d8 (phase2) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-092e3364146a927d8
Public IPv4 address
16.171.197.232 | open address
Private IPv4 addresses
172.31.34.207
IPv6 address
–
Instance state
Running
Public DNS
ec2-16-171-197-232.eu-north-1.compute.amazonaws.com | open address
Hostname type
IP name: ip-172-31-34-207.eu-north-1.compute.internal
Private IP DNS name (IPv4 only)
ip-172-31-34-207.eu-north-1.compute.internal
Answer private resource DNS name
IPv4 (A)
Instance type
t3.micro
Elastic IP addresses
–
Auto-assigned IP address
16.171.197.232 [Public IP]
VPC ID
vpc-0ac697ba9b24a8cd3
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role
cloudwatch-agent
Subnet ID
subnet-0d4e22284c703fb22 (Bastion-subnet)
Auto Scaling Group name
–
IMDSv2
Instance ARN
Managed
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Europe (Stockholm)
Kalaiyarasi
CloudWatch
EC2
API Gateway
EC2 > Instances > i-092e3364146a927d8 > Connect to instance
Connect Info
Connect to an instance using the browser-based client.
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
Instance ID
i-092e3364146a927d8 (phase2)
Connect using a Public IP
Connect using a public IPv4 or IPv6 address
Connect using a Private IP
Connect using a private IP address and a VPC endpoint
Public IPv4 address
16.171.197.232
IPv6 address
–
Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.
ec2-user
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.
Cancel
Connect
Privacy
Terms
Cookie preferences





Commands Executed:

1. Initialize Terraform
   terraform init

2. Validate and Plan
   terraform plan

3. Apply to Provision Infrastructure
   terraform apply

Errors Faced and Solutions:

Error	Reason	Solution
Unsupported argument in module	Variables not defined in module `variables.tf`	Added missing variable definitions
No valid credential sources found	IAM user/role permissions missing	Configured AWS CLI with credentials
S3 403 Forbidden	Bucket policy or IAM permissions missing	Updated IAM permissions and bucket policy

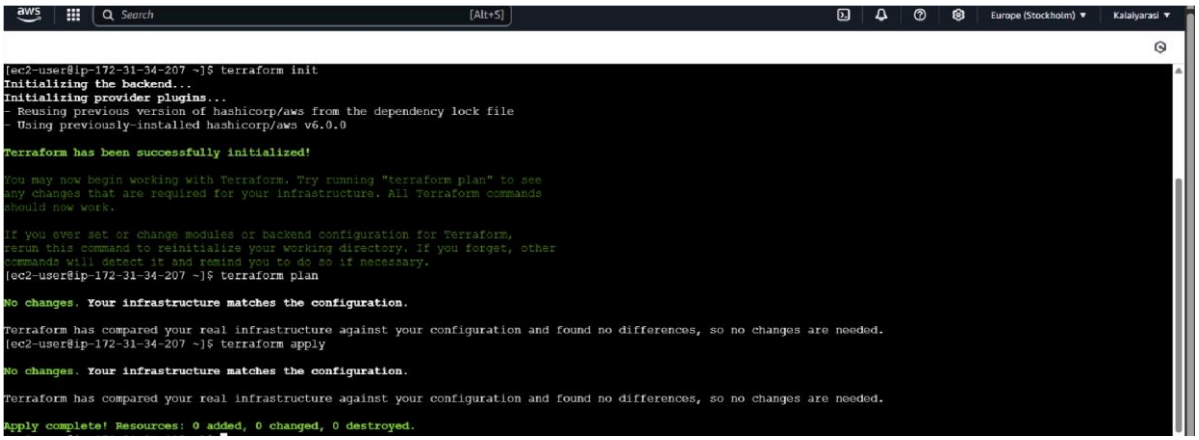
Outcomes:

Modular Terraform code for:

Network infrastructure (VPC, subnets)

Compute resources (EC2 instances for bastion and Jenkins)

Security configurations (security groups)



Remote backend configuration with S3 and DynamoDB.

Verified provisioning success using terraform plan and apply.

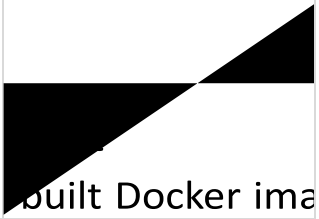
PHASE 3: APPLICATION DEPLOYMENT AND CI/CD

OBJECTIVE:

To containerize the Kimai application and create a fully automated CI/CD pipeline for its deployment.

TOOLS & TECHNOLOGIES USED:

Docker Hub	Containerize the Kimai HTML
Jenkins	Automate build and push pipeline
GitHub	Source code hosting and

AWS EC2	 Host the website live on a server
Docker Hub	 built Docker ima

PROJECT STRUCTURE:

```
Kimai-TechForce/
 ├── docker/
|    ├── Dockerfile
|    └── index.html
 ├── jenkins/
|    └── Jenkinsfile
```

STEP BY STEP WORKFLOW: STEP 1:

PREPARE WEBSITE FILES

index.html was created with simple UI content for Kimai homepage.

These files were copied inside the docker/ folder.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta http-equiv="refresh" content="0;
URL='https://www.kimai.org/'" />
  <title>Redirecting to Kimai.org...</title>
</head>
<body>
 <p>Redirecting to <a
href="https://www.kimai.org/">Kimai.org</a></p>
</body>
</html>
```

STEP 2: DOCKERFILE (INSIDE DOCKER/ FOLDER)

```
FROM nginx:alpine
WORKDIR /usr/share/nginx/html
COPY index.html . EXPOSE 80
```

This Dockerfile uses a lightweight NGINX base image and replaces the default page with your index.html.

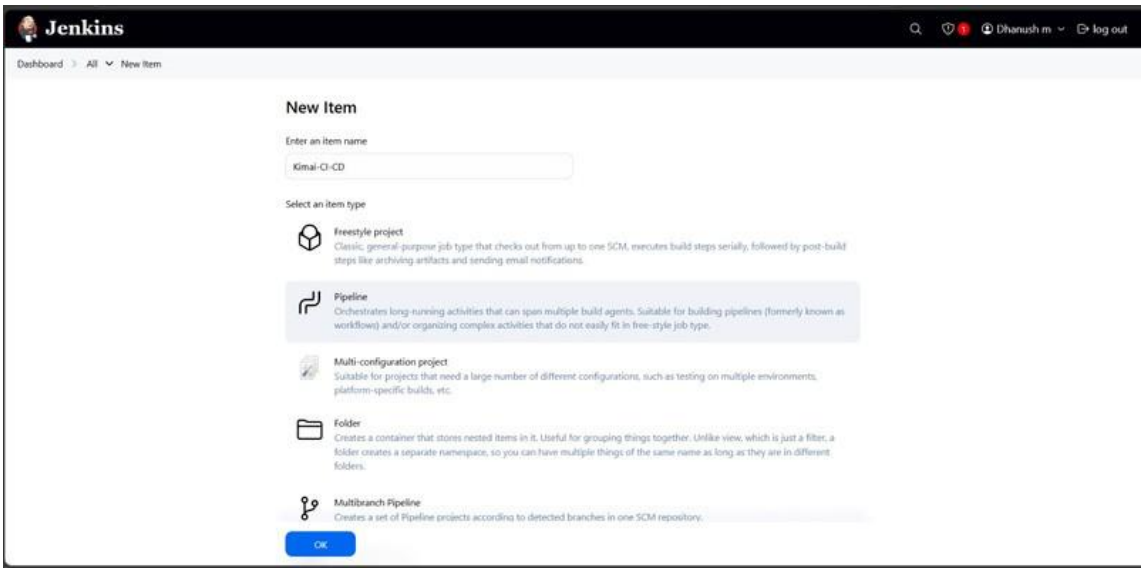
STEP 3: JENKINSFILE (INSIDE JENKINS/ FOLDER)

```
pipeline {     agent
any

   environment {
      DOCKER_IMAGE = "kalaisk/kimai-html:latest"
 }

 stages { stage('Checkout') {
 steps {
 git url: 'https://github.com/subadevanc/Kimai-
TechForce.git'
 }
 }
 stage('Build Image') {  steps {
 dir('docker') {
 bat "docker build -t ${DOCKER_IMAGE} ."
 }
 }
 }
 stage('Push to DockerHub') {
 steps {  withCredentials([usernamePassword(credentialsId:
'docker-token', usernameVariable: 'DOCKER_USER',
passwordVariable: 'DOCKER_PASS')]) {  bat """
 echo %DOCKER_PASS% | docker login -u %DOCKER_USER% --
password-stdin
   docker push${DOCKER_IMAGE}
   """
    }
```

```
    }
  }
 }
}
```

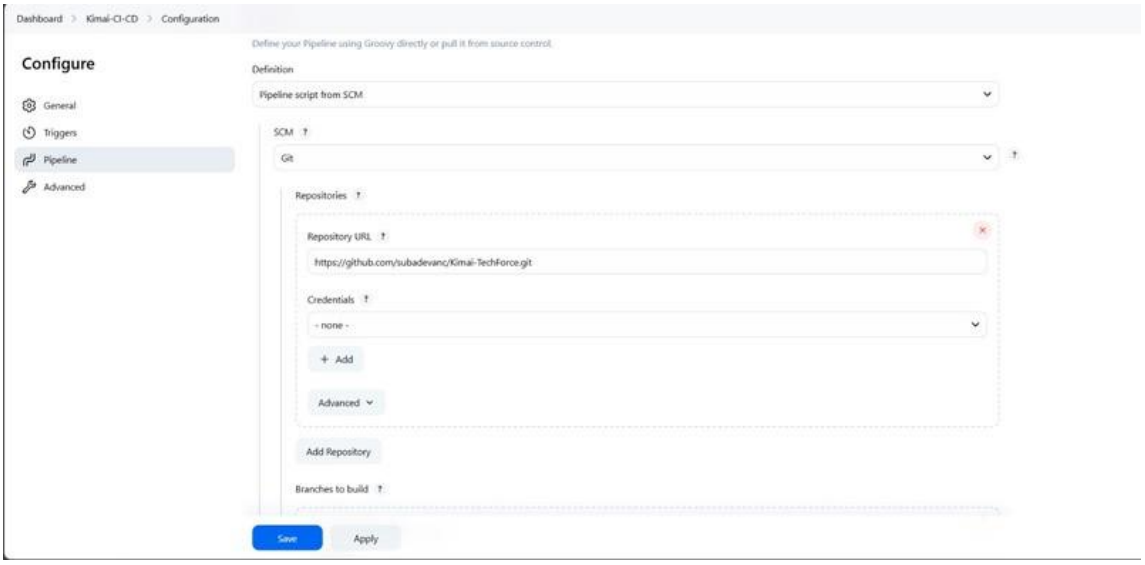
STEP 4: JENKINS PIPELINE SETUP

1. Open Jenkins Dashboard → New Item →
Pipeline

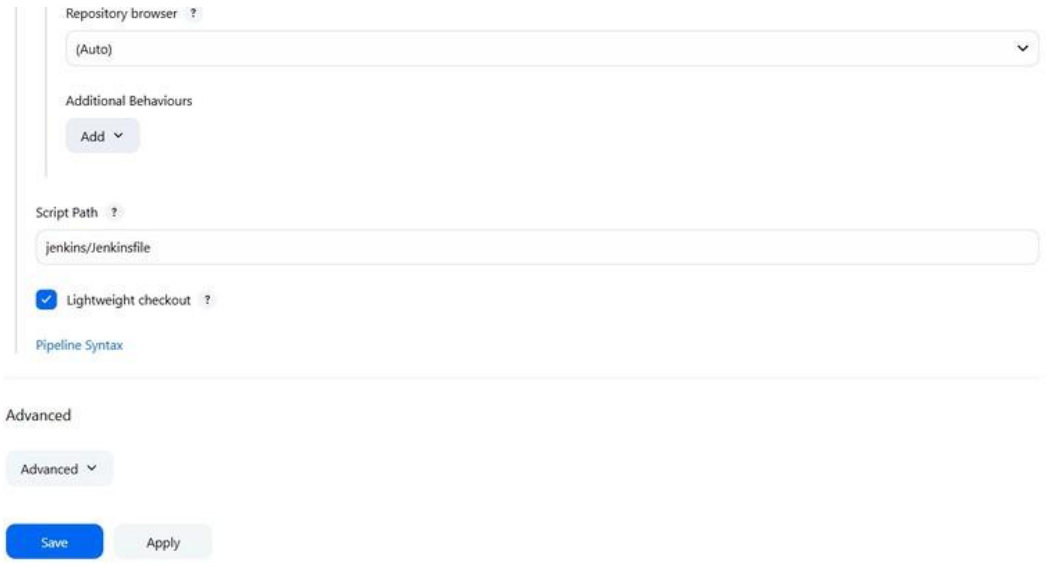


2. Set GitHub repo URL:

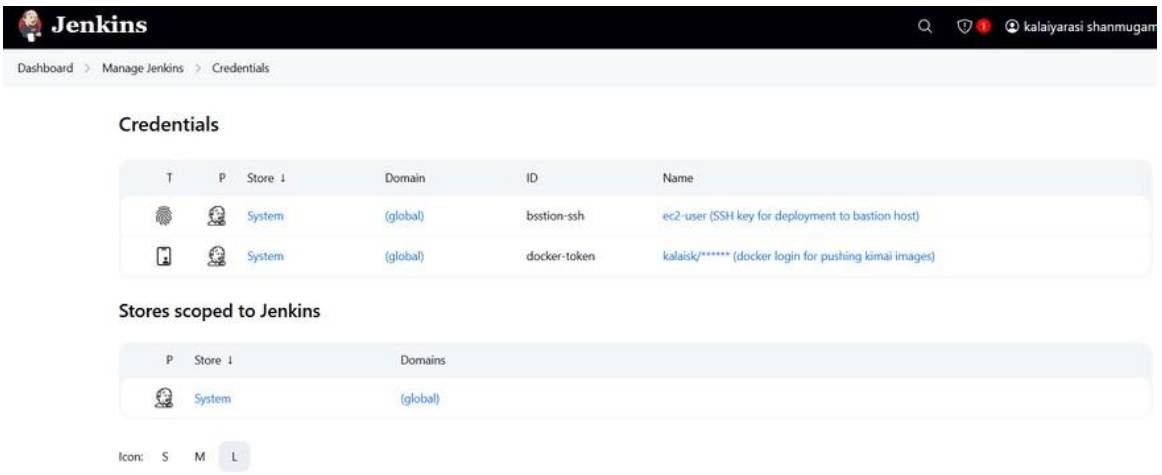
https://github.com/subadevanc/Kimai-TechForce.git



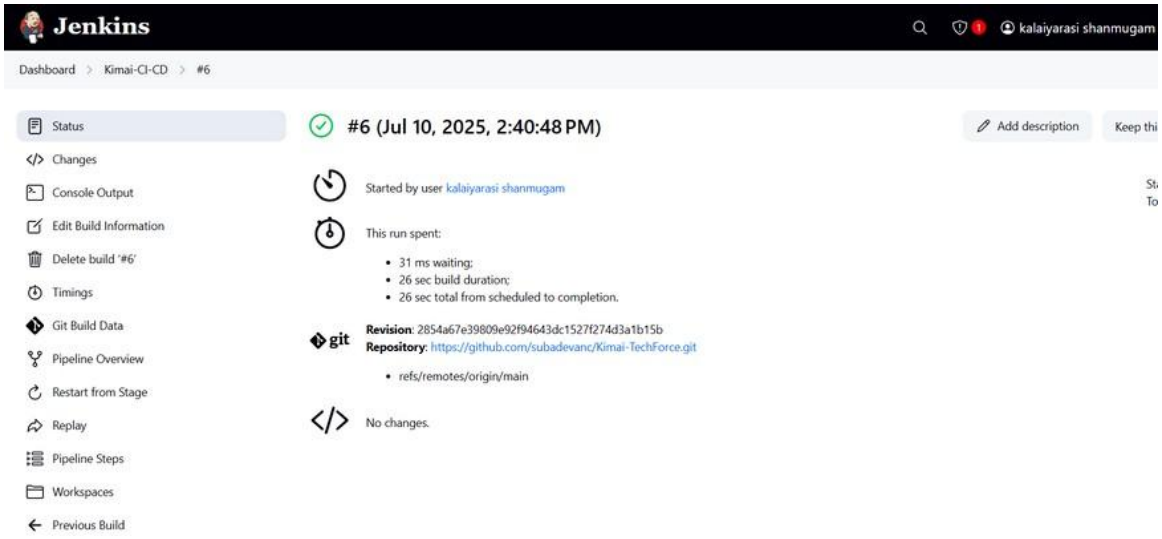
3. Jenkinsfile path: jenkins/Jenkinsfile

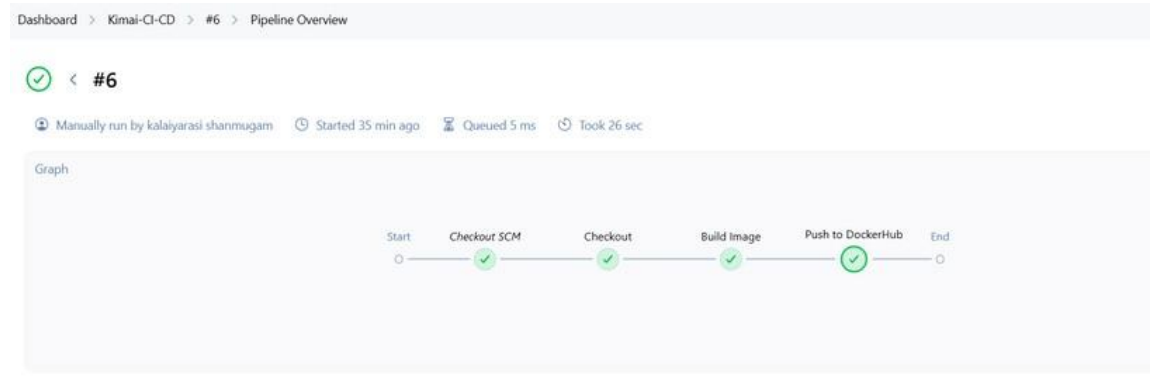


4. Add DockerHub credentials in Jenkins →
   Credentials → ID = docker-token



5. Click Build Now





STEP 5: DEPLOY DOCKER IMAGE ON AWS EC2

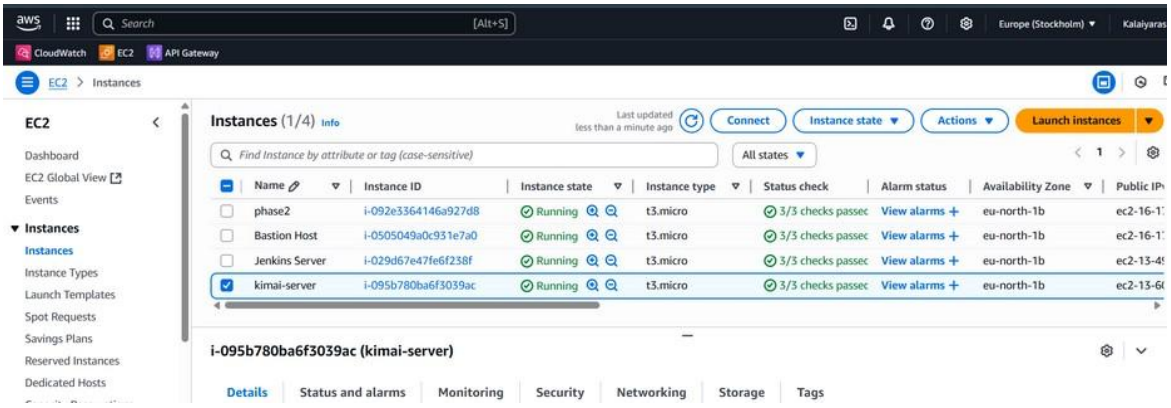
1. Launch EC2 Ubuntu 24.04 instanceGo to AWS

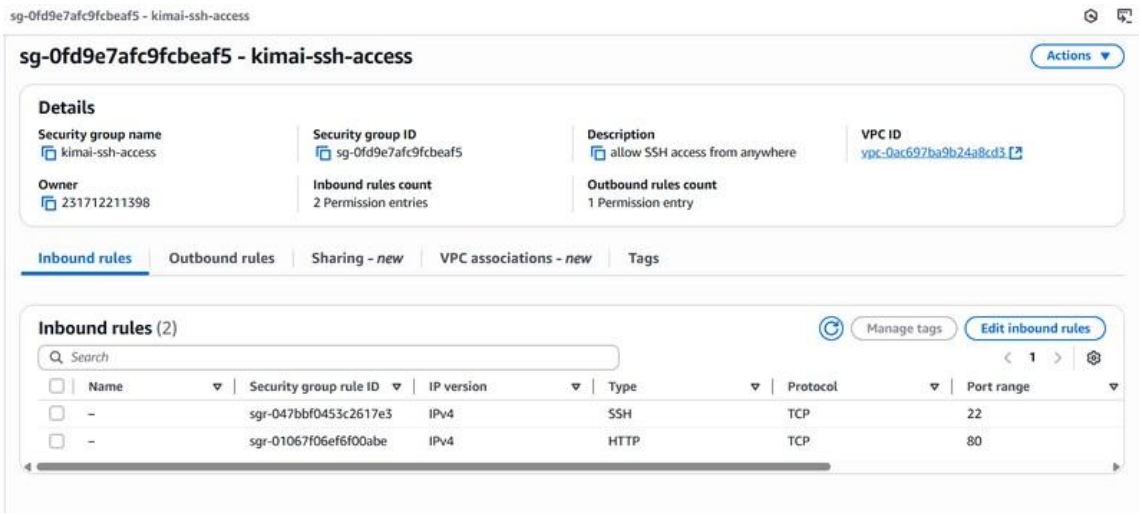
→ EC2 → Launch instance

Choose Ubuntu 24.04

Allow SSH (22) and HTTP (80) in Security Group

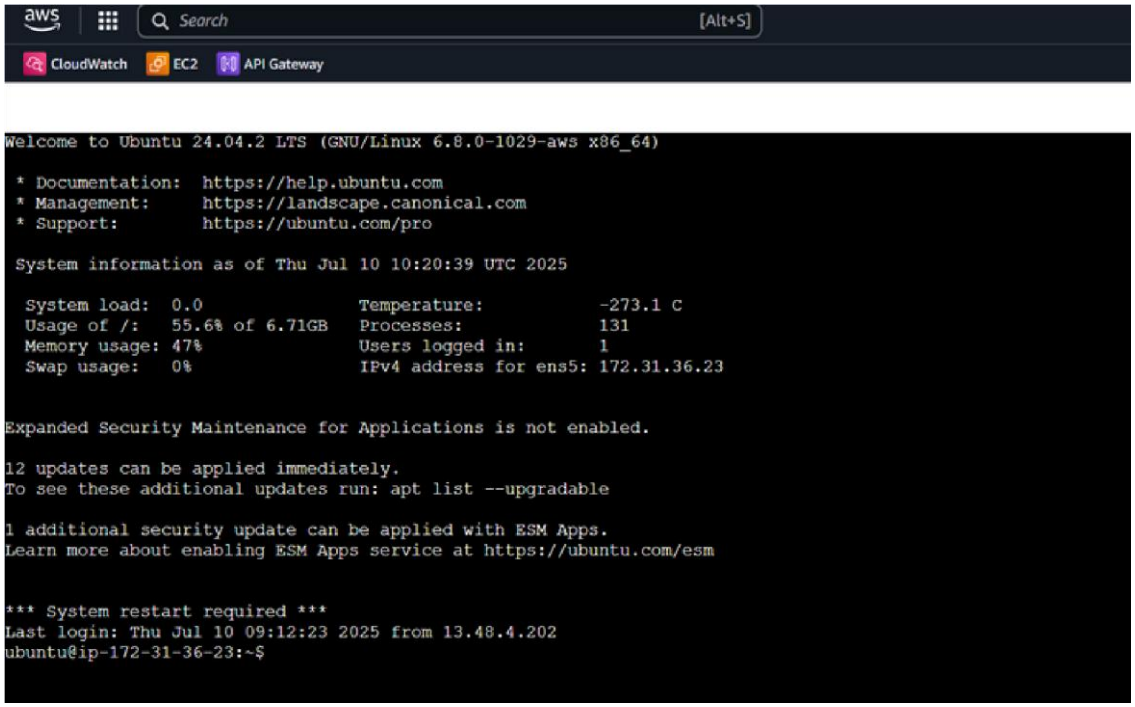
Download your .pem key (e.g., phase2.pem)





2. SSH into EC2 ssh -i "phase2.pem"

   ubuntu@13.60.234.66



3. Install Docker

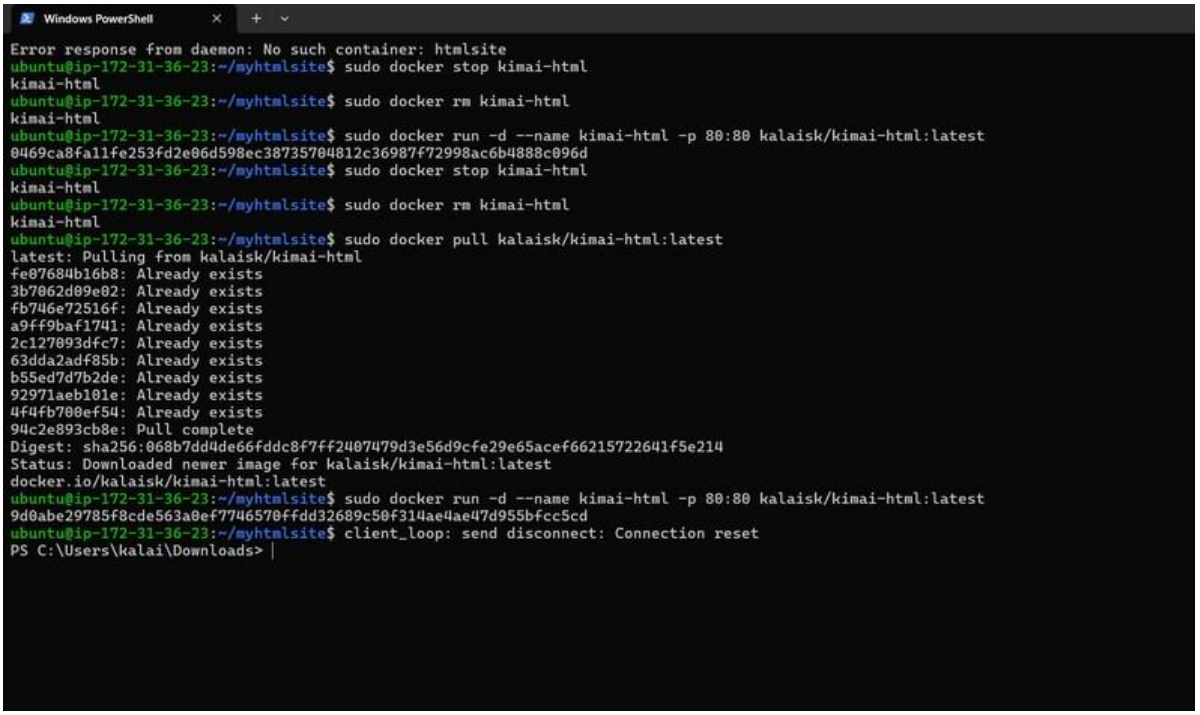
sudo apt update sudo apt install

docker.io -y sudo systemctl enable

docker sudo systemctl start docker

4. Pull & Run Docker Image

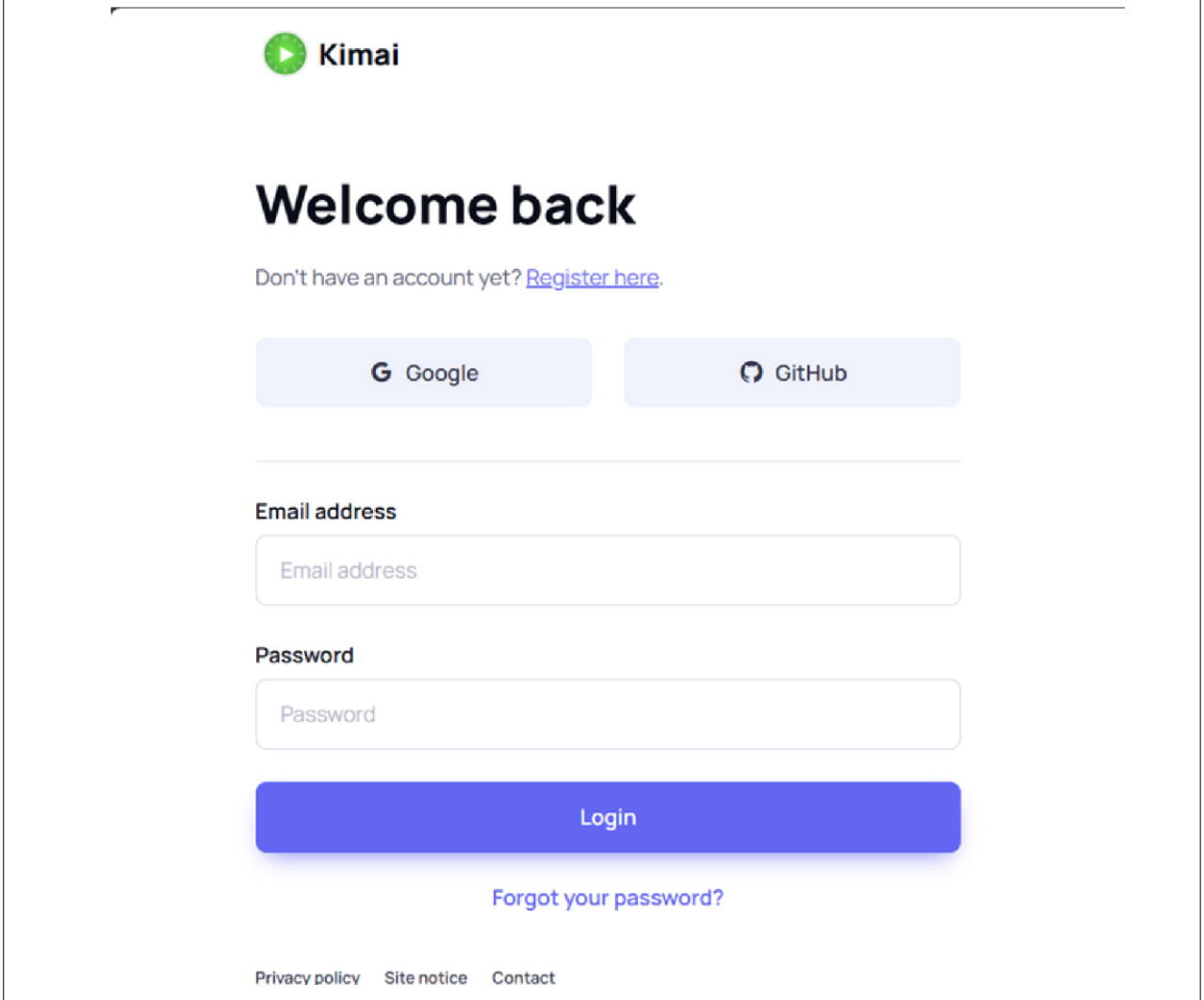
sudo docker pull kalaisk/kimai-html:latest sudo docker run -d --name kimai-html -p 80:80 kalaisk/kimai-html:latest



5. Open Your Website

In browser:

http://13.60.234.66


Kimai
Welcome back
Don't have an account yet? Register here.
Google
GitHub
Email address
Email address
Password
Password
Login
Forgot your password?
Privacy policy
Site notice
Contact

PHASE 4: SECURITY IMPLEMENTATION

OBJECTIVE

The goal of this phase is to implement robust security controls at every layer of the infrastructure—network, system, application, and identity—ensuring secure deployment and limited access exposure of critical resources.

---

1. NETWORK HARDENING

- Implemented AWS Security Groups as per the LLD to control inbound and outbound traffic.
- Allowed only necessary traffic:
    - Port 22 (SSH) – open for Bastion Host to allow remote access.
    - Port 80 (HTTP) – open for web access to the Kimai application.
- Ensured that no application or database server is exposed directly to the internet.
- Deployed the Application Load Balancer (ALB) in public subnets across 2 Availability Zones, making the service highly available and internet-facing.

Evidence:

Security Group configuration

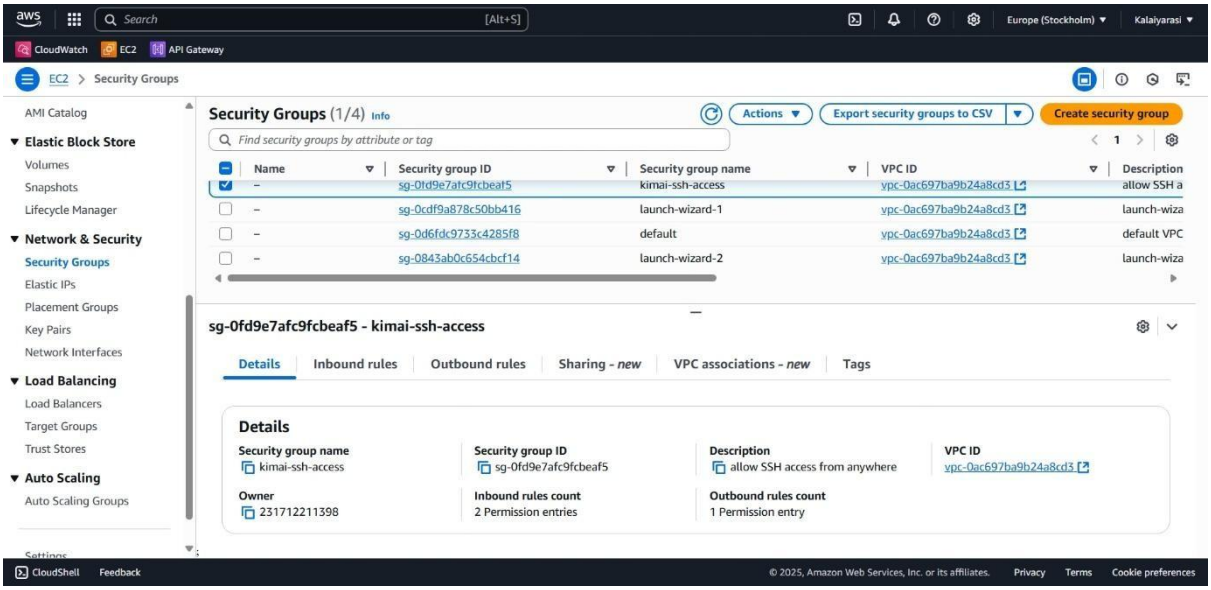


Fig1: AWS Security Group kimai-ssh-access configured with minimal permissions to ensure limited exposure.

Application Load Balancer (ALB)

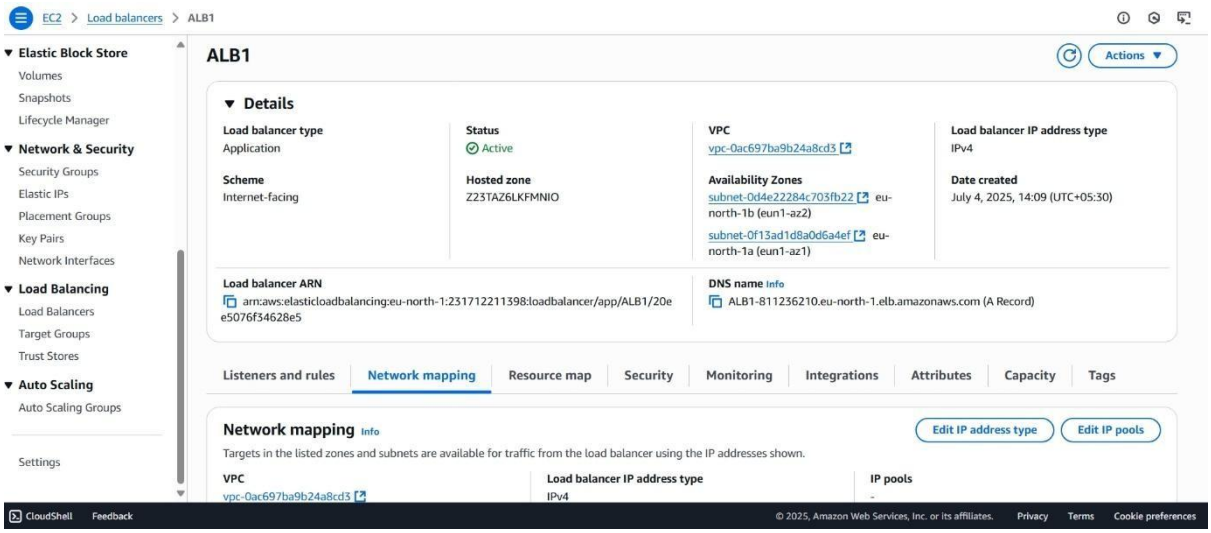


Fig2 : ALB Configuration

---

2. SECURE REMOTE ACCESS

- Deployed a **Bastion Host** in the **public subnet**.
    - Allowed SSH (port 22) access **only** from a **trusted IP range**.
- All other EC2 instances (Application Servers, Jenkins) were placed in **private subnets**.
    - These allowed SSH only from the **Bastion Host's security group**.
- **SSH Key-Based Authentication** was enforced.
    - **Disabled password login** by modifying sshd_config.

Evidence:

Bastion Host in Public Subnet

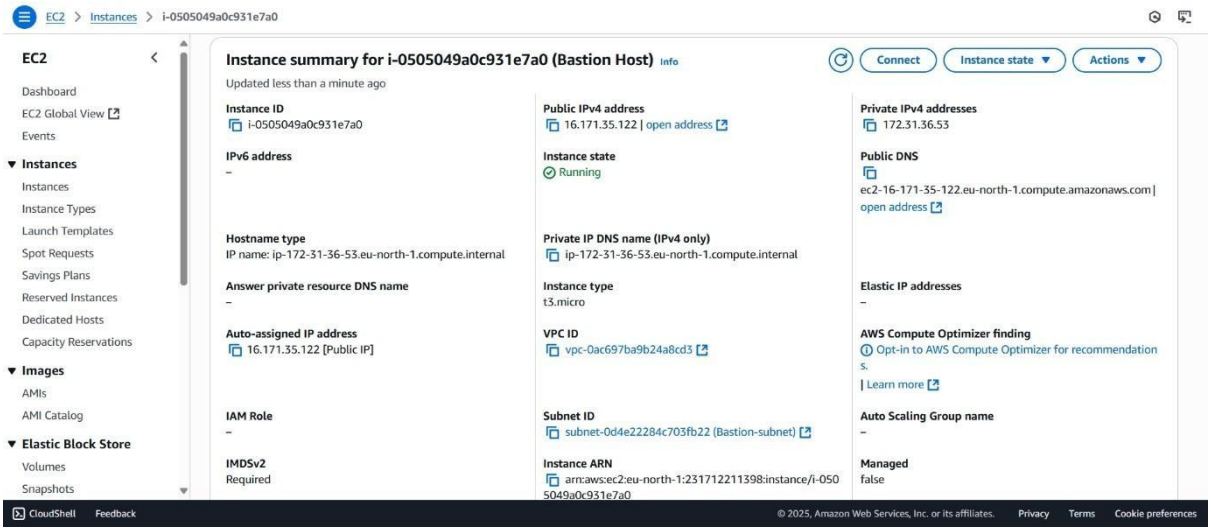


Fig3: Bastion Host deployed in public subnet with assigned public IP address for remote access.

Fig4 : Password authentication disabled in /etc/ssh/sshd_config; only SSH key-based access is allowed, fulfilling secure remote access compliance.

---

3. IDENTITY AND ACCESS MANAGEMENT (IAM)

Activities:

- Created IAM roles for each major component: ○ JenkinsRole, AppServerRole, BastionRole
- Attached **least privilege policies** to each role, e.g.:
  - JenkinsRole: Read from ECR, CloudWatch Logs ○ AppServerRole: Access S3 Bucket for logs
- Ensured **no access keys** were hardcoded.
  - Used **IAM Role-based access** on EC2 instances.
  - Validated via curl http://169.254.169.254/latest/meta-data/iam/security-credentials/

Evidence:

IAM Roles List

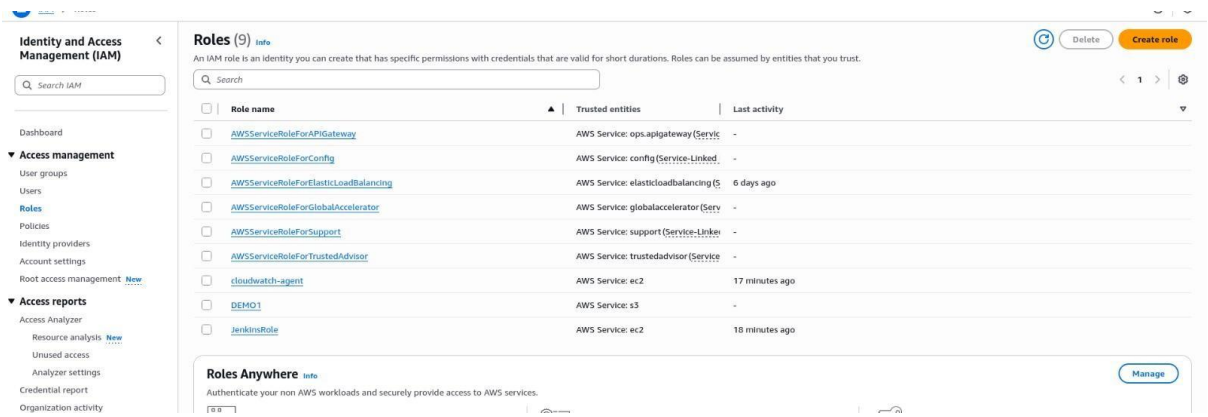


Fig5: IAM roles like JenkinsRole and cloudwatch-agent were created and configured for secure, temporary credential-based access via EC2.

Custom Policies Attached- IAM Policies

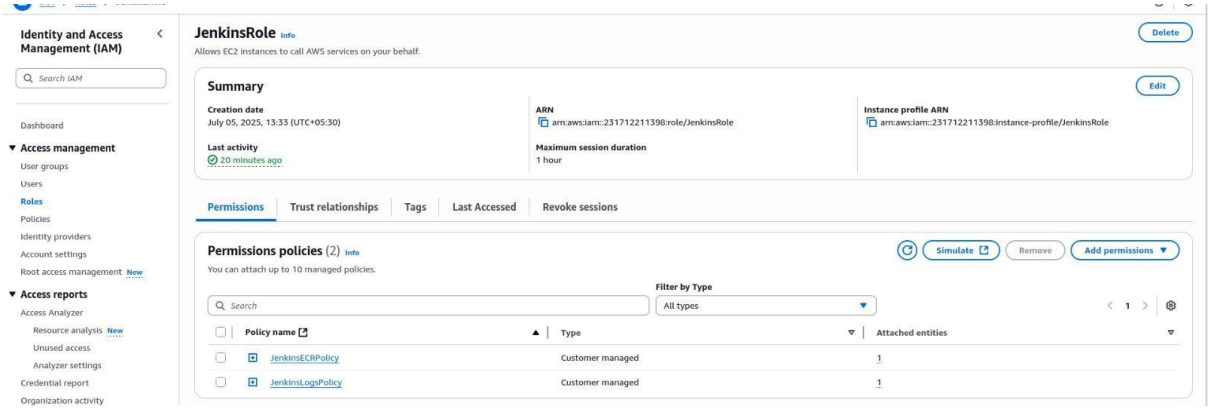


Fig6: IAM role JenkinsRole configured with custom permissions (JenkinsECRPolicy, JenkinsLogsPolicy), attached to EC2 instance to ensure secure, keyless access to AWS resources.

Jenkins EC2 Instance with IAM Role

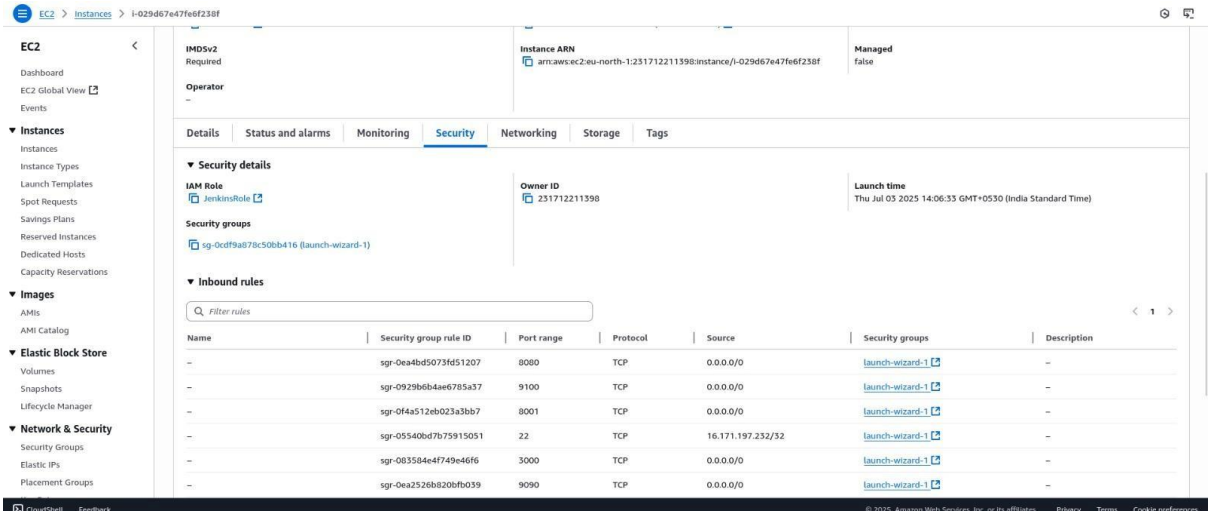


Fig7: IAM role JenkinsRole attached to EC2 instance to ensure keyless access to AWS services, with port 22 restricted to a trusted IP for secure administration.

---

4. SECURITY AUDIT AND VALIDATION

Tools Used:

- **Nmap** to scan exposed ports from public IPs
- **AWS Security Hub** for configuration compliance

Nmap Scan

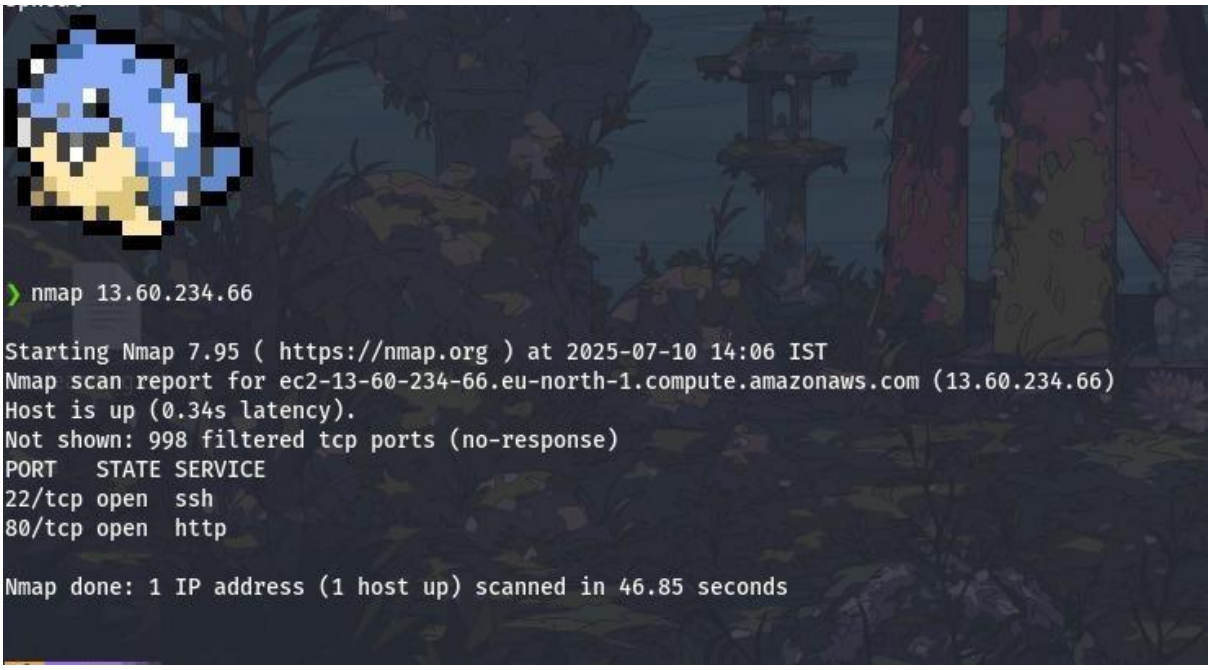


Fig8: Nmap scan results confirming that only essential ports (22 for SSH and 80 for HTTP) are open on the EC2 instance

Security Audit – Port Exposure Validation

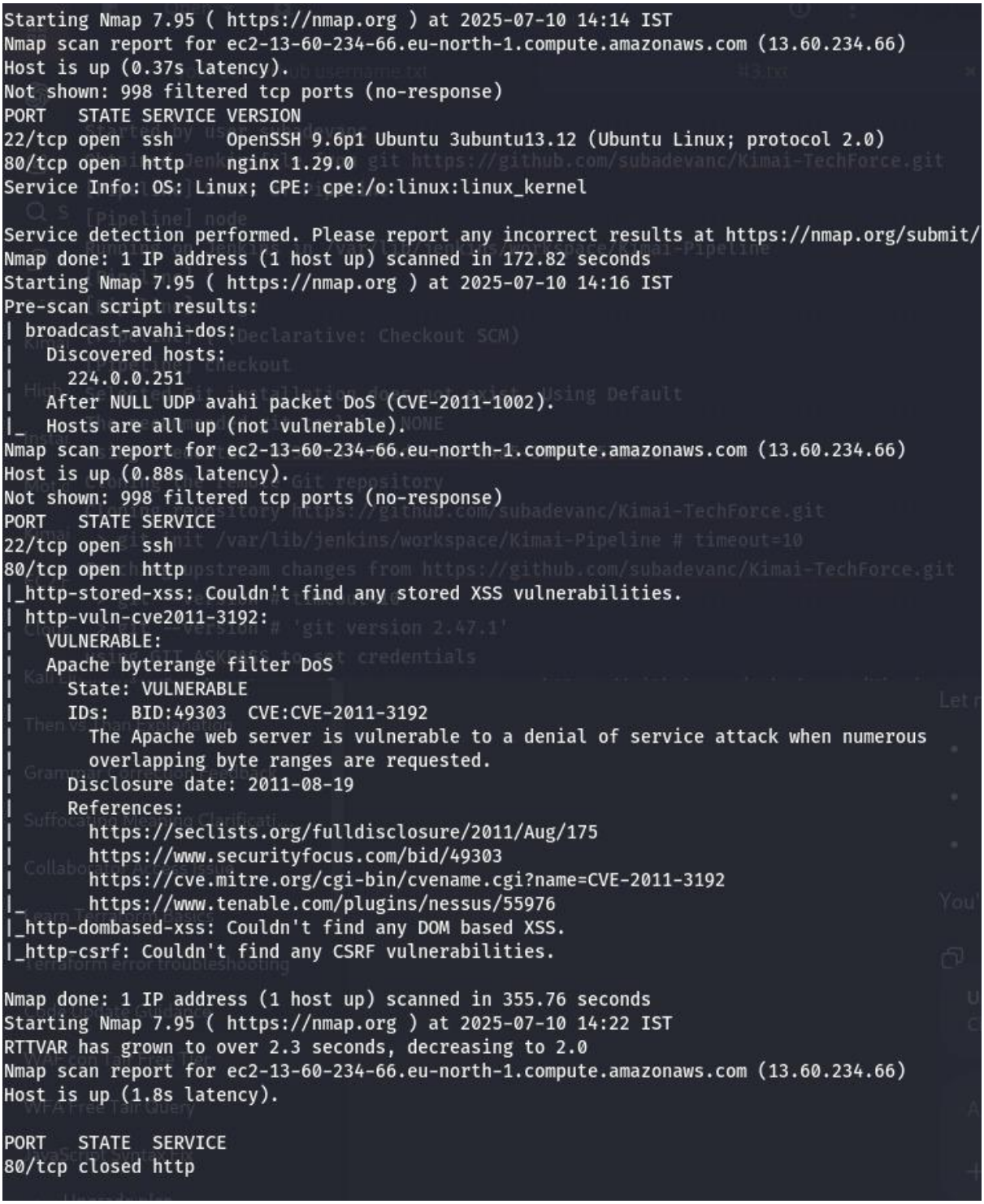


Fig9: Security audit using nmap confirms that all unnecessary ports are blocked by firewall/security groups

---

CONCLUSION

This phase successfully implemented multiple layers of security, including IAM roles, security group configurations, and SSH hardening, to protect the infrastructure from external threats. By enforcing strict access controls and eliminating public exposure of critical resources, the environment is now well-secured and aligned with best practices — making it suitable for production-level deployment.

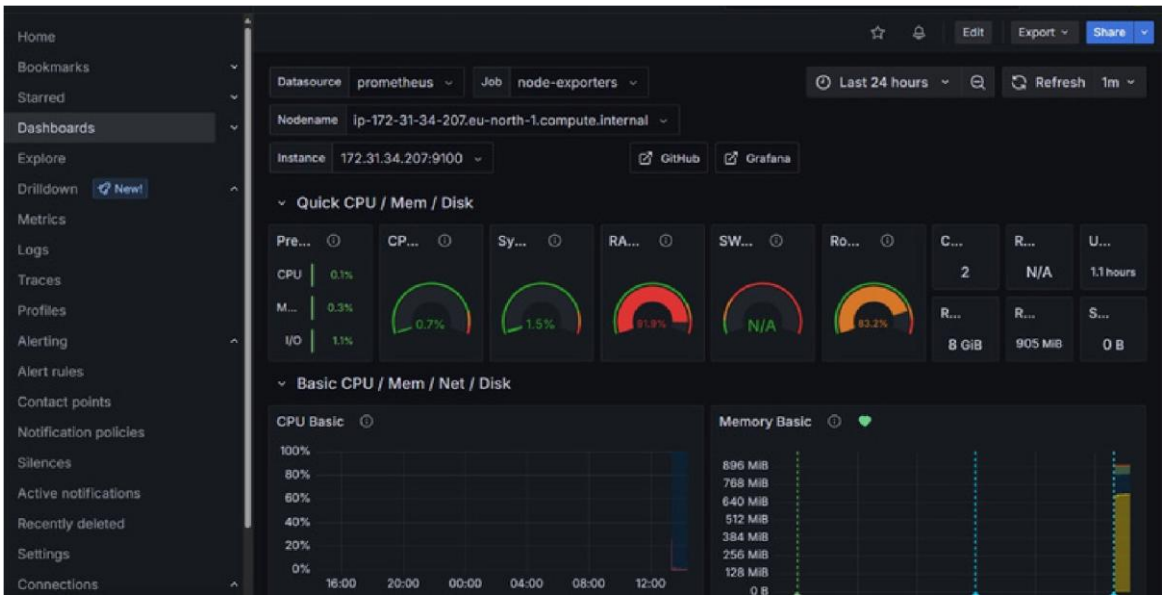
Phase 5: Monitoring, Logging, and Alerting Documentation

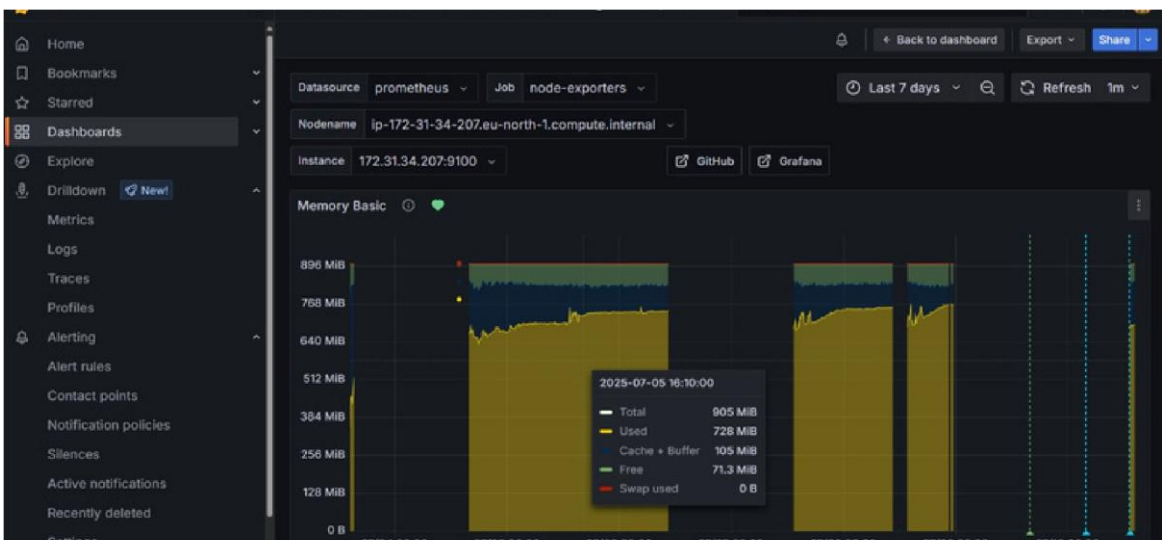
1. Monitoring Setup (Prometheus + Grafana)

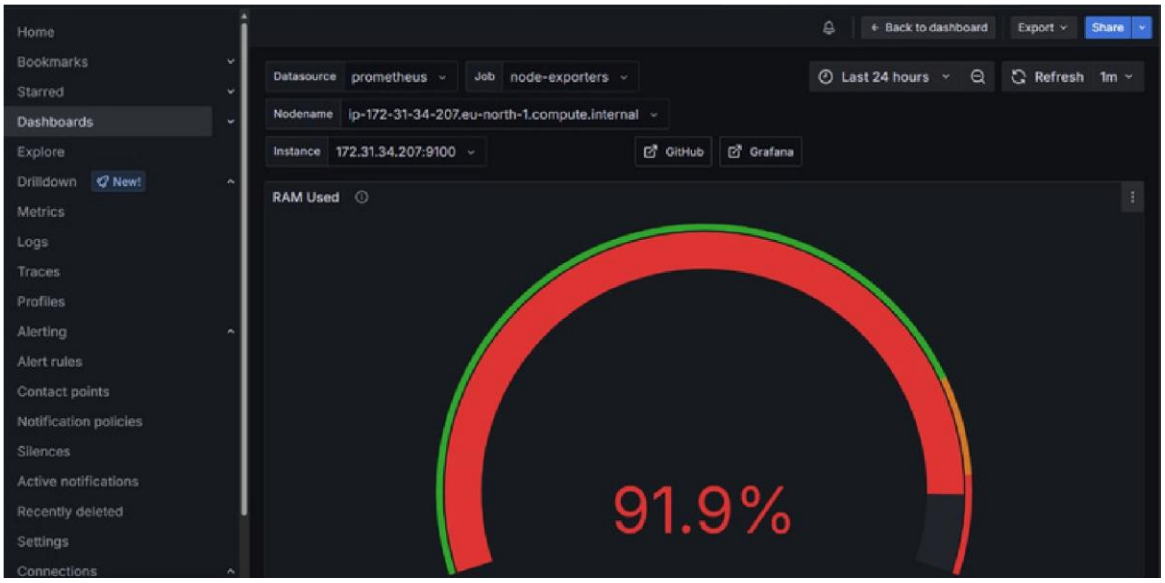
Monitoring was achieved using Prometheus to scrape metrics from EC2 instances via Node Exporter, and Grafana for visualization.

Dashboards were created to monitor CPU, memory, and disk usage

Prometheus configuration included jobs for node-exporter, and Grafana was connected to Prometheus as a data source





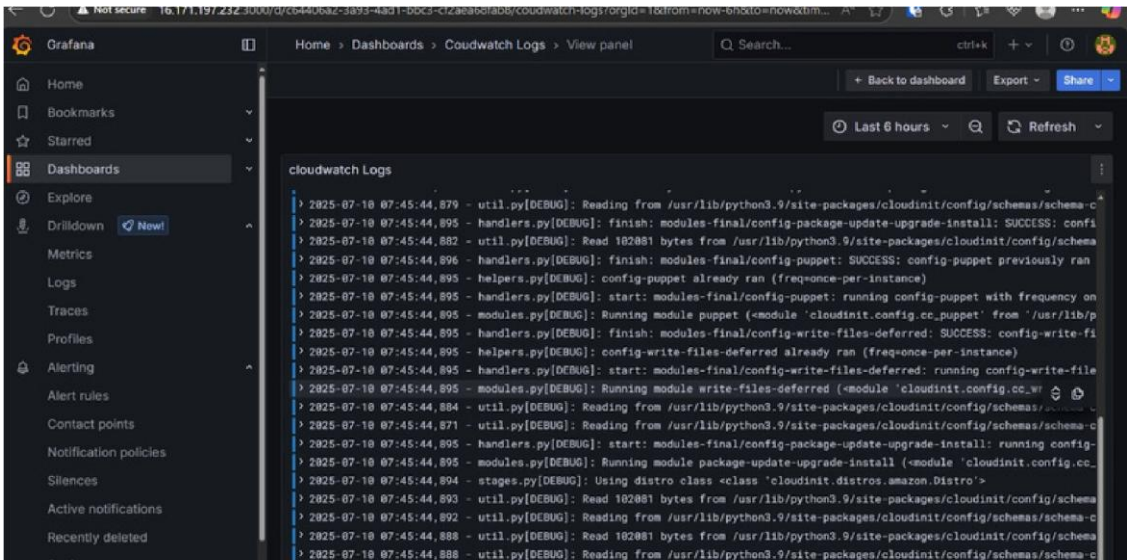


2. Centralized Logging (CloudWatch Logs)

CloudWatch Agent was installed and configured on EC2 to push logs (e.g., cloud-init.log) to AWS CloudWatch.

The log group was integrated with Grafana via the CloudWatch data source to visualize logs directly in Grafana dashboards.

IAM roles with CloudWatchAgentServerPolicy were attached for permission to publish logs.



3. Alerting Configuration (Grafana Unified Alerting + AWS SNS)

Grafana alert rules were created for CPU, memory, and disk usage thresholds.

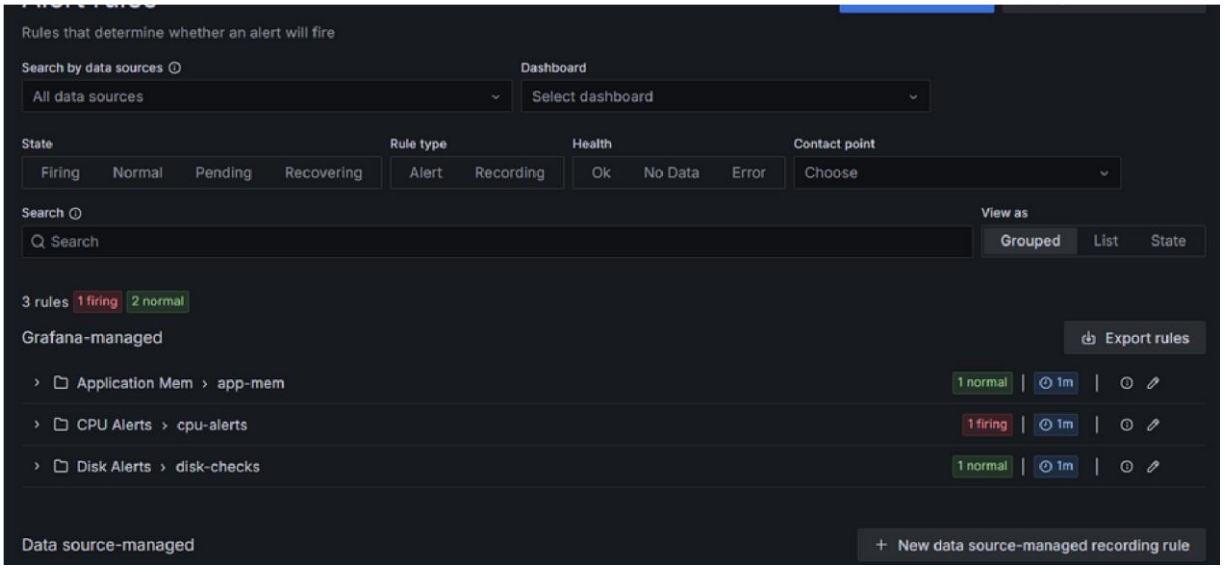
When thresholds were breached, alerts were triggered and routed through SNS to email notifications.

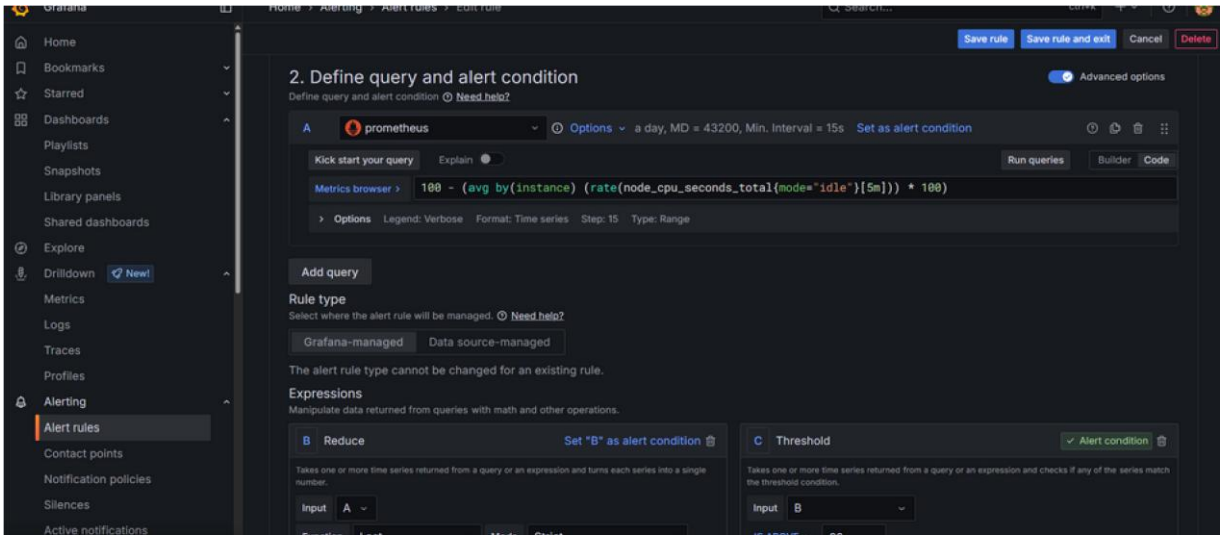
A dedicated SNS topic was created, subscribed via email, and confirmed by the user.

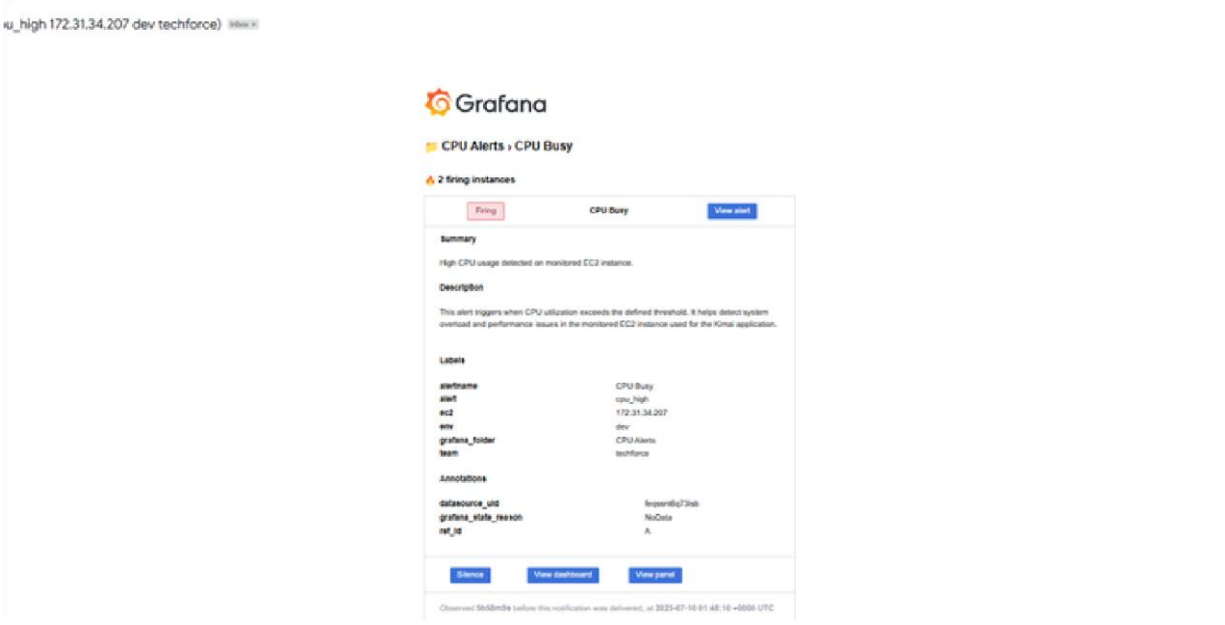
Alerts such as CPU High and Memory Low were tested and triggered successfully.

This monitoring and alerting system strengthens operational excellence and reliability of the deployed infrastructure.

It provides real-time system awareness, immediate alerting via email, and complete log traceability via CloudWatch.







Lessons Learned

Grafana alert queries require careful configuration (e.g., "Range" type, step settings).

Alert testability improves with mock queries like `vector(100)`.

SNS integration can be reused across alerts and was effective for email-based alerts.

CloudWatch Logs are powerful for log retention, especially when visualized in Grafana.