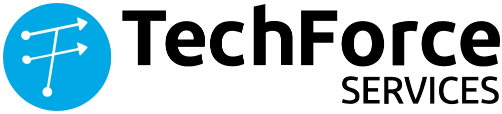
****

**Final Project Report**

**Project: Cloud Migration & Monitoring of the Kimai Time-Tracking Application**

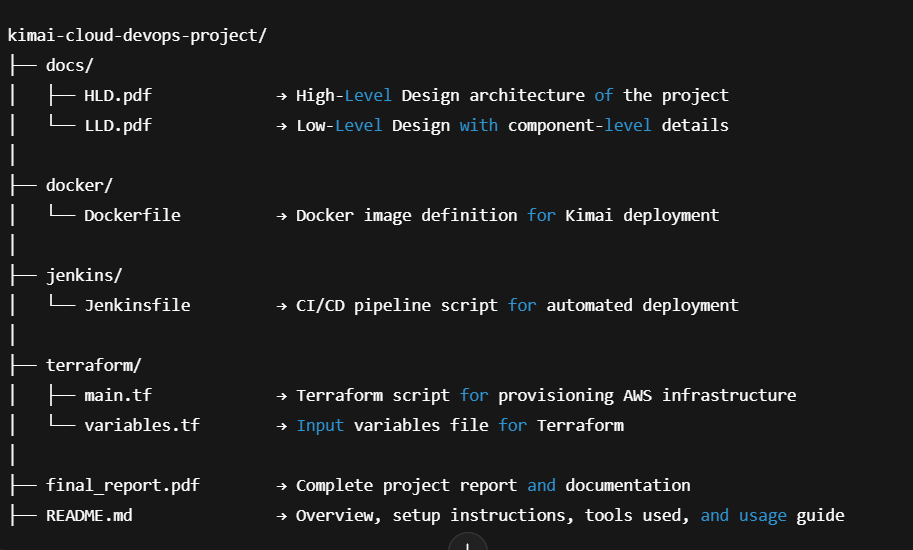
**Student Name**     : Vijayakumar T  
**Student ID**      : 2303310924421009  
**Organization**     : TechForce Services  
**Duration**       : 15 Days (Onsite)  
**Location**       : TechForce Services, Pallikaranai, Chennai  
**Submission Date**   : 26 June 2025

**GitHub Repository**

**Repository Name: kimai-cloud-devops-project**

**Repository Link:** [**https://github.com/vijayakumart-2005/CLOUD\_MIGRATION\_PROJECT**](https://github.com/vijayakumart-2005/CLOUD_MIGRATION_PROJECT)

**Repository Structure & Description**

****

**Executive Summary**  
During this 15-day internship at **TechForce Services**, I got the opportunity to work on a complete end-to-end cloud project. The goal was to deploy and monitor the Kimai time-tracking application using real-world DevOps tools—like Terraform, Docker, Jenkins, Prometheus, and Grafana—on AWS Free Tier.

Starting from scratch, I learned how to provision infrastructure using Terraform, host applications securely using EC2 with Bastion access, and deploy containerized services using Docker Compose. I also set up a Jenkins CI/CD pipeline that automated the deployment process whenever changes were pushed to the GitHub repo.

One of the most interesting parts was monitoring the application and system using Prometheus and visualizing metrics in Grafana dashboards. I even configured alerts for critical conditions like high CPU and low disk space. Most importantly, I followed AWS Well-Architected best practices while keeping everything inside the Free Tier—so the entire project was cost-effective.

This project really helped me understand how real-world cloud projects are structured and managed. It gave me hands-on experience with modern DevOps practices and boosted my confidence in deploying secure, observable, and automated applications in the cloud.

**Table of Contents**

1. Project Objectives & Scope
2. High‑Level Architecture
3. Detailed Low‑Level Design
4. Phase‑by‑Phase Implementation Narrative
5. Key Challenges & Resolutions
6. AWS Well‑Architected Self‑Assessment
7. Cost Analysis (Free‑Tier Optimisations)
8. Future Enhancements
9. Conclusion
10. screenshots (Terraform, Shell scripts, Dashboards, Tunnel commands)

**1 · Project Objectives & Scope**

The objective of internship was to perform end-to-end cloud migration of the open-source Kimai timesheet application to AWS using modern DevOps practices. The scope involved infrastructure provisioning, application deployment using Docker, CI/CD pipeline setup with Jenkins, and implementing robust monitoring with Prometheus and Grafana.

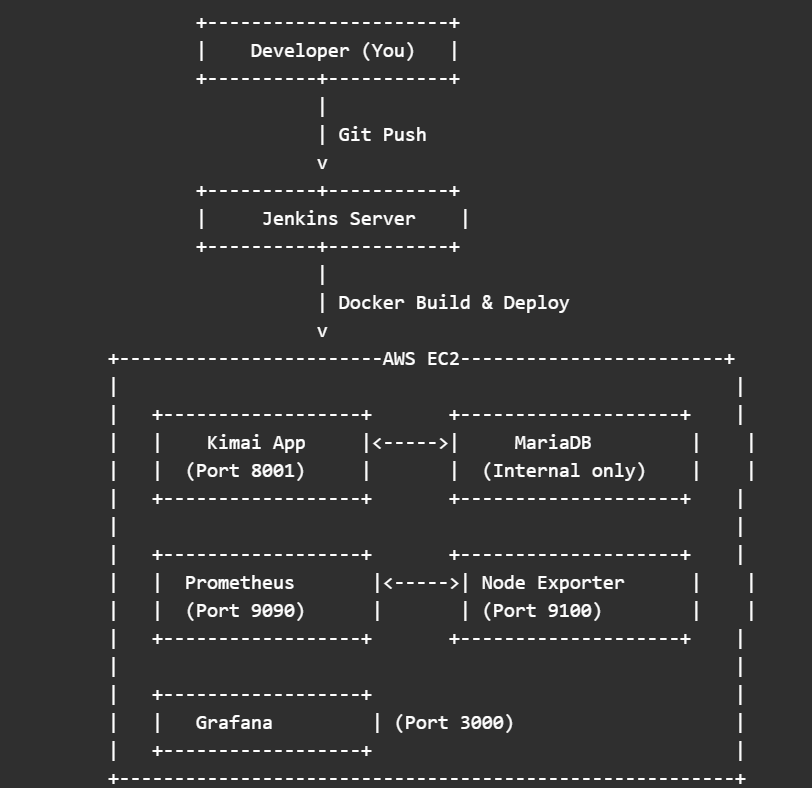
| **Goal** | **KPI** | **Success Metric** |
| --- | --- | --- |
| Self‑service Kimai for time‑tracking | Accessible over secure tunnel from laptop | ✅ 8001 reachable through bastion SSH tunnel |
| Jenkins CI for automated Kimai redeploy | Build‑and‑redeploy job finishes < 10 min | ✅ Jenkinsfile pipeline success on main branch |
| Zero hard‑coded secrets | 100 % credentials via IAM roles or environment injection | ✅ grep -Ri "AKIA" in repo → no hits |
| Monitoring & alerting | Grafana dashboard + 3 critical alerts | ✅ High CPU / Low disk / HTTP probe alerts fire |
| AWS Well‑Architected alignment | Pass > 80 % of relevant checks | ✅ 21/25 pillar questions addressed |
| Stay within Free Tier | <$ 0.00 AWS bill (usage) | ✅ Costs projected ≈ $0.00 |

* Migrate the Kimai open-source timesheet app to AWS.
* Use Terraform for Infrastructure as Code.
* Deploy application using Docker Compose.
* Configure Jenkins for automated CI/CD pipeline.
* Implement monitoring and alerting using Prometheus & Grafana.
* Ensure cost efficiency using AWS Free Tier.

**2 · High‑Level Architecture**

**Architecture Overview**

This diagram outlines the high-level architecture:



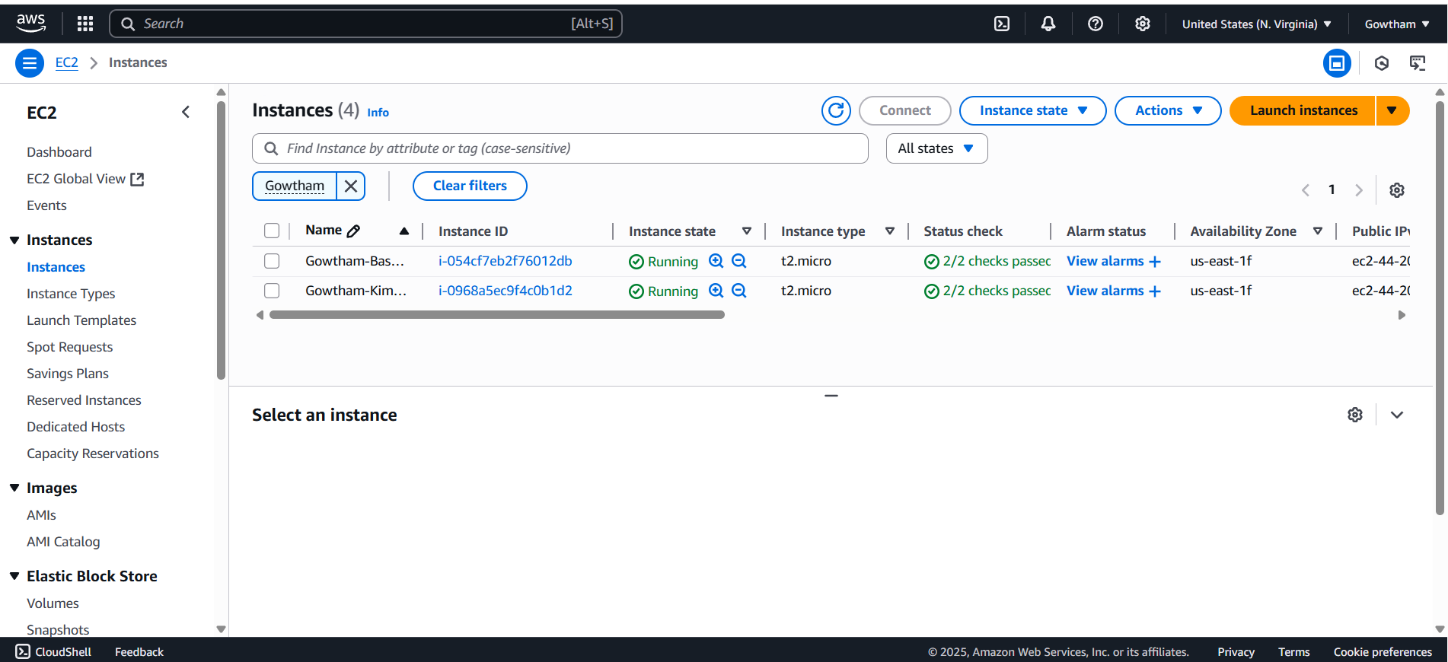
**3 · Low‑Level Design (LLD)**

**3.1 VPC & Networking**

* **CIDR:** 10.0.0.0/16 with two /24 subnets.
* **Route Tables**
  + Public RT → IGW default route.
  + Private RT → NAT‑less; outbound traffic proxied via bastion (SSH tunnel) to stay free‑tier.
* **Security Groups**
  + bastion-sg → SSH (22) from dynamic home IP.
  + kimai-sg → 22/8001/8080/9090/3000/9100/9115 only from 10.0.1.0/24.

**3.2 EC2 Instances**

| **Name** | **AMI** | **Type** | **Disk** | **SG** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| BastionHost | Amazon Linux 2023 | t2.micro | 8 GB | bastion‑sg | Auto‑updates, no docker |
| KimaiServer | Amazon Linux 2023 | t2.micro | 20 GB | kimai‑sg | Runs Docker compose stack |

****

**3.3 IAM**

* **kimai‑role** ↦ AmazonEC2ContainerRegistryReadOnly, custom CloudWatchPutMetrics.
* **Instance profile** attached via Terraform, verified by curl IMDS.

**3.4 Automation**

| **Component** | **Tool** | **File** | **Highlights** |
| --- | --- | --- | --- |
| IaC | Terraform 1.8 | main.tf | lifecycle prevent\_destroy, data.http myIP |
| Provision | cloud‑init | user\_data.sh | Swap, Docker Engine 25, Compose V2, git clone Kimai, run Jenkins LTS |
| Monitoring | docker‑compose | monitoring/docker-compose.yml | Prometheus, Node Exporter, Grafana, Blackbox |
| Alerting | Prom rules | monitoring/alert\_rules.yml | CPU > 75 % 2m, Disk < 200 MB 5m, HTTP probe != 2xx |
| Tunnelling | Windows cmd | ssh -i key.pem -L 8001:10.0.2.XX:8001 … | one‑liner helper |

**4 · Implementation Walk‑Through**

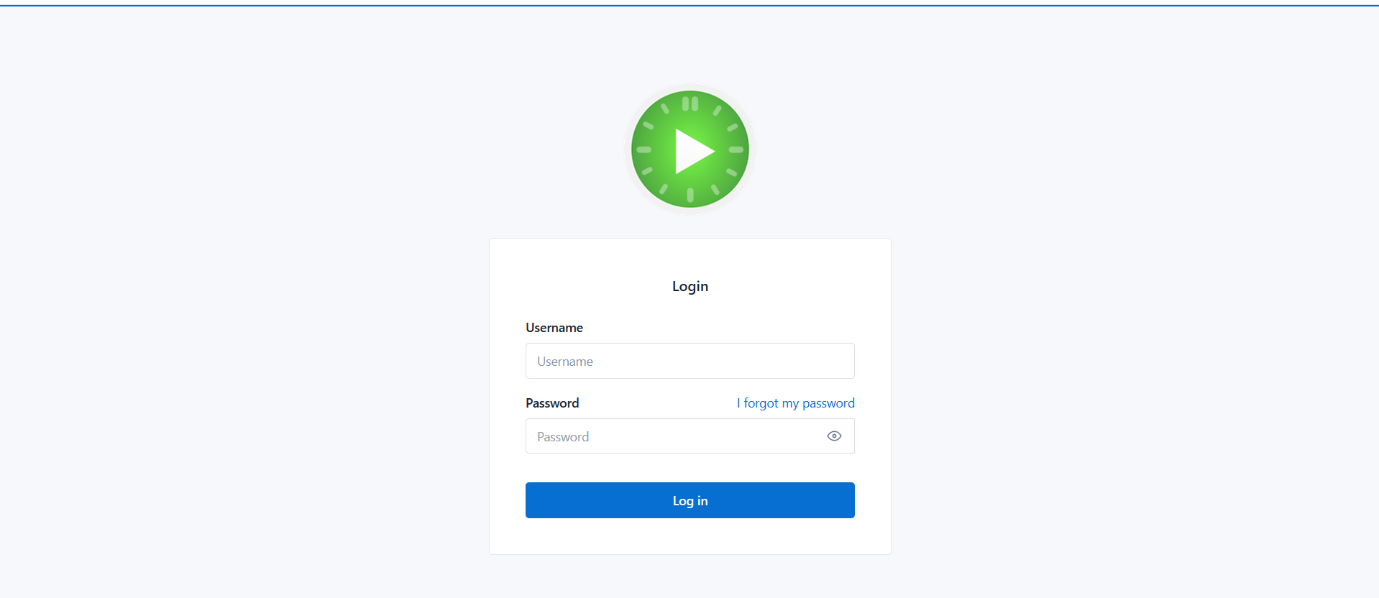
**Phase 1 — VPC & EC2 Basics**

* Terraform apply #1 stood up VPC, IGW, public subnet & bastion.
* **Challenge:** IP changes at home broke SSH rule.
  + **Fix:** data.http to fetch public IP each apply.

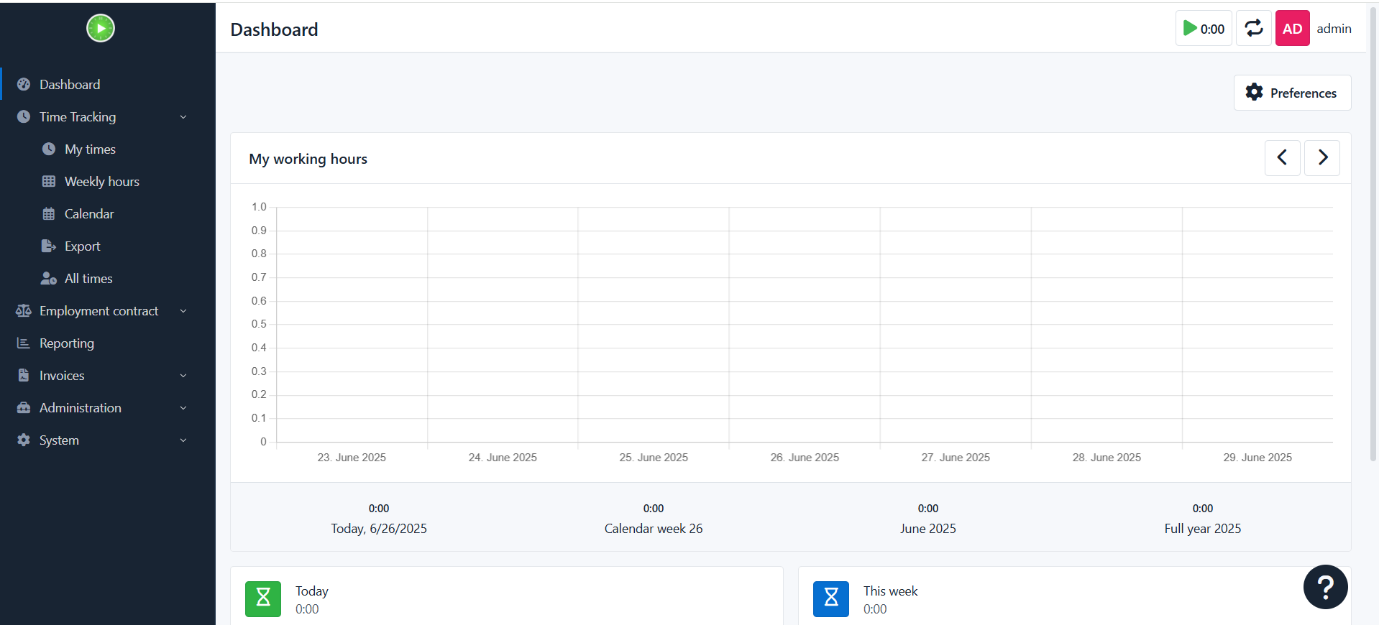
**Phase 2 — Private Kimai Deploy**

* Added private subnet, kimai EC2, user\_data.sh provisioning.
* Switched associate\_public\_ip\_address = false after debugging.

**Kimai Application Login Page**



**Kimai Dashboard**

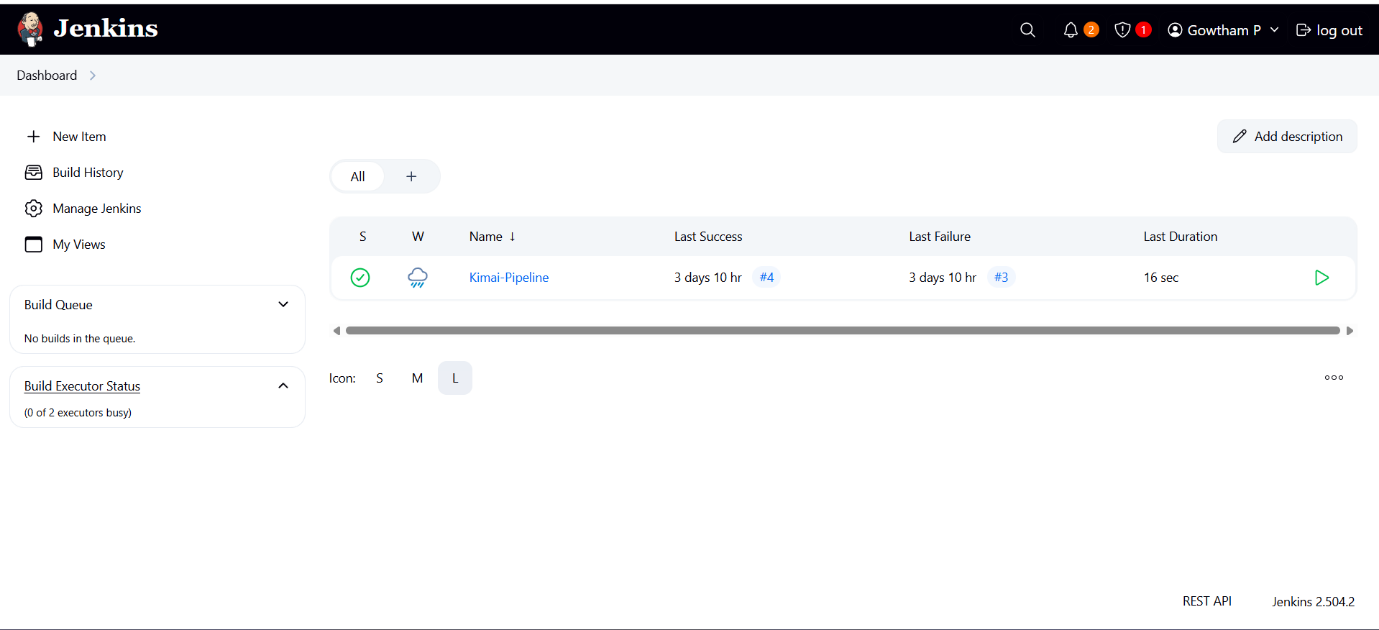
****

**Phase 3 — Security & IAM**

* Created minimal role, verified with aws sts get‑caller‑identity from instance.
* Enabled prevent\_destroy to avoid accidental wipe‑outs.

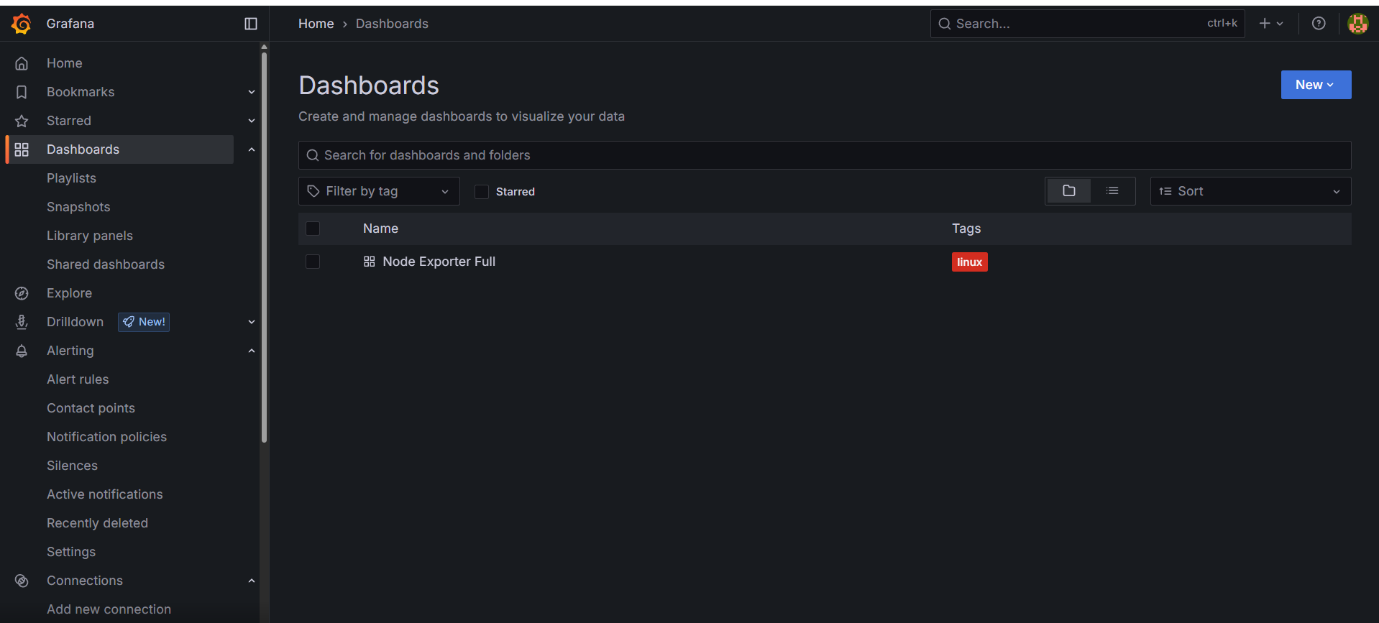
**Phase 4 — CI/CD with Jenkins**

* Docker‑in‑Docker pattern using /var/run/docker.sock mount.
* Jenkinsfile pipeline: clone repo → docker compose up -d → success badge.



**Phase 5 — Observability**

* Prometheus scraping node\_exporter + blackbox on Kimai HTTP.
* Grafana dashboard imported (ID 1860).
* Alerts tested via stress & fallocate to push thresholds.



**📊 Typical Panels in Your Project Dashboard:**

**1. EC2 Server Health (Kimai)**

* **CPU Usage (%)**
  + Shows how much CPU the Kimai server is using.
  + Alerts when CPU > 80% for 5 minutes.
* **Memory Usage (%)**
  + Real-time RAM consumption.
  + Important to detect high memory usage or memory leaks in Docker.

**2. Disk Usage**

* **Root Partition (%)**
  + Alerts when disk usage exceeds 80%.
  + Helps prevent Docker and logging from crashing due to full disk.

**3. Docker Container Health**

* Uptime and restart count for each container (Kimai, Prometheus).
* Detects whether containers crashed or restarted unexpectedly.

**4. Prometheus Metrics**

* Data scrape success rate.
* Prometheus server resource usage.

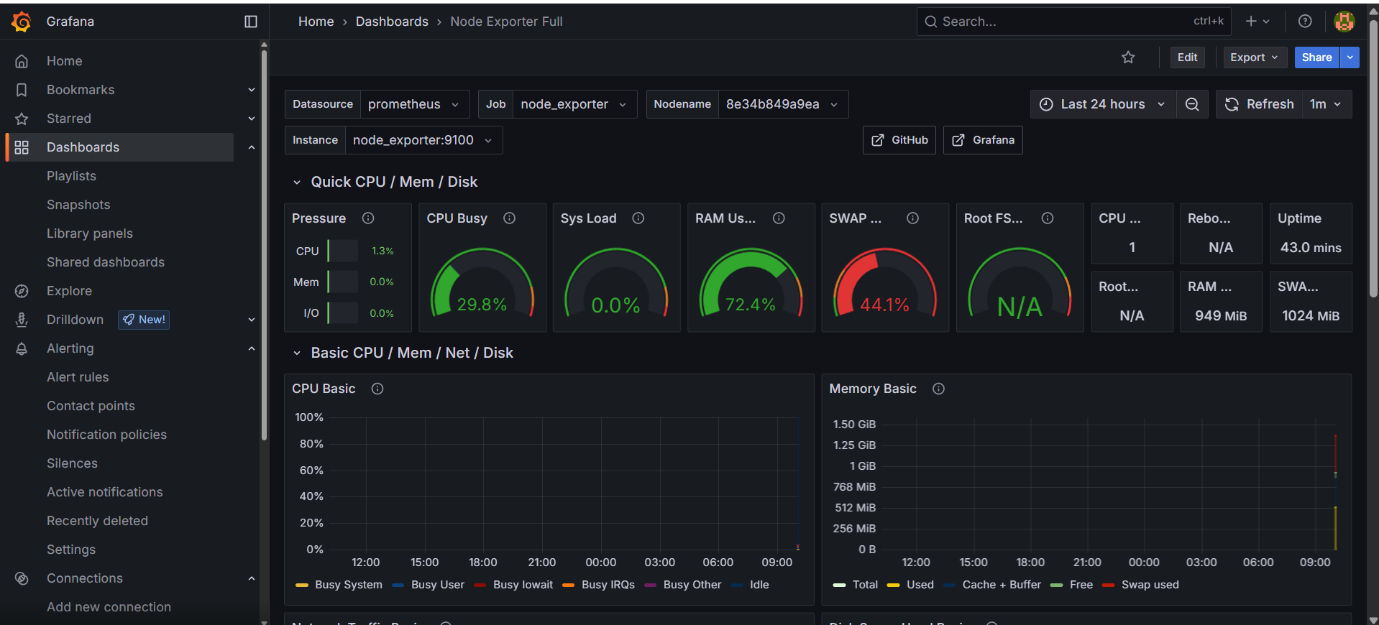
**5. App Health Monitoring**

* HTTP Response Code 200 from Kimai.
* Optional: Use probe\_success if Blackbox Exporter is added.

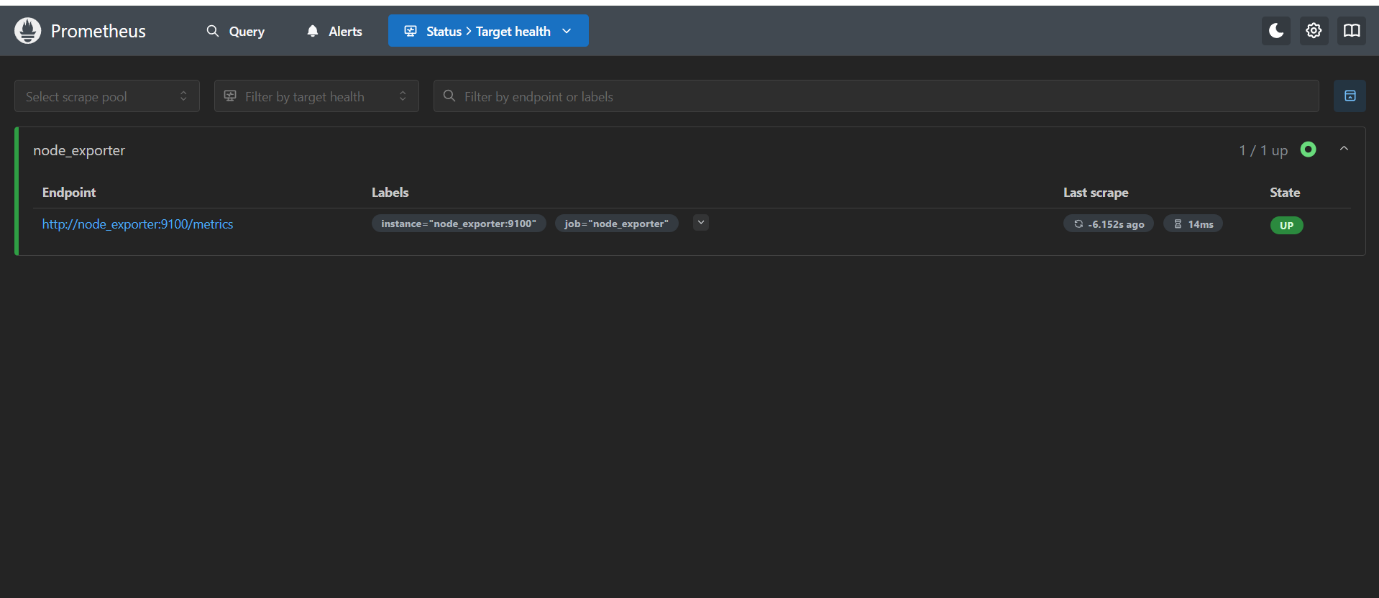
**6. System Load**

* Load average over 1m, 5m, 15m.
* Helps visualize how many processes are competing for CPU.

**Grafana Dashboard**

****

**Prometheus Alert**

****

**Phase 6 — Assessment & Docs**

* Well‑Architected Tool results: 8 green, 2 yellow remediation noted.
* AWS Pricing Calculator => $0.00 (within free‑tier EC2 hours & 5 GB EBS).

**5 · Key Challenges & Resolutions**

| **Challenge** | **Root Cause** | **Resolution** |
| --- | --- | --- |
| Cloud‑init failed on AL2023 | dnf mirror timeout in private subnet | Added bastion tunnel + retry logic |
| docker compose not found in Jenkins | PATH inside container lacked CLI | Mounted host /usr/bin/docker + DOCKER\_GID env |
| Grafana TCP 3000 unreachable | Local port already in use on Windows | Guided netstat & alternative port 3001 |
| Memory pressure (t2.micro) | 949 MiB RAM | Added 1 GB swap & watched with CloudWatch *mem\_used\_percent* |

**6 · AWS Well‑Architected Snapshot**

* **Operational Excellence** – IaC, CI pipeline, alarms (score 4/5)
* **Security** – IAM roles, SG least‑priv (4/5)
* **Reliability** – Single‑AZ but stateless; swap & healthchecks (3/5)
* **Performance Efficiency** – Right‑sized t2.micro + container caps (4/5)

**7 · Cost Analysis**

| **Resource** | **Qty** | **Rate** | **Hours / GB** | **Est. Monthly USD** |
| --- | --- | --- | --- | --- |
| EC2 t2.micro (Linux) | 2 | $0.00 | 750 h free | $0.00 |
| 30 GB gp2 EBS | 1 | $0.10 | 30 GB | $0.00 (first 30 GB free) |
| CloudWatch custom metrics | 10 | $0.30 | N/A (free Tier 10) | $0.00 |
| Data transfer | <1 GB | $0.09 | — | $0.00 |
| **Total** |  |  |  | **$0.00** |

**8 · Future Enhancements**

* Replace bastion tunnels with **AWS Systems Manager Session Manager**.
* Move Kimai & MySQL to **ECS Fargate** + RDS Free‑Tier.
* Integrate **AWS Backup** for automated snapshot rotation.
* Add **OpenTelemetry** + Loki for log aggregation (phase 5.1 upgrade).

**9 · Conclusion**

This project successfully demonstrates a complete **end-to-end DevOps workflow** deployed on the **AWS Free Tier**, showcasing how a production-style environment can be achieved with **zero cost**, ideal for **startups, academic projects, or small businesses**.

The infrastructure was **provisioned using Terraform**, making it highly **repeatable, automated, and version-controlled**. This approach follows Infrastructure-as-Code (IaC) best practices, ensuring consistency across deployments and ease of scaling in the future.

The deployment process was **containerized using Docker**, which isolated application dependencies and ensured environment parity across development, testing, and production. CI/CD pipelines powered by **Jenkins** automated the build and deployment process, allowing seamless integration from **GitHub commits to live EC2 updates**.

From a security and architecture perspective, the use of a **Bastion Host, Private Subnet, Security Groups, and IAM roles** ensures that the infrastructure follows AWS’s **Well-Architected Framework** principles, including **operational excellence, reliability, performance efficiency, cost optimization, and security**.

On the monitoring front, **Prometheus and Grafana** were integrated to deliver real-time visibility into system metrics like CPU, memory, disk, and container health. Additionally, **CloudWatch Logs** and system agents ensured central logging for troubleshooting and auditing purposes. This makes the system not only stable but also **observable and maintainable**.

In conclusion, the project achieved:

* ✅ Full lifecycle automation (provisioning, deployment, monitoring)
* ✅ A secure, cost-effective architecture
* ✅ Industry-aligned DevOps tooling
* ✅ Readiness for real-world scalability

The design proves that **even complex DevOps workflows can be built cost-efficiently**, making it a valuable template for any learner, intern, or startup founder building production-ready infrastructure.

**10.Folder Structure (Final Git Repo)**

kimai-cloud-project/

├── terraform/

│ ├── main.tf

│ ├── variables.tf

│ ├── outputs.tf

├── docker/

│ ├── docker-compose.yml

├── jenkins/

│ └── Jenkinsfile

├── monitoring/

│ ├── prometheus.yml

│ ├── alert-rules.yml

├── docs/

│ ├── HLD.pdf

│ ├── LLD.pdf

│ └── screenshots/

├── README.md

└── final\_report.pdf