AFM DevSecOps Project

End-to-End DevSecOps Project .

Swapnil Gavhale, DevOps/DevSecOps Engineer

Tech Stack: AWS (EC2, S3, IAM, ECR, RDS) · Terraform · GitLab CI/CD · Docker · Amazon EKS · SonarQube · Trivy · OWASP ZAP · Prometheus · Grafana · CloudWatch

GitHub

Click here

Executive Summary

AFM is a **constraint-driven DevSecOps portfolio project** that demonstrates how real production platforms are built, secured, and operated. It goes beyond mere tool installation by deliberately operating under real-world limitations.

Unlike typical demo projects, AFM navigates significant constraints, including:

- Limited budget
- No DNS or HTTPS
- Single-node Kubernetes cluster
- Real CI/CD failures
- Resource pressure and recovery scenarios

The project covers essential aspects of modern platform delivery:

- Infrastructure as Code with Terraform
- CI/CD-driven platform provisioning
- Secure application delivery (SAST, SCA, DAST)
- Kubernetes deployment on Amazon EKS
- Observability and alerting
- Cost-aware design decisions

AFM exemplifies how **real DevOps and platform teams work effectively under practical constraints**—not just ideal conditions.

Problem Statement & Motivation

Problem Statement

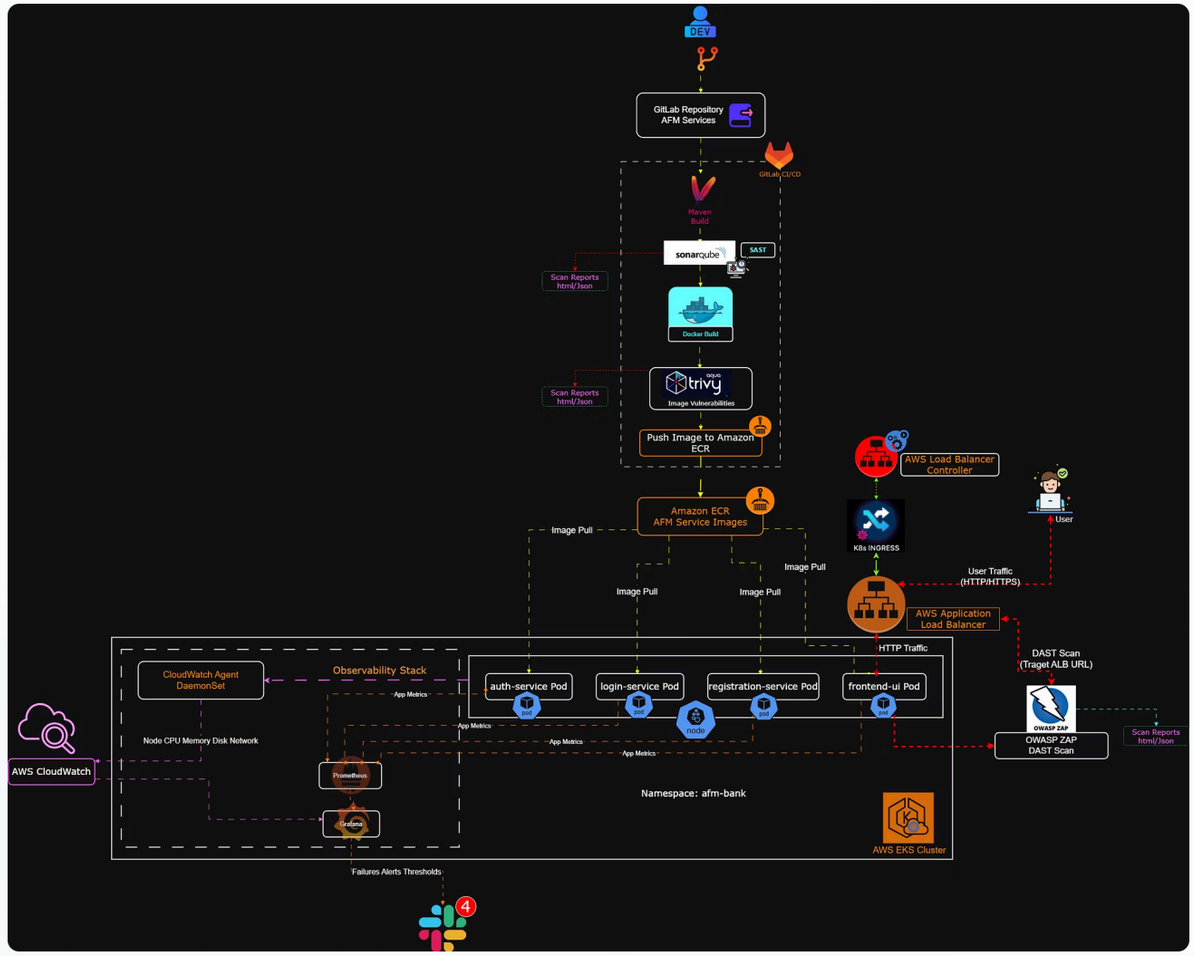
While many DevOps projects demonstrate ideal scenarios and tool functionality, real-world engineers face complex challenges: designing under constraints, making trade-offs, handling failures, and balancing cost, security, and reliability in imperfect systems.

Motivation Behind AFM

AFM was created to simulate these enterprise DevOps challenges by building systems incrementally, documenting critical decisions, and embracing learning from failures.

The guiding philosophy: **Start small → Evolve → Migrate → Secure → Observe → Optimize**

High-Level Platform Architecture



Explanation

- GitLab CI/CD: Orchestrates builds, security scans, and deployments.
- Amazon EKS: Hosts AFM microservices.
- AWS ALB: Exposes the frontend via Ingress.
- Security & Observability: Layers surrounding the runtime environment.

AFM DevSecOps Flow (Lifecycle)

Code Commit → Maven Build → SonarQube (SAST) → Docker Build → Trivy (Image Scan) → Push to Amazon ECR → Deploy to Amazon EKS → OWASP ZAP (DAST) → Monitor & Alert

Why This Matters

- Shift-Left Security: Enforced before & after deployment
- Immutable Images
- Explicit Runtime Vulnerability Testing
- Real-time Behavior Monitoring

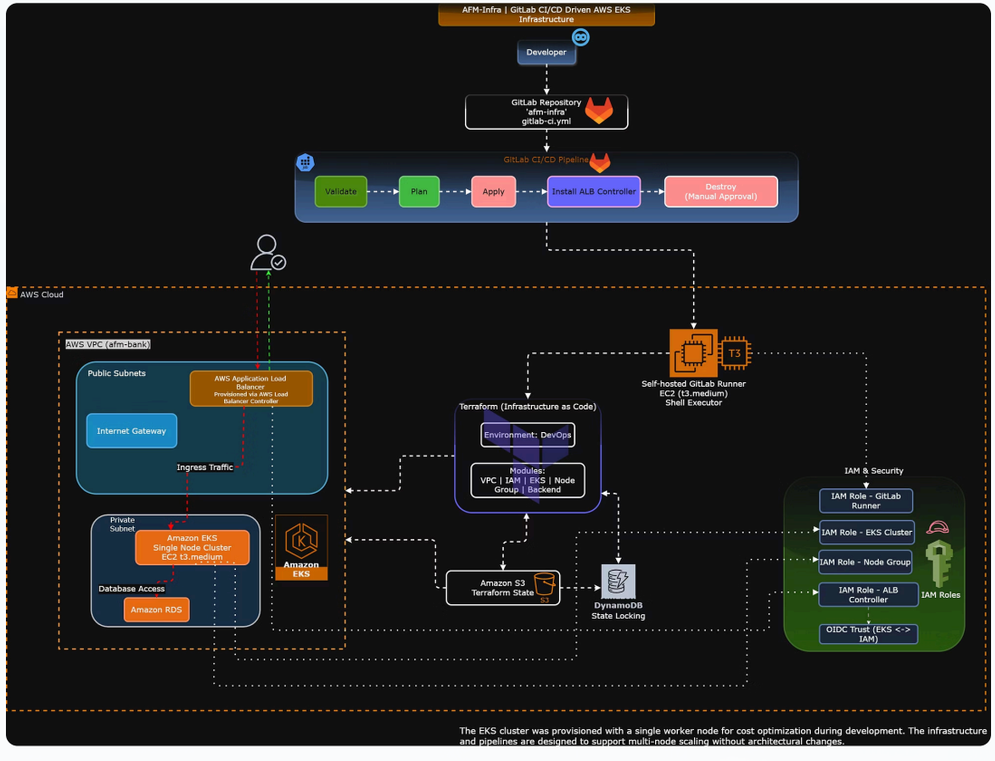
This mirrors **production DevSecOps pipelines**.

Infrastructure Automation (EKS Provisioning via Terraform)

Infrastructure as Code Strategy

Infrastructure is managed exclusively via Terraform, executed through GitLab CI/CD.

- Remote state (S3)
- State locking (DynamoDB)
- Modular design
- Manual approval for apply and destroy
- No local Terraform execution



Kubernetes Design & Constraints

EKS Cluster Design

- Single-node EKS cluster (t3.medium: 2 vCPU, 4 GB RAM)
- Managed node group with cost-aware configuration

Real Constraint Observed

The single-node cluster reliably scheduled ~15–17 pods (Kubernetes system, AFM application, and observability components) before reaching resource limits.

Resolution

Addressed by reducing replica counts, tuning resource requests, and optimizing observability.

> Single-node EKS is harder, not easier—and that was intentional.

Security Strategy (DevSecOps)

Layered Security Approach

Layer	Tool	Purpose
SAST	SonarQube	Code-level issues
SCA	Trivy	Image vulnerabilities
DAST	OWASP ZAP	Runtime testing

OWASP ZAP Design

- ZAP targets the **ALB URL**
- Runs **outside the cluster**
- Tests the live application
- Not part of production traffic

This reflects **real DAST usage**.

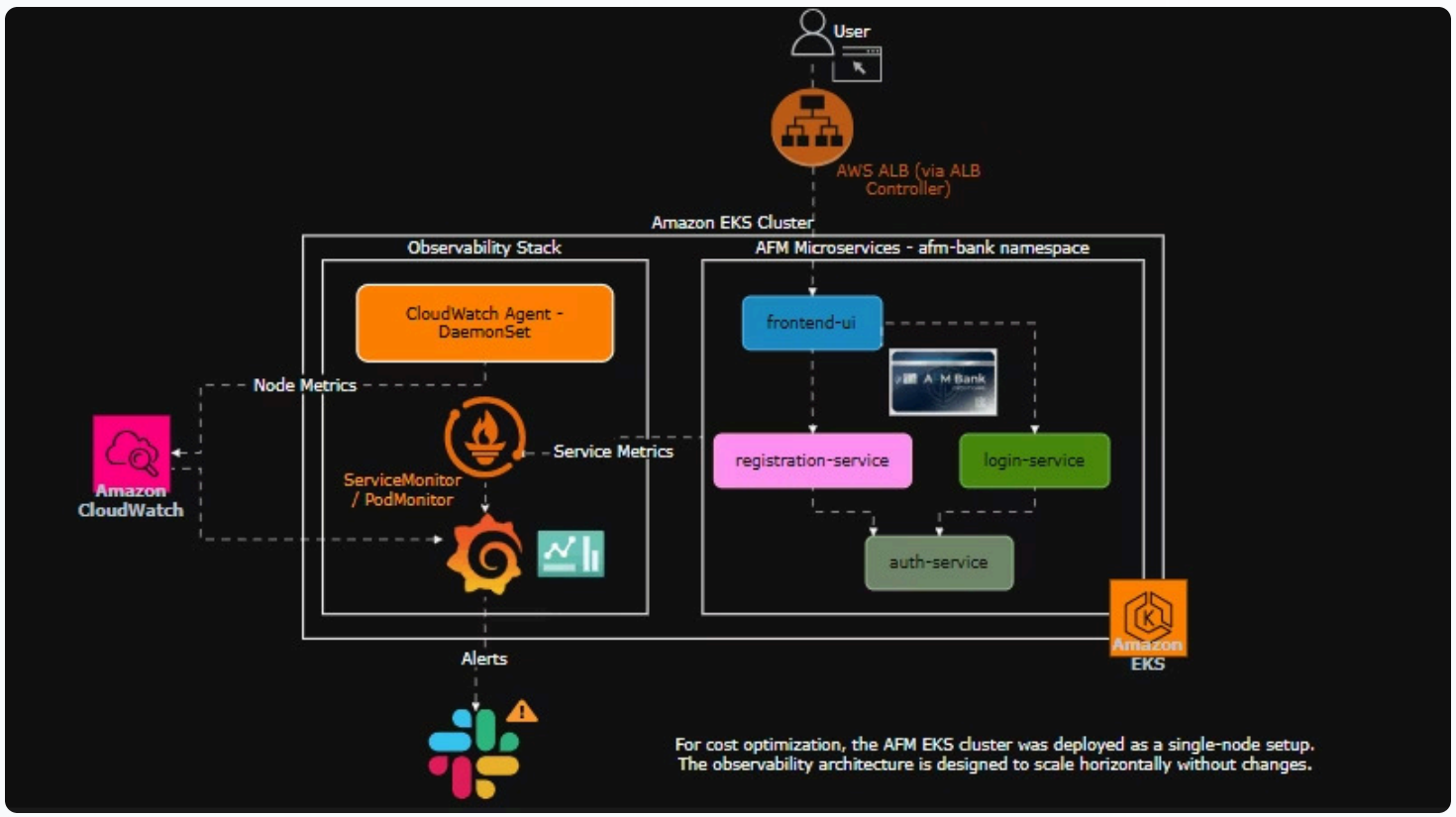
Observability & Monitoring

Observability Stack

- **Prometheus** – Application & service metrics
- **CloudWatch Agent (DaemonSet)** – Node metrics
- **Grafana** – Unified dashboards
- **Slack** – Alert notifications

Diagram to place on this page:

👉 **AFM Observability Architecture Diagram**



Key Learning

Initial observability deployment caused scheduling pressure.

The system was stabilized through tuning and resource optimization.

DNS, HTTPS & Secrets (Design Decisions)

Why No DNS or HTTPS?

Due to the absence of a custom domain, ACM's DNS ownership requirement, and ALB's lack of support for self-signed certificates, HTTPS was intentionally deferred. The platform currently operates over HTTP.

Secrets Strategy

Secrets are managed securely using GitLab CI/CD protected variables, with AWS Secrets Manager integration planned for when HTTPS and IRSA are implemented.

Failures & Learnings

Real Issues Faced

- Runner disk exhaustion
- Pod scheduling failures
- Observability overhead
- Resource contention

What Was Learned

- Capacity planning matters
- Defaults are rarely safe
- Observability has a cost
- Automation must be controlled

Failures were **documented, fixed, and learned from**.

Key Skills Demonstrated

- Terraform (AWS IaC)
- GitLab CI/CD
- Kubernetes (EKS)
- DevSecOps pipelines
- Cloud networking
- Observability & alerting
- Cost-aware engineering
- Failure handling & recovery

Final Takeaway

AFM demonstrates how **real DevOps platforms are designed, secured, and operated under constraints**

This portfolio reflects:

- Production thinking
- Ownership
- Engineering maturity
- Readiness for real DevOps teams