# 🛡️ DevSecOps Assessment Report: Secure Microservice Deployment

## 📌 Project Overview

This project demonstrates the end-to-end implementation of a **secure Node.js microservice with MongoDB**, incorporating **DevSecOps best practices** across infrastructure, containers, application code, and CI/CD workflows.

The solution is split into:

- 🔧 **Terraform Infrastructure on AWS** for production-grade deployments
- 🐳 **Docker Compose Microservice** for local development and security testing
- 🔄 **GitHub Actions CI/CD pipeline** enforcing automated security checks before deployment

---

## 🚀 Infrastructure-as-Code (IaC) – Terraform on AWS

The `terraform/` directory deploys:

- Amazon EKS (Kubernetes Cluster)

- AWS Application Load Balancer (via Ingress)

- Amazon DocumentDB (MongoDB-compatible)

- AWS Secrets Manager (for Mongo URI)

- Amazon EFS (persistent shared storage)

- AWS WAF, IAM, Route 53, and other security constructs

### ✅ Security Features:

- Secrets pulled securely from **AWS Secrets Manager**
- **No hardcoded credentials**

- WAF + security groups to isolate access
- Terraform layout supports **Checkov** scans

### 🔧 Deployment Instructions: Please refer [README.md](README.md) file added in terraform directory.

---

## 🧪 Secure Microservice – Node.js + MongoDB

A hardened microservice with the following DevSecOps principles:

### 🔐 Container Security:

- Built using **Alpine Linux** + **multi-stage Docker builds**
- **Non-root user (appuser)** with minimal capabilities
- Read-only root filesystem + `tmpfs` mount
- Resource limits + `no-new-privileges: true` + dropped capabilities

### 🐳 Run Locally:

```
git clone <repo-url>
cd secure-microservice-demo/app
docker-compose up --build
```

### ✅Prerequisities:

Before deploying this project, ensure the following tools, services, and configurations are in place:

🛠️ **Tooling**

- Docker: For building and pushing container images
- AWS CLI v2: For authentication and infrastructure interaction
- kubectl: To interact with the Kubernetes cluster
- Helm v3: For managing Kubernetes deployments
- Terraform v1.4+: For provisioning AWS infrastructure

☁️ **AWS Requirements**

- An active AWS account
- A configured ECR repository (Docker images are pushed here)
- IAM credentials with sufficient permissions (for ECR, EKS, EFS, ALB, ACM, Route 53)

- S3 bucket dynamodb table created with same name in terraform/environment/prod/backend.tf file
- A domain name managed in Route 53

🔐 **GitHub Actions CI/CD Secrets**
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
-AWS_REGION

### ✅ Test API:

curl http://localhost:3000/health

---

## 🔁 CI/CD – GitHub Actions with Security Gates

The pipeline enforces secure delivery of container images before pushing to Amazon ECR.

### 🧩 Key Stages:

1. **Static Analysis**
    - Semgrep (with GitHub Security tab integration)
    - SonarCloud Scan (quality gate + PR check)
2. **Secrets Detection**
    - TruffleHog (file system & git history)
3. **Container Scanning**
    - Trivy for OS/package CVEs
4. **Security Gate**
    - Fails if any critical issues are found in above scans
5. **Push to ECR**
    - Only if security gate passes
6. **Notifications**
    - Pipeline status logged in final step

### 🔧 Setup Instructions:

1. Store AWS credentials and tokens as GitHub secrets:
   - `AWS_ACCESS_KEY_ID`
   - `AWS_SECRET_ACCESS_KEY`
   - `SONAR_TOKEN`
   - `SEMGREP_APP_TOKEN`
2. Update `env:` in the workflow:
   AWS_ACCOUNT_ID: <your-account-id>
   AWS_REGION: us-east-1
   IMAGE_NAME: secure-microservice
3. CI/CD will trigger on `push` and `pull_request` to `main` or `develop`

### 🛑 Security Gate Logic:

# Fail if any of these fail:
- Semgrep
- Sonarcloud
- Trivy
- Secrets scan

---

## 🔍 Security Tools Used

| Category | Tools |
|---|---|
| Static Analysis | Semgrep, SonarCloud |
| Secrets Detection | TruffleHog |
| Container Scanning | Trivy, Dockle |
| Infrastructure Scanning | Checkov (Terraform) |
| CI/CD Platform | GitHub Actions |
| Container Registry | Amazon ECR |

---

## ✅ Highlights

- 🔐 No hardcoded secrets – fully externalized using AWS Secrets Manager
- 🧱 Hardened container image with rootless runtime and capability drops
- 🚫 Automatic fail gates for vulnerabilities and leaked secrets
- 🧪 Manual security testing steps included in README for validation
- 📦 Clean deployment to ECR after passing quality + security checks

---

## 🧹 Stop Application/container

### Local:

docker-compose down -v
docker rmi $(docker images -q secure-microservice_app)

### Cloud:

cd terraform
terraform destroy -var-file=config/parameters-vmox-demo.tfvars

# 🚀 Quick Start Step
**Step 1: Authenticate AWS CLI :**
aws configure

It will ask you for Your access key,Your secret key, Default region name, Default output format

**Step 2: Deploy AWS Infrastructure**
terraform init
terraform apply

**Step 3: Build & Push Docker Image**

1. Authenticate with AWS ECR -
   aws ecr get-login-password --region <AWS_REGION>
   docker login --username AWS --password-stdin <ECR_REPO>

2. Build the Docker image -
   docker build -t web-app ./web-app/

3. Tag the Docker image -
   docker tag web-app:latest <ECR_REPO>/web-app:latest

4. Push the image to ECR:
   docker push <ECR_REPO>/web-app:latest

**Step 4: Get the EFS File System ID:**
aws efs describe-file-systems --query "FileSystems[*].FileSystemId" --region <AWS_REGION>

**Step 5: Connect to Kubernetes and Deploy**

1. Update kubeconfig for EKS
   aws eks update-kubeconfig --region <AWS_REGION> --name <EKS_CLUSTER_NAME>

   This ensures kubectl and helm commands interact with the right cluster.

2. Verify Connection to Kubernetes kubectl get nodes
3. Deploy the Helm Chart
   helm upgrade --install web-app-release ./helm \
   --namespace web-app --create-namespace \
   --set efs.fileSystemId=<efs-id> \
   --set image.repository=<ECR_REPO>/web-app \
   --set image.tag=latest

## 📊 Monitoring & Alerting

| Tool | Purpose |
|---|---|
| Prometheus | Metrics collection (infra + app) |
| Grafana | Real-time dashboards |
| Alertmanager | Slack/Email/PagerDuty alerting |

**Metrics Monitored**:

- API latency, error rates, request volume
- Pod restarts, CPU/mem usage, disk I/O
- EFS usage, ALB health, ingress latency

---

## 🔍 Manual Security Scanning

### Terraform Scan

```
Checkov scan terraform/
```

---

## ✅ Deliverables

- ✅ Hardened Dockerfile (multi-stage, no root)
- ✅ GitHub Actions pipeline (CI/CD + security gates)
- ✅ AWS Infrastructure via Terraform (EKS, ALB, DocumentDB)
- ✅ Secure secrets management via AWS Secrets Manager
- ✅ Helm charts for Kubernetes deployment
- ✅ Monitoring via Prometheus + Grafana
- ✅ PDF report documenting implementation (optional)
- ✅ This README

---