Project Title

Expert DevOps Implementation: A Zero-Trust, Resilient Self-Hosted Kubernetes Architecture for the "Glossary" Application

Project Objective

The objective is to architect and implement a production-grade, zero-trust, and highly resilient DevOps lifecycle for the "Glossary" application. This project will establish a sophisticated GitOps workflow using a **self-hosted, multi-master Kubernetes cluster**, fortified with a service mesh for advanced security and traffic management. The core focus is on building a fully automated, observable, and self-healing infrastructure on AWS, managed entirely through code, with a **custom Jenkins CI/CD pipeline**. The successful candidate will demonstrate expert-level skills in infrastructure management, cloud-native security, progressive delivery patterns, and resilience engineering.

Technologies and Tools

- **Application Stack:** React.js (Frontend), Node.js/Express (Backend), MongoDB (Database)
- **Containerization:** Docker, Docker Compose
- **Cloud Provider:** Amazon Web Services (AWS)
- **Container Orchestration: Self-Hosted Kubernetes (Multi-Master, Stacked etcd topology)**
- **Infrastructure as Code (IaC):** Terraform (with Remote State, Modules, and Workspaces)
- **CI/CD & GitOps: Jenkins (Self-hosted, Master-Agent configuration)**, Argo CD (for GitOps CD)
- **Progressive Delivery:** Argo Rollouts
- **Service Mesh:** Istio or Linkerd
- **Observability Stack:** Prometheus, Grafana, AWS X-Ray, Fluentd
- **Security & Policy: SonarQube**, Trivy, tfsec, Open Policy Agent (OPA) Gatekeeper, AWS Secrets Manager, IAM
- **Resilience Testing:** Litmus Chaos

Phase I: Zero-Trust Kubernetes Foundation & Advanced IaC (Weeks 1-3)

This phase focuses on building a secure, production-ready, self-hosted Kubernetes environment with a service mesh, all managed by a modular and collaborative IaC setup.

Deadline: 20/09/2025

Week 1: Hardened Containers & Helm Templating

- **Task 1:** Create optimized, multi-stage `Dockerfile`s with a focus on minimal base images and non-root execution.
- **Task 2:** Integrate **Trivy** into a CI pre-check to scan container images for critical vulnerabilities on every pull request.
- **Task 3:** Develop parameterized **Helm charts** for the application services, ensuring configurations can be managed for different environments (dev, staging, prod).
- **Task 4:** Test the Helm charts by deploying the application stack locally using Minikube or a similar lightweight Kubernetes cluster.

Week 2: Modular IaC & Self-Hosted Kubernetes Cluster

- **Task 1:** Structure the Terraform code into reusable modules (VPC, EC2, Security Groups) and implement a **remote state backend with S3 and DynamoDB for state locking**.
- **Task 2:** Write Terraform scripts to provision the core infrastructure: a custom VPC, and the required EC2 instances for **two master nodes** and multiple worker nodes across different Availability Zones.
- **Task 3:** Automate the installation and configuration of a multi-master Kubernetes cluster using `kubeadm`, ensuring a **stacked etcd topology** and high availability for the control plane.
- **Task 4:** Provision and configure a **Service Mesh (Istio or Linkerd)** within the cluster and enable **mutual TLS (mTLS)** to enforce zero-trust communication by default.

Week 3: Ingress, Policy as Code & Initial Deployment

- **Task 1:** Deploy and configure the **AWS Load Balancer Controller** and a service mesh Ingress Gateway to manage all external traffic.
- **Task 2:** Implement **Open Policy Agent (OPA) Gatekeeper** and write several custom policies in Rego. **Examples:** enforce that all deployments must have specific resource limits, disallow services of type `LoadBalancer`, and require specific labels for cost tracking.
- **Task 3:** Manually deploy the application to the cluster, ensuring it integrates with the service mesh and adheres to all OPA policies.
- **Task 4:** Validate end-to-end functionality, including traffic routing through the ingress gateway and mTLS encryption between pods.

Phase II: Progressive Delivery, Observability & Resilience (Weeks 4-6)

This phase automates the deployment lifecycle with advanced release strategies and implements a comprehensive framework for monitoring and validating system resilience.

Deadline: 20/10/2025

Week 4: Implementing the Jenkins & GitOps Pipeline

- **Task 1:** Provision two EC2 instances for a **Jenkins master and a Jenkins agent**. Configure the connection and necessary tooling (Docker, kubectl) on the agent.
- **Task 2:** Create a `Jenkinsfile` in the application repository to define a multi-stage CI pipeline that performs static code analysis with **SonarQube**, runs security scans, builds the Docker image, and pushes it to ECR.
- **Task 3:** Set up and configure **Argo CD** in the Kubernetes cluster to monitor a dedicated Git repository for application manifests.
- **Task 4:** The final stage of the Jenkins pipeline will update the image tag in the manifests repository, which will automatically be detected by Argo CD to trigger the deployment process.

Week 5: Deep Observability & Resilience Engineering

- **Task 1:** Deploy the **kube-prometheus-stack** and configure the service mesh to export detailed metrics (e.g., request volume, success rates, latencies) to Prometheus.
- **Task 2:** Create two distinct, advanced Grafana dashboards: one specifically for visualizing the canary analysis process from Argo Rollouts, and a **second, comprehensive dashboard to visualize general system health metrics exported by Prometheus** (e.g., node resource utilization, pod status, persistent volume usage).
- **Task 3:** Implement **AWS X-Ray** for distributed tracing to monitor the full lifecycle of requests.
- **Task 4:** Install **Litmus Chaos** and design and execute a **Chaos Engineering experiment**. **Example:** simulate a master node failure to prove the control plane's high availability, then use the Grafana dashboards to prove the system remains stable.

Week 6: Optimization, Final Hardening & Documentation

- **Task 1:** Implement a cost optimization strategy by configuring worker node groups to use a mix of **On-Demand and Spot Instances**.
- **Task 2:** Configure the **Horizontal Pod Autoscaler (HPA)** to work with custom metrics exposed by the service mesh for more intelligent scaling.
- **Task 3:** Conduct a final security review of all IAM policies, Kubernetes RBAC, and service mesh authorization policies.
- **Task 4:** Prepare final project documentation, including detailed architecture diagrams illustrating the GitOps flow, service mesh interactions, and the progressive delivery strategy.

Final Deliverables & Submission Guidelines

- **Code & Configuration:** All source code, Terraform modules, Helm charts, Kubernetes manifests, OPA policies, and the `Jenkinsfile`.
- **Git Repositories:** Links to the application source code repository and the separate GitOps manifests repository.
- **Observability & Resilience Artifacts:** Exported JSON models of the custom Grafana dashboards and the YAML manifests for the Litmus Chaos experiment.

- **Comprehensive Documentation:** A `README.md` file containing:
  - A detailed architecture diagram of the entire system.
  - A step-by-step guide to bootstrapping the environment.
  - An explanation of the progressive delivery workflow and the chaos experiment performed.
  - Links to the Grafana dashboards and the public URL of the deployed application.

Submission & Collaboration Guide

- All code and configuration will be managed in two primary GitHub repositories: one for application source code and another for the Kubernetes manifests (the GitOps repo).
- All changes must be submitted via Pull Requests and must pass automated security and policy checks before review.
- The `main` branch of the manifests repository will represent the desired state of the production environment and will be the single source of truth for Argo CD.

Evaluation Criteria

- **Cloud-Native Architecture:** The solution demonstrates an expert understanding of building and managing a self-hosted, highly available Kubernetes cluster, service mesh, and GitOps principles.
- **Automation & IaC:** The entire infrastructure and application lifecycle are managed declaratively through code with a high degree of automation.
- **Progressive Delivery:** The canary release strategy is implemented correctly, is fully automated, and uses metrics analysis for decision-making.
- **Security & Governance:** The system is secure by design, leveraging a zero-trust network model (mTLS), proactive policy enforcement (OPA), and static code analysis.
- **Resilience & Observability:** The candidate can not only monitor the system's health but can also proactively test and prove its resilience to failure through chaos engineering.
- **Problem-Solving & Documentation:** The documentation is professional and thorough, and the overall solution demonstrates an ability to architect and execute a complex, real-world engineering project.