Notes API: DevOps & Cloud-Native Project Report

1. Executive Summary

This project demonstrates a production-grade Cloud-Native workflow for a Node.js REST API. The focus is on automating the lifecycle of an application—from code commit to Kubernetes deployment—using industry-standard tools for CI/CD, GitOps, Observability, and DevSecOps.

Core Technology Stack

- Runtime/Framework: Node.js, Express
- Orchestration: Kubernetes (Minikube), Helm
- CI/CD & GitOps: GitHub Actions, Argo CD
- Security: Trivy (SAST/DAST)
- Observability: Prometheus, Grafana
- Testing: Jest, Supertest

---

2. System Architecture & Workflow

The system employs a microservice-oriented architecture designed for scalability and immutability.

1. Code: Developers push to GitHub.
2. CI: GitHub Actions runs tests, security scans, and builds a Docker image.
3. Registry: The immutable image is pushed to a container registry.
4. CD (GitOps): Argo CD detects changes in the Helm chart and synchronizes the Kubernetes cluster state.
5. Runtime: Kubernetes manages Pods, Services, and Ingress to expose the API.

---

3. Kubernetes & GitOps Implementation

The deployment strategy prioritizes automation and "Infrastructure as Code" (IaC).

- Helm Orchestration: Used to package Kubernetes manifests. It manages environment-specific variables via values.yaml, handling Deployments, Services (NodePort/ClusterIP), and Ingress.
- Argo CD (GitOps): Acts as the "source of truth." It eliminates manual kubectl commands by automatically reconciling the cluster state with the Git repository, ensuring zero configuration drift.
- Resilience: The deployment includes Liveness and Readiness probes, resource limits (CPU/Memory), and runs as a non-root user to minimize the attack surface.

---

4. Observability Stack

A "Three Pillars" approach to observability ensures the system is measurable and debuggable.

- Prometheus & ServiceMonitor: The API uses prom-client to expose custom metrics (request rates, status codes, note counts). A ServiceMonitor allows the Prometheus Operator to auto-discover the pods.
- Grafana Visualization: A custom dashboard provides real-time insights into system health, throughput, and error rates.
- Structured Logging: Logs are emitted in JSON format with unique Trace IDs, allowing for correlation between logs and specific API requests.

---

5. DevSecOps & CI/CD Pipeline

Security is "shifted left" by integrating it directly into the GitHub Actions workflow.

The CI Pipeline Stages:

1. Testing: Executes Jest unit and integration tests (aiming for high coverage).
2. SAST (Static Analysis): Trivy scans the source code and dependencies for known CVEs.
3. Containerization: Multi-stage Docker build creates a slim, production-ready image.
4. DAST (Image Scanning): Trivy scans the final Docker image for OS-level vulnerabilities before it is pushed to the registry.

---

6. Results & Lessons Learned

Project Outcomes

- Full Automation: Zero-touch deployment from git push to production.
- Security-First: Integrated vulnerability scanning prevents insecure code from reaching the cluster.
- High Availability: Kubernetes handles self-healing and load balancing via NGINX Ingress.

Key Technical Challenges

- Ingress Routing: Resolving hostname conflicts and path-based routing within Minikube.
- Metrics Scraping: Configuring the ServiceMonitor to align with the Prometheus Operator's labeling requirements.
- Persistence: Future iterations will replace the current in-memory storage with a persistent database (e.g., PostgreSQL) using K8s PersistentVolumeClaims.

---

7. Conclusion

The Notes API project successfully integrates the core tenets of modern DevOps. By combining GitOps for deployment reliability, Trivy for security, and Prometheus/Grafana for visibility, the project establishes a robust framework suitable for scaling real-world cloud-native applications.