# DEVOPS STRATEGY & PLATFORM STABILITY PLAN

## CASE STUDY – 1

**NAME:** Prasanna N

**ROLL NO:** 727722EUCS129

**CLASS:** CSE-B

## Scenario

As a DevOps engineer at a fast-growing e-commerce company, you are tasked with improving system stability after multiple service outages have negatively impacted both revenue and customer confidence. The executive leadership expects a concrete, actionable plan that ensures high reliability, availability, and performance of the platform while maintaining rapid delivery cycles.

## Objectives

- Reliability: Eliminate single points of failure and reduce mean time to recovery (MTTR) to under 10 minutes.
- Availability: Maintain at least 99.9% uptime across core services, including storefront, cart, and checkout.
- Performance: Ensure product pages load within 2 seconds at the 95th percentile and checkout API latency remains under 400 ms during peak load.
- Deployment Safety: Limit change-related incidents to < 8% and enable automated rollback within 3 minutes.
- Security & Governance: Integrate DevSecOps practices without compromising delivery speed.

## Service Level Indicators (SLIs) & Service Level Objectives (SLOs)

- Availability: Successful responses ÷ total requests → Target SLO: 99.9% monthly.
- Latency: Track p95 latency for product listing and checkout → Target SLO: p95 < 2s.
- Error Rate: Monitor failed transactions and 5xx error responses → Target SLO: < 0.2%.
- Recovery Time: Alert to resolution → Target SLO: MTTR ≤ 10 minutes.
- Deployment Stability: Failed deploys ÷ total deploys → Target SLO: < 8%.

## Recommended Architecture (High-Level)

- Application Layer: Containerized microservices deployed on Kubernetes (EKS/GKE) with autoscaling across multiple Availability Zones.
- Database Layer: Cloud-native managed databases (Amazon Aurora / Cloud SQL) with read replicas, PITR, and automated failover.
- Networking: Stateless web services behind Application Load Balancers; static content served via CDN (CloudFront/Akamai).
- Caching: Redis/Memcached for session storage, query caching, and API response acceleration.
- Messaging & Eventing: Kafka/PubSub for asynchronous workloads.
- Security: Secrets management using Vault/KMS, TLS everywhere, and WAF/DDoS protection at the edge.

## Core DevOps Practices

1) Infrastructure as Code (IaC)

- Use Terraform/Ansible to provision compute, networking, and database resources.
- Enforce peer review of IaC code with policy-as-code tools (OPA/Conftest).

2) Continuous Integration & Delivery (CI/CD)
• CI with GitHub Actions or GitLab CI: build, unit/integration testing, static code analysis.
• CD with Argo Rollouts or Spinnaker: blue-green/canary deployments.
• Automated rollbacks on failure signals.

3) Observability & Monitoring
• Metrics collection using Prometheus + Grafana.
• Distributed tracing via Jaeger/OpenTelemetry.
• Centralized logging via ELK or Loki.

4) Incident Response & Reliability Engineering
• On-call rotation with PagerDuty/Opsgenie.
• Defined runbooks for recurring incidents.
• Chaos engineering drills with Gremlin/Litmus.

5) Security & Compliance (DevSecOps)
• Image scanning with Trivy/Clair.
• Policy enforcement with OPA Gatekeeper.
• Short-lived credentials and secrets vaulting.

6) Performance & Scalability
• Load testing using Locust/k6.
• API rate limiting with NGINX/Envoy.
• Multi-layer caching policies.

7) Disaster Recovery & Business Continuity
• RTO/RPO definitions per service.
• Automated database backups and quarterly restore drills.
• Multi-region failover for checkout/payment services.

## Phased Execution Roadmap

• Phase 1 (Weeks 1–3): Stabilization
- Set up monitoring, alerting, and runbooks.
- Resolve existing single points of failure.
- Establish on-call schedule.

• Phase 2 (Weeks 4–6): Hardening
- Deploy IaC for baseline infrastructure.
- Introduce CI/CD pipelines with automated tests.
- Enable WAF + CDN configuration.

• Phase 3 (Weeks 7–12): Scaling & Optimization
- Roll out GitOps with Argo CD.
- Conduct load testing and chaos experiments.
- Perform disaster recovery drill and document results.

## Risks & Mitigations

• Tool Complexity: Minimize tool sprawl by standardizing on a core stack.
• Cultural Resistance: Provide DevOps training and foster blameless postmortems.
• Cost Growth: Monitor cloud spend and right-size resources.

## Expected Outcomes

- Achieve ≥ 99.9% availability and faster incident resolution.
- Safer deployments with reduced failure rates.
- Increased customer confidence through faster, stable, and secure delivery.
- Improved engineering efficiency by embedding reliability into workflows.

## Conclusion

This plan emphasizes automation, observability, resilient architecture, and security as core principles of DevOps adoption. By executing the roadmap and adhering to defined SLOs, the company can ensure a highly reliable, performant, and secure e-commerce platform that supports growth while protecting both revenue and customer trust.