ShopMicro Production Platform

ShopMicro is a production-oriented, cloud-native microservices platform engineered using modern DevOps and Platform Engineering practices.

The system simulates an e-commerce workload and is deployed using Infrastructure-as-Code, containerized microservices, Kubernetes orchestration, full-stack observability, CI/CD automation, and layered security controls.

The objective of this capstone was not only to deploy an application, but to design, automate, secure, observe, and operate a production-style cloud platform end-to-end.

---

1. Problem Statement & Architecture Summary

Modern microservices systems require automation, observability, scalability, and security by default. Manual infrastructure provisioning, weak visibility, and inconsistent deployment processes introduce operational risk.

ShopMicro addresses this by implementing:

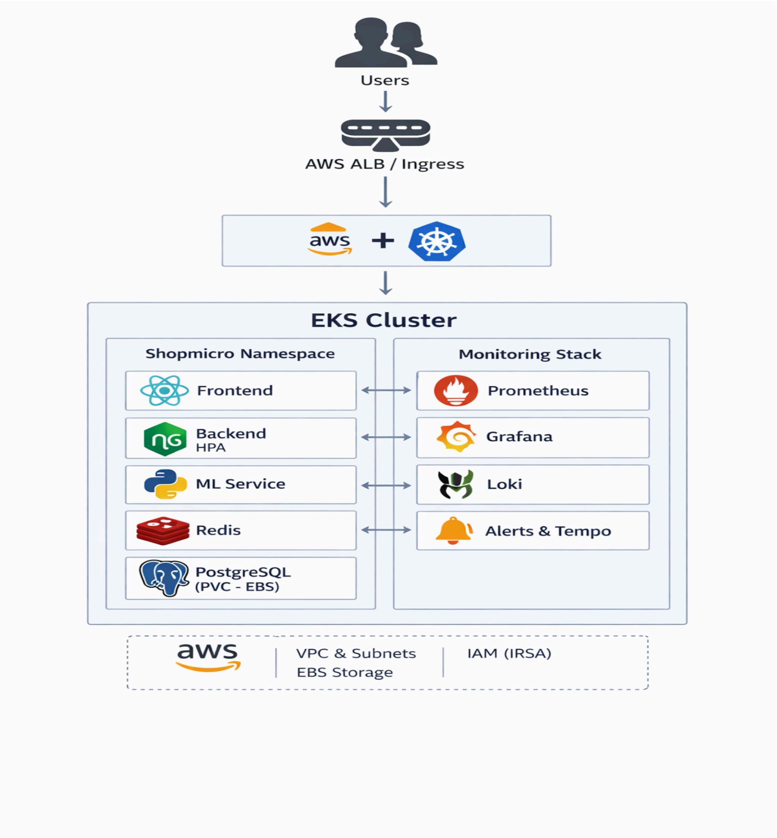
- Containerized microservices (Frontend, Backend, ML service)
- Kubernetes-based orchestration
- Infrastructure-as-Code using Terraform
- CI/CD pipelines with validation and policy enforcement
- Full LGTM observability stack
- Secure networking and IAM boundaries
- Backup and recovery automation

The architecture follows a layered model:

- Application Layer (React, Node.js, Flask)
- Platform Layer (Kubernetes, Ingress, HPA)
- Observability Layer (Prometheus, Loki, Tempo, Grafana)
- Infrastructure Layer (Terraform modules)
- Governance Layer (CI/CD, policy-as-code, drift detection)

---

2. Architecture Diagram



The diagram illustrates:

User traffic entering via Ingress

Frontend communicating with Backend

Backend communicating with ML service

PostgreSQL persistent storage

Redis caching layer

Observability stack integration

3. Infrastructure Design

Custom modular Terraform architecture implemented:

- VPC module (network segmentation, subnets, routing)
- EKS module (cluster and node groups)
- IAM roles and policies
- Node group configuration
- Security boundaries

Remote state strategy documented using:

- S3 backend
- DynamoDB table for state locking

This ensures consistency, collaboration safety, and drift prevention.

---

4. Kubernetes Implementation

Platform controls implemented:

- Dedicated namespace isolation
- ConfigMaps and Secrets (no hardcoded secrets)
- PersistentVolumeClaims using EBS CSI
- Horizontal Pod Autoscaler (backend service)
- Pod anti-affinity rules for resilience
- Ingress-based traffic routing
- NetworkPolicies enforcing least-privilege communication
- Resource requests and limits

Rolling updates and rollback capability were demonstrated.

---

5. Observability Implementation

LGTM stack deployed:

- Prometheus-compatible metrics
- Loki for centralized logging
- Tempo for distributed tracing
- Grafana dashboards

Three dashboards implemented:

- Platform overview
- Backend service health
- Logs and trace correlation

SLIs defined:

- API availability
- Request latency (P95)
- Error rate

SLOs defined with business justification.

Alerts configured for:

- High error rate
- Sustained high latency

---

6. CI/CD Implementation

GitHub Actions workflows implemented:

- Docker image build and push
- Terraform validatio
- Drift detection workflow
- Development deployment automation

Pipeline stages include:

- Linting and formatting
- Test execution
- Infrastructure validation
- Controlled deployment

---

7. Security Design

Security controls implemented:

- Least privilege IAM policies
- No public SSH exposure
- IRSA for Kubernetes controller access
- Encrypted storage volumes

Network segmentation via NetworkPolicies

Secrets managed via Kubernetes Secrets

---

8. Reliability & Operations

Reliability mechanisms implemented:

Rolling updates with zero-downtime strategy

Demonstrated rollback capability

Automated PostgreSQL backup CronJob

Persistent volume strategy

Incident runbook for backend outage scenario

Custom operational CLI tool (shopctl) for environment validation

---

9. Deployment Commands

Infrastructure

```
terraform init
terraform validate
terraform plan
terraform apply
```

Kubernetes Deployment

```
kubectl apply -f k8s/
```

Verify Resources

```
kubectl get pods -n shopmicro
kubectl get svc -n shopmicro
kubectl get ingress -n shopmicro
```

---

10. Test & Verification Commands

Verify backend:

```
kubectl exec -it <frontend-pod> -n shopmicro -- curl http://backend:8080/products
```

Check HPA:

```
kubectl get hpa -n shopmicro
```

Check logs:

```
kubectl logs <backend-pod> -n shopmicro
```

---

11. Observability Usage Guide

Metrics:

Access Grafana

Navigate to Platform Overview dashboard

View API latency and error rate

Logs:

Open Grafana Explore

Query Loki datasource

Filter by namespace=shopmicro

Traces:

Open Tempo datasource

Search by trace ID from logs

---

12. Rollback Procedure

If deployment failure occurs:

```
kubectl rollout undo deployment/backend -n shopmicro
```

Verify rollback:

```
kubectl rollout status deployment/backend -n shopmicro
```

---

13. Backup & Restore Procedure

Backup:

PostgreSQL backup CronJob stores dumps to persistent storage

Manual backup trigger:

```
kubectl create job --from=cronjob/db-backup db-backup-manual -n shopmicro
```

Restore:

Load SQL dump into PostgreSQL pod

Validate data integrity

---

14. Known Limitations & Next Improvements

Limitations:

- Payment processing not implemented
- Single-region deployment
- Limited load testing scope
- Manual cost optimization tuning

Future Improvements:

- Multi-region architecture
- Blue/Green deployments
- Automated restore verification
- FinOps cost dashboards