DEVSECOPS TEST

TAKE-HOME TEST: SECURE MICROSERVICES PLATFORM WITH OBSERVABILITY

OBJECTIVE:

Build a production-ready, secure, and observable microservices platform using modern DevSecOps practices. The system should demonstrate infrastructure automation, security hardening, monitoring integration, and incident response capabilities - all running locally using containers and local Kubernetes.

This test will evaluate:

- Infrastructure as Code with Terraform and Ansible
- Container security and Kubernetes deployment
- Secret management with Infisical
- APM monitoring with Elasticsearch
- Automated alerting and incident response
- Security scanning and compliance automation
- System design and architecture documentation

REQUIREMENTS:

1. TECHNOLOGY STACK

Required:

- **Backend**: NestJS with TypeScript
- **Database**: PostgreSQL with connection pooling
- **Container**: Docker with security best practices
- **Orchestration**: minikube or kind (local Kubernetes)
- **Secret Management**: Infisical with `.machine.infisical.json`
- **Monitoring**: Elasticsearch APM + Kibana (Docker Compose)
- **Infrastructure**: Terraform (for local resources) + Ansible
- **Automation**: GitHub Actions or Makefile-based automation
- **Security**: Container scanning with Trivy

Optional Enhancements:

- **Message Queue**: Redis for caching and sessions
- **Load Balancer**: Nginx with rate limiting
- **Backup**: Automated PostgreSQL backup scripts
- **Chaos Testing**: Simple failure simulation

FEATURES TO IMPLEMENT:

BASIC FEATURES

Note: Focus on demonstrable, working solutions over complex enterprise features.

1. Secure Backend API

- RESTful API with JWT authentication
- Health checks (`/health`, `/metrics`, `/ready`)
- Input validation and rate limiting
- Elasticsearch APM instrumentation
- Structured logging with correlation IDs

2. Infrastructure as Code

- **Terraform**: Local infrastructure setup (Docker networks, volumes)
- **Ansible**: Application deployment and security hardening
- **Docker Compose**: Local development environment
- Automated environment provisioning scripts

3. Secret Management

- Infisical integration with `.machine.infisical.json`
- Database credentials and API keys management
- Kubernetes secrets injection
- Secret rotation demonstration

4. Container Security & Deployment

- Multi-stage Dockerfile with security best practices
- Non-root user, minimal base images
- Container scanning with Trivy in CI pipeline
- Kubernetes deployment with security policies
- Resource limits and health checks

5. Database High Availability

- PostgreSQL with automated backup
- Connection pooling (PgBouncer/connection pooling)
- Database monitoring and alerting
- Simple failover simulation

6. Monitoring & APM

- Elasticsearch APM integration
- Kibana dashboards for application metrics

- Custom business metrics
- Log aggregation and parsing
- Performance monitoring under load

7. Automated Alerting

- Error rate and latency thresholds
- Infrastructure health monitoring
- Chat-bot integration (Slack webhook or Discord)
- Automated incident response scripts
- Alert escalation policies

8. System Design & Topology Documentation

- Complete system architecture diagrams
- Network topology with security zones
- Data flow diagrams (request processing, monitoring, secrets)
- Component interaction maps
- Infrastructure automation workflow diagrams
- Security architecture layers visualization
- Monitoring and observability stack design

BONUS FEATURES (OPTIONAL)

9. Advanced Security

- SAST scanning integration (SonarQube or Snyk)
- Kubernetes security policies (Pod Security Standards)
- Network policies for service isolation
- Security audit logging

10. CI/CD Pipeline

- Automated security scanning gates
- Infrastructure validation
- Zero-downtime deployment simulation
- Automated rollback procedures

11. Performance & Scalability

- Load testing with simple tools (ab, wrk)
- Horizontal Pod Autoscaler configuration
- Database performance optimization
- Caching strategy implementation

12. Incident Response Automation

- Ansible playbooks for common incidents

- Automated service recovery
- Chaos engineering simulation
- Runbook automation

EVALUATION CRITERIA

1. **Infrastructure Automation (25%)**
2. **Security Implementation (25%)**
3. **System Design & Architecture (20%)**
4. **Monitoring & Observability (15%)**
5. **Container & K8s Best Practices (10%)**
6. **Code Quality & Documentation (5%)**

DELIVERABLES

• **GitHub repo with:**

- Working local environment (single command setup)
- Complete setup instructions (`make setup && make deploy`)
- Architecture documentation explaining design decisions
- **System topology diagrams and design documentation**
- Demo script showing all features working
- Incident response simulation
- Performance testing results

Repository Structure:

```
├── README.md (comprehensive setup guide)
├── Makefile (automation commands)
├── ARCHITECTURE.md (design decisions)
├── SYSTEM_DESIGN.md (topology diagrams and system design)
├── apps/
│   └── api/ (NestJS backend application)
├── infrastructure/
│   ├── terraform/ (local infrastructure)
│   ├── ansible/ (deployment playbooks)
│   └── kubernetes/ (K8s manifests)
├── monitoring/
│   └── docker-compose.yml (ELK stack)
├── security/
│   ├── trivy-scan.yml
│   └── policies/
├── scripts/
│   ├── setup.sh
│   ├── deploy.sh
│   └── demo.sh
```

```
├── diagrams/ (system topology and architecture diagrams)
│   ├── system-architecture.png
│   ├── network-topology.png
│   ├── data-flow.png
│   └── security-zones.png
└── docs/
    ├── DEMO.md
    ├── TROUBLESHOOTING.md
    └── ASSUMPTIONS.md
```

Demo Requirements:

1. **Single command setup**: `make setup` creates entire environment
2. **Working API**: All endpoints functional with monitoring
3. **System design presentation**: Clear explanation of architecture decisions
4. **Security demonstration**: Container scan results and secret management
5. **Monitoring dashboards**: Live metrics in Kibana
6. **Incident simulation**: Automated response to simulated failure
7. **Performance test**: Load testing with metrics collection
8. **Topology walkthrough**: Demonstrate understanding of system components

Success Criteria:

- Environment starts successfully on clean machine
- All monitoring dashboards show live data
- Security scans pass with no critical vulnerabilities
- Incident response automation works as demonstrated
- API responds under load with proper monitoring
- Secrets are properly managed and rotated
- System design documentation is comprehensive and clear

Time Allocation Suggestion:

Expected Deliverable Timeline: 7 days maximum for core features + bonus features as time permits.