DevOps Learning Platform - Complete Implementation Guide

Table of Contents

Prerequisites

Required Tools Installation

```
# Docker & Docker Compose
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-
        $(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Kubernetes (kubectl)
curl -LO "https://dl.k8s.io/release/$(curl -L -s
        https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Helm
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee
        /usr/share/keyrings/helm.gpg > /dev/null
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg]
        https://baltocdn.com/helm/stable/debian/ all main" | sudo tee
        /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm

# Kind (Kubernetes in Docker) for local development
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind

# ArgoCD CLI
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-
        cd/releases/latest/download/argocd-linux-amd64
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
```

Docker & Docker Compose

1. Create Dockerfiles

Backend Dockerfile

```
# devops-backend/Dockerfile
FROM python:3.12-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Copy poetry files
COPY pyproject.toml poetry.lock ./

# Install poetry and dependencies
RUN pip install poetry
RUN poetry config virtualenvs.create false
RUN poetry install --no-dev

# Copy application code
COPY . .
```

```
# Expose port
EXPOSE 8000

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/healthz || exit 1

# Run the application
CMD ["poetry", "run", "fastapi", "run", "app/main.py", "--host", "0.0.0.0", "--port",
        "8000"]
```

Frontend Dockerfile

```
# devops-frontend/Dockerfile
FROM node:18-alpine AS builder

WORKDIR /app

# Copy package files
COPY package*.json ./
RUN npm ci

# Copy source code and build
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine

# Copy built assets
COPY --from=builder /app/dist /usr/share/nginx/html

# Copy nginx configuration
COPY nginx.conf /etc/nginx/nginx.conf

# Expose port
EXPOSE 80

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD wget --no-verbose --tries=1 --spider http://localhost/ || exit 1

CMD ["nginx", "-g", "daemon off;"]
```

Frontend Nginx Configuration

```
# devops-frontend/nginx.conf
events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    server {
        listen 80;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html;

        # Gzip compression
        gzip on;
        gzip_types text/plain text/css application/json application/javascript text/xml
application/xml application/xml+rss text/javascript;
```

```
        # Handle client-side routing
        location / {
            try_files $uri $uri/ /index.html;
        }

        # API proxy (if needed for same-origin)
        location /api/ {
            proxy_pass http://backend:8000/api/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        # Health check endpoint
        location /health {
            access_log off;
            return 200 "healthy\n";
            add_header Content-Type text/plain;
        }
    }
}
```

2. Docker Compose Configuration

```yaml
# docker-compose.yml
version: '3.8'

services:
  backend:
    build:
      context: ./devops-backend
      dockerfile: Dockerfile
    container_name: devops-backend
    ports:
      - "8000:8000"
    environment:
      - ENVIRONMENT=production
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/healthz"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    restart: unless-stopped
    networks:
      - devops-network
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.backend.rule=Host(`api.devops-platform.local`)"
      - "traefik.http.services.backend.loadbalancer.server.port=8000"

  frontend:
    build:
      context: ./devops-frontend
      dockerfile: Dockerfile
    container_name: devops-frontend
    ports:
      - "80:80"
    depends_on:
      backend:
        condition: service_healthy
    environment:
      - VITE_API_URL=http://localhost:8000
    healthcheck:
```

```yaml
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider",
        "http://localhost/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    restart: unless-stopped
    networks:
      - devops-network
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.frontend.rule=Host(`devops-platform.local`)"
      - "traefik.http.services.frontend.loadbalancer.server.port=80"

  # Reverse proxy for load balancing and SSL termination
  traefik:
    image: traefik:v3.0
    container_name: traefik
    command:
      - "--api.insecure=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
    ports:
      - "80:80"
      - "443:443"
      - "8080:8080"  # Traefik dashboard
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    networks:
      - devops-network
    restart: unless-stopped

networks:
  devops-network:
    driver: bridge

volumes:
  backend_data:
```

3. Docker Commands

```bash
# Build and run with Docker Compose
cd /path/to/devops-learning-platform
docker-compose up --build -d

# View logs
docker-compose logs -f

# Scale services
docker-compose up --scale backend=3 -d

# Stop services
docker-compose down

# Remove everything including volumes
docker-compose down -v --remove-orphans

# Build individual images
docker build -t devops-backend:latest ./devops-backend
docker build -t devops-frontend:latest ./devops-frontend

# Run individual containers
docker run -d --name backend -p 8000:8000 devops-backend:latest
docker run -d --name frontend -p 80:80 devops-frontend:latest
```

```
# Container management
docker ps                    # List running containers
docker logs backend          # View container logs
docker exec -it backend bash # Access container shell
docker stats                 # View resource usage
```

Kubernetes Deployment

1. Create Local Kubernetes Cluster

```
# Create Kind cluster with custom configuration
cat <<EOF > kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
- role: worker
- role: worker
EOF

# Create cluster
kind create cluster --config=kind-config.yaml --name devops-platform

# Verify cluster
kubectl cluster-info
kubectl get nodes
```

2. Kubernetes Manifests

Namespace

```
# k8s/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: devops-platform
  labels:
    name: devops-platform
```

Backend Deployment

```
# k8s/backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: devops-platform
  labels:
    app: backend
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: devops-backend:latest
        imagePullPolicy: Never  # For Kind local images
        ports:
        - containerPort: 8000
        env:
        - name: ENVIRONMENT
          value: "production"
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /healthz
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: devops-platform
spec:
  selector:
    app: backend
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 8000
  type: ClusterIP
```

Frontend Deployment

```
# k8s/frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: devops-platform
  labels:
    app: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
```

```yaml
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: devops-frontend:latest
        imagePullPolicy: Never  # For Kind local images
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: devops-platform
spec:
  selector:
    app: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

ConfigMap for Environment Variables

```yaml
# k8s/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: devops-platform
data:
  ENVIRONMENT: "production"
  API_URL: "http://backend-service:8000"
  LOG_LEVEL: "info"
```

3. Deploy to Kubernetes

```
# Load Docker images into Kind
kind load docker-image devops-backend:latest --name devops-platform
kind load docker-image devops-frontend:latest --name devops-platform

# Apply Kubernetes manifests
kubectl apply -f k8s/namespace.yaml
```

```
kubectl apply -f k8s/configmap.yaml
kubectl apply -f k8s/backend-deployment.yaml
kubectl apply -f k8s/frontend-deployment.yaml

# Verify deployments
kubectl get all -n devops-platform
kubectl get pods -n devops-platform -w

# Check logs
kubectl logs -f deployment/backend -n devops-platform
kubectl logs -f deployment/frontend -n devops-platform

# Port forward for testing
kubectl port-forward service/frontend-service 8080:80 -n devops-platform
kubectl port-forward service/backend-service 8000:8000 -n devops-platform
```

CI/CD Pipeline

1. GitHub Actions Workflow

```yaml
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME_BACKEND: ${{ github.repository }}/backend
  IMAGE_NAME_FRONTEND: ${{ github.repository }}/frontend

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.12'

    - name: Set up Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'

    - name: Install Python dependencies
      run: |
        cd devops-backend
        pip install poetry
        poetry install

    - name: Install Node.js dependencies
      run: |
        cd devops-frontend
        npm ci

    - name: Run backend tests
      run: |
        cd devops-backend
        poetry run pytest
```

```yaml
    - name: Run frontend tests
      run: |
        cd devops-frontend
        npm run test

    - name: Run linting
      run: |
        cd devops-backend
        poetry run flake8 app/
        cd ../devops-frontend
        npm run lint

  security-scan:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        scan-type: 'fs'
        scan-ref: '.'
        format: 'sarif'
        output: 'trivy-results.sarif'

    - name: Upload Trivy scan results
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: 'trivy-results.sarif'

  build-and-push:
    needs: [test, security-scan]
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v4

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Extract metadata (backend)
      id: meta-backend
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME_BACKEND }}
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=sha,prefix={{branch}}-
          type=raw,value=latest,enable={{is_default_branch}}

    - name: Extract metadata (frontend)
      id: meta-frontend
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME_FRONTEND }}
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=sha,prefix={{branch}}-
          type=raw,value=latest,enable={{is_default_branch}}
```

```yaml
    - name: Build and push backend image
      uses: docker/build-push-action@v5
      with:
        context: ./devops-backend
        push: true
        tags: ${{ steps.meta-backend.outputs.tags }}
        labels: ${{ steps.meta-backend.outputs.labels }}

    - name: Build and push frontend image
      uses: docker/build-push-action@v5
      with:
        context: ./devops-frontend
        push: true
        tags: ${{ steps.meta-frontend.outputs.tags }}
        labels: ${{ steps.meta-frontend.outputs.labels }}

  deploy:
    needs: build-and-push
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v4

    - name: Set up kubectl
      uses: azure/setup-kubectl@v3
      with:
        version: 'latest'

    - name: Configure kubectl
      run: |
        echo "${{ secrets.KUBE_CONFIG }}" | base64 -d > kubeconfig
        export KUBECONFIG=kubeconfig

    - name: Deploy to Kubernetes
      run: |
        export KUBECONFIG=kubeconfig
        kubectl set image deployment/backend backend=${{ env.REGISTRY }}/${{
        env.IMAGE_NAME_BACKEND }}:latest -n devops-platform
        kubectl set image deployment/frontend frontend=${{ env.REGISTRY }}/${{
        env.IMAGE_NAME_FRONTEND }}:latest -n devops-platform
        kubectl rollout status deployment/backend -n devops-platform
        kubectl rollout status deployment/frontend -n devops-platform
```

2. GitLab CI/CD (Alternative)

```yaml
# .gitlab-ci.yml
stages:
  - test
  - security
  - build
  - deploy

variables:
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"

test-backend:
  stage: test
  image: python:3.12
  before_script:
    - cd devops-backend
    - pip install poetry
    - poetry install
  script:
    - poetry run pytest
    - poetry run flake8 app/
```

```yaml
  coverage: '/TOTAL.*\s+(\d+%)$/'

test-frontend:
  stage: test
  image: node:18
  before_script:
    - cd devops-frontend
    - npm ci
  script:
    - npm run test
    - npm run lint
  artifacts:
    reports:
      coverage_report:
        coverage_format: cobertura
        path: devops-frontend/coverage/cobertura-coverage.xml

security-scan:
  stage: security
  image: aquasec/trivy:latest
  script:
    - trivy fs --exit-code 0 --format template --template "@contrib/sarif.tpl" -o trivy-
        report.sarif .
    - trivy fs --exit-code 1 --severity HIGH,CRITICAL .
  artifacts:
    reports:
      sast: trivy-report.sarif

build-backend:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  script:
    - cd devops-backend
    - docker build -t $CI_REGISTRY_IMAGE/backend:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE/backend:$CI_COMMIT_SHA
  only:
    - main

build-frontend:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  script:
    - cd devops-frontend
    - docker build -t $CI_REGISTRY_IMAGE/frontend:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE/frontend:$CI_COMMIT_SHA
  only:
    - main

deploy-staging:
  stage: deploy
  image: bitnami/kubectl:latest
  script:
    - kubectl config use-context $KUBE_CONTEXT
    - kubectl set image deployment/backend
        backend=$CI_REGISTRY_IMAGE/backend:$CI_COMMIT_SHA -n devops-platform-staging
    - kubectl set image deployment/frontend
        frontend=$CI_REGISTRY_IMAGE/frontend:$CI_COMMIT_SHA -n devops-platform-staging
    - kubectl rollout status deployment/backend -n devops-platform-staging
    - kubectl rollout status deployment/frontend -n devops-platform-staging
  environment:
```

```yaml
    name: staging
    url: https://staging.devops-platform.com
  only:
    - main

deploy-production:
  stage: deploy
  image: bitnami/kubectl:latest
  script:
    - kubectl config use-context $KUBE_CONTEXT
    - kubectl set image deployment/backend
        backend=$CI_REGISTRY_IMAGE/backend:$CI_COMMIT_SHA -n devops-platform
    - kubectl set image deployment/frontend
        frontend=$CI_REGISTRY_IMAGE/frontend:$CI_COMMIT_SHA -n devops-platform
    - kubectl rollout status deployment/backend -n devops-platform
    - kubectl rollout status deployment/frontend -n devops-platform
  environment:
    name: production
    url: https://devops-platform.com
  when: manual
  only:
    - main
```

Monitoring & Observability

1. Prometheus & Grafana Setup

```yaml
# monitoring/prometheus-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      evaluation_interval: 15s

    rule_files:
      - "alert_rules.yml"

    alerting:
      alertmanagers:
        - static_configs:
            - targets:
              - alertmanager:9093

    scrape_configs:
      - job_name: 'prometheus'
        static_configs:
          - targets: ['localhost:9090']

      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
            action: keep
            regex: true
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
            action: replace
            target_label: __metrics_path__
            regex: (.+)
```

```
          - source_labels: [__address__,
        __meta_kubernetes_pod_annotation_prometheus_io_port]
            action: replace
            regex: ([^:]+)(?::\d+)?;(\d+)
            replacement: $1:$2
            target_label: __address__

      - job_name: 'devops-backend'
        static_configs:
          - targets: ['backend-service:8000']
        metrics_path: '/metrics'

      - job_name: 'devops-frontend'
        static_configs:
          - targets: ['frontend-service:80']
        metrics_path: '/metrics'

  alert_rules.yml: |
    groups:
      - name: devops-platform-alerts
        rules:
          - alert: HighErrorRate
            expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.1
            for: 5m
            labels:
              severity: warning
            annotations:
              summary: "High error rate detected"
              description: "Error rate is {{ $value }} errors per second"

          - alert: HighMemoryUsage
            expr: container_memory_usage_bytes / container_spec_memory_limit_bytes > 0.8
            for: 5m
            labels:
              severity: warning
            annotations:
              summary: "High memory usage"
              description: "Memory usage is above 80%"

          - alert: PodCrashLooping
            expr: rate(kube_pod_container_status_restarts_total[15m]) > 0
            for: 5m
            labels:
              severity: critical
            annotations:
              summary: "Pod is crash looping"
              description: "Pod {{ $labels.pod }} is restarting frequently"

# Install Prometheus and Grafana using Helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

# Create monitoring namespace
kubectl create namespace monitoring

# Install Prometheus
helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false \
  --set prometheus.prometheusSpec.podMonitorSelectorNilUsesHelmValues=false

# Install Grafana (if not included in kube-prometheus-stack)
helm install grafana grafana/grafana \
  --namespace monitoring \
  --set adminPassword=admin123 \
  --set service.type=NodePort
```

```
# Get Grafana admin password
kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" |
        base64 --decode

# Port forward to access services
kubectl port-forward service/prometheus-kube-prometheus-prometheus 9090:9090 -n monitoring
kubectl port-forward service/grafana 3000:80 -n monitoring
```

2. Application Metrics

Backend Metrics (FastAPI)

```
# devops-backend/app/metrics.py
from prometheus_client import Counter, Histogram, Gauge, generate_latest
from fastapi import Request
import time

# Metrics
REQUEST_COUNT = Counter('http_requests_total', 'Total HTTP requests', ['method',
        'endpoint', 'status'])
REQUEST_DURATION = Histogram('http_request_duration_seconds', 'HTTP request duration')
ACTIVE_CONNECTIONS = Gauge('active_connections', 'Active connections')
COURSE_ACCESS_COUNT = Counter('course_access_total', 'Total course accesses',
        ['course_id'])

def record_request_metrics(request: Request, response_time: float, status_code: int):
    REQUEST_COUNT.labels(
        method=request.method,
        endpoint=request.url.path,
        status=status_code
    ).inc()
    REQUEST_DURATION.observe(response_time)

# Add to main.py
from fastapi import FastAPI, Request
from fastapi.responses import Response
import time
from .metrics import record_request_metrics, generate_latest

@app.middleware("http")
async def metrics_middleware(request: Request, call_next):
    start_time = time.time()
    response = await call_next(request)
    process_time = time.time() - start_time
    record_request_metrics(request, process_time, response.status_code)
    return response

@app.get("/metrics")
async def metrics():
    return Response(generate_latest(), media_type="text/plain")
```

3. Logging with ELK Stack

```
# logging/elasticsearch.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
  namespace: logging
spec:
  replicas: 1
  selector:
    matchLabels:
      app: elasticsearch
  template:
    metadata:
```

```yaml
      labels:
        app: elasticsearch
    spec:
      containers:
      - name: elasticsearch
        image: docker.elastic.co/elasticsearch/elasticsearch:8.8.0
        env:
        - name: discovery.type
          value: single-node
        - name: ES_JAVA_OPTS
          value: "-Xms512m -Xmx512m"
        - name: xpack.security.enabled
          value: "false"
        ports:
        - containerPort: 9200
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
---
apiVersion: v1
kind: Service
metadata:
  name: elasticsearch
  namespace: logging
spec:
  selector:
    app: elasticsearch
  ports:
  - port: 9200
    targetPort: 9200

# Install ELK Stack using Helm
helm repo add elastic https://helm.elastic.co
helm repo update

kubectl create namespace logging

# Install Elasticsearch
helm install elasticsearch elastic/elasticsearch \
  --namespace logging \
  --set replicas=1 \
  --set minimumMasterNodes=1

# Install Kibana
helm install kibana elastic/kibana \
  --namespace logging \
  --set service.type=NodePort

# Install Filebeat for log collection
helm install filebeat elastic/filebeat \
  --namespace logging \
  --set daemonset.enabled=true
```

GitOps Implementation

1. ArgoCD Installation

```
# Install ArgoCD
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-
        cd/stable/manifests/install.yaml
```

```
# Wait for ArgoCD to be ready
kubectl wait --for=condition=available --timeout=300s deployment/argocd-server -n argocd

# Get ArgoCD admin password
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" |
        base64 -d

# Port forward to access ArgoCD UI
kubectl port-forward svc/argocd-server -n argocd 8080:443

# Login with ArgoCD CLI
argocd login localhost:8080 --username admin --password <password> --insecure
```

2. GitOps Repository Structure

```
gitops-config/
├── applications/
│   ├── backend.yaml
│   ├── frontend.yaml
│   └── monitoring.yaml
├── environments/
│   ├── staging/
│   │   ├── backend/
│   │   └── frontend/
│   └── production/
│       ├── backend/
│       └── frontend/
└── base/
    ├── backend/
    └── frontend/
```

3. ArgoCD Application Manifests

```yaml
# gitops-config/applications/backend.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: devops-backend
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/yourusername/devops-platform-gitops
    targetRevision: HEAD
    path: environments/production/backend
  destination:
    server: https://kubernetes.default.svc
    namespace: devops-platform
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m

# gitops-config/applications/frontend.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
```

```yaml
metadata:
  name: devops-frontend
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/yourusername/devops-platform-gitops
    targetRevision: HEAD
    path: environments/production/frontend
  destination:
    server: https://kubernetes.default.svc
    namespace: devops-platform
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

4. Kustomization for Environment-Specific Configs

```yaml
# gitops-config/base/backend/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - deployment.yaml
  - service.yaml
  - configmap.yaml

commonLabels:
  app: backend
  version: v1.0.0

# gitops-config/environments/production/backend/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

namespace: devops-platform

resources:
  - ../../../base/backend

patchesStrategicMerge:
  - deployment-patch.yaml

images:
  - name: devops-backend
    newTag: latest

replicas:
  - name: backend
    count: 3
```

5. GitOps Workflow Commands

```bash
# Apply ArgoCD applications
kubectl apply -f gitops-config/applications/ -n argocd

# Sync applications manually
argocd app sync devops-backend
argocd app sync devops-frontend

# Check application status
```

```
argocd app list
argocd app get devops-backend

# Enable auto-sync
argocd app set devops-backend --sync-policy automated

# Rollback to previous version
argocd app rollback devops-backend

# Refresh application (check for changes)
argocd app refresh devops-backend
```

Ingress Controller

1. NGINX Ingress Controller Installation

```
# Install NGINX Ingress Controller
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-
        v1.8.1/deploy/static/provider/kind/deploy.yaml

# Wait for ingress controller to be ready
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s

# Verify installation
kubectl get pods -n ingress-nginx
```

2. Ingress Configuration

```yaml
# k8s/ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: devops-platform-ingress
  namespace: devops-platform
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/cors-allow-origin: "*"
    nginx.ingress.kubernetes.io/cors-allow-methods: "GET, POST, PUT, DELETE, OPTIONS"
    nginx.ingress.kubernetes.io/cors-allow-headers: "DNT,User-Agent,X-Requested-With,If-
        Modified-Since,Cache-Control,Content-Type,Range,Authorization"
spec:
  ingressClassName: nginx
  rules:
  - host: devops-platform.local
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 8000
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

```
  - host: api.devops-platform.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 8000
```

3. TLS/SSL Configuration

```
# k8s/tls-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: devops-platform-tls-ingress
  namespace: devops-platform
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - devops-platform.com
    - api.devops-platform.com
    secretName: devops-platform-tls
  rules:
  - host: devops-platform.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 8000
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

4. Cert-Manager for SSL Certificates

```
# Install cert-manager
kubectl apply -f https://github.com/cert-manager/cert-
        manager/releases/download/v1.12.0/cert-manager.yaml

# Create ClusterIssuer for Let's Encrypt
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: your-email@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
```

```
    - http01:
        ingress:
          class: nginx
EOF
```

5. Load Balancing and Rate Limiting

```yaml
# k8s/advanced-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: devops-platform-advanced-ingress
  namespace: devops-platform
  annotations:
    nginx.ingress.kubernetes.io/rate-limit: "100"
    nginx.ingress.kubernetes.io/rate-limit-window: "1m"
    nginx.ingress.kubernetes.io/limit-connections: "10"
    nginx.ingress.kubernetes.io/upstream-hash-by: "$remote_addr"
    nginx.ingress.kubernetes.io/load-balance: "round_robin"
    nginx.ingress.kubernetes.io/proxy-body-size: "10m"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "60"
spec:
  ingressClassName: nginx
  rules:
  - host: devops-platform.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 8000
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

Security Best Practices

1. Network Policies

```yaml
# k8s/network-policies.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: devops-platform-network-policy
  namespace: devops-platform
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
```

```
        matchLabels:
          app: frontend
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 8000
  egress:
  - to: []
    ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: frontend-network-policy
  namespace: devops-platform
spec:
  podSelector:
    matchLabels:
      app: frontend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 80
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: backend
    ports:
    - protocol: TCP
      port: 8000
```

2. Pod Security Standards

```
# k8s/pod-security-policy.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: devops-platform
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

3. RBAC Configuration

```
# k8s/rbac.yaml
apiVersion: v1
kind: ServiceAccount
```

```yaml
metadata:
  name: devops-platform-sa
  namespace: devops-platform
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: devops-platform
  name: devops-platform-role
rules:
- apiGroups: [""]
  resources: ["pods", "services", "configmaps"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devops-platform-rolebinding
  namespace: devops-platform
subjects:
- kind: ServiceAccount
  name: devops-platform-sa
  namespace: devops-platform
roleRef:
  kind: Role
  name: devops-platform-role
  apiGroup: rbac.authorization.k8s.io
```

4. Secrets Management

```bash
# Create secrets for sensitive data
kubectl create secret generic app-secrets \
  --from-literal=database-password=supersecret \
  --from-literal=api-key=your-api-key \
  -n devops-platform

# Use external secrets operator for cloud integration
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets external-secrets/external-secrets -n external-secrets-system
        --create-namespace
```

Troubleshooting

1. Common Issues and Solutions

Docker Issues

```bash
# Container won't start
docker logs <container-name>
docker inspect <container-name>

# Port conflicts
docker ps -a
netstat -tulpn | grep <port>

# Image build failures
docker build --no-cache -t <image-name> .
docker system prune -a
```

Kubernetes Issues

```
# Pod stuck in Pending state
kubectl describe pod <pod-name> -n devops-platform
kubectl get events -n devops-platform --sort-by='.lastTimestamp'

# Service not accessible
kubectl get endpoints -n devops-platform
kubectl port-forward service/<service-name> <local-port>:<service-port> -n devops-platform

# Ingress not working
kubectl describe ingress -n devops-platform
kubectl logs -n ingress-nginx deployment/ingress-nginx-controller
```

CI/CD Pipeline Issues

```
# GitHub Actions debugging
# Check workflow logs in GitHub Actions tab
# Verify secrets are set correctly
# Test Docker builds locally

# ArgoCD sync issues
argocd app get <app-name>
argocd app sync <app-name> --dry-run
kubectl logs -n argocd deployment/argocd-application-controller
```

2. Monitoring and Debugging Commands

```
# Resource usage
kubectl top nodes
kubectl top pods -n devops-platform

# Cluster information
kubectl cluster-info
kubectl get componentstatuses

# Network debugging
kubectl run debug-pod --image=nicolaka/netshoot -it --rm
nslookup backend-service.devops-platform.svc.cluster.local

# Application logs
kubectl logs -f deployment/backend -n devops-platform
kubectl logs -f deployment/frontend -n devops-platform --previous
```

3. Performance Optimization

```
# Horizontal Pod Autoscaler
kubectl autoscale deployment backend --cpu-percent=70 --min=2 --max=10 -n devops-platform

# Vertical Pod Autoscaler
kubectl apply -f https://github.com/kubernetes/autoscaler/releases/download/vertical-pod-
        autoscaler-0.13.0/vpa-release.yaml
```

Conclusion

This comprehensive guide covers the implementation of DevOps practices for your learning platform including:

- **Containerization** with Docker and Docker Compose
- **Orchestration** with Kubernetes
- **CI/CD** pipelines with GitHub Actions/GitLab CI
- **Monitoring** with Prometheus, Grafana, and ELK stack
- **GitOps** with ArgoCD
- **Ingress** management with NGINX

- **Security** best practices

Each section provides step-by-step commands and configurations that you can execute to implement these DevOps practices in your environment.

Next Steps:

1. Start with Docker containerization
2. Set up local Kubernetes cluster
3. Implement CI/CD pipeline
4. Add monitoring and observability
5. Configure GitOps workflow
6. Set up ingress and security

Remember to adapt the configurations to your specific environment and requirements. Test each component thoroughly before moving to the next step.