Kubernetes CLI Guide - Sections 17-21 (Continued)

17. StatefulSets (Continued)

```bash
```

```bash
#!/bin/bash
# save as statefulset-demo.sh (continued from previous section)
echo "📊 StatefulSet Demo: Scaling and Persistence"

# Deploy the MySQL StatefulSet
kubectl apply -f mysql-statefulset.yaml

echo "⏳ Waiting for StatefulSet to be ready..."
kubectl wait --for=condition=ready pod -l app=mysql --timeout=300s

echo "🔍 Initial StatefulSet status:"
kubectl get statefulset mysql
kubectl get pods -l app=mysql

echo "💾 Checking persistent volumes:"
kubectl get pvc -l app=mysql

# Test data persistence by connecting to each MySQL instance
echo "🧪 Testing individual pod identity and persistence:"
for i in {0..2}; do
    echo "--- mysql-$i ---"
    kubectl exec mysql-$i -- mysql -u root -prootpassword -e "
        CREATE DATABASE IF NOT EXISTS pod_$i;
        USE pod_$i;
        CREATE TABLE IF NOT EXISTS test_table (id INT, pod_name VARCHAR(50));
        INSERT INTO test_table VALUES ($i, 'mysql-$i');
        SELECT * FROM test_table;
    " 2>/dev/null || echo "Pod mysql-$i not ready yet"
done

# Demonstrate ordered scaling
echo "📈 Testing ordered scaling (scale up to 5):"
kubectl scale statefulset mysql --replicas=5
kubectl get pods -l app=mysql -w &
WATCH_PID=$!
sleep 30
kill $WATCH_PID

echo "📉 Testing ordered scaling (scale down to 2):"
kubectl scale statefulset mysql --replicas=2
kubectl get pods -l app=mysql

# Test persistence after pod deletion
```

```
echo "🔄 Testing persistence after pod deletion:"
kubectl delete pod mysql-1
kubectl wait --for=condition=ready pod mysql-1 --timeout=120s

# Verify data survived pod recreation
kubectl exec mysql-1 -- mysql -u root -prootpassword -e "
    USE pod_1;
    SELECT * FROM test_table;
" 2>/dev/null

echo "✅ StatefulSet demo complete!"
echo "🧹 Cleanup: kubectl delete statefulset mysql && kubectl delete service mysql-headless && kubectl delete pvc -l
```

Mini-Project: Distributed Database Cluster

Create a complete distributed database setup with leader election and data replication:

bash

```bash
#!/bin/bash
# save as distributed-db-cluster.sh
echo "🗄️ Creating Distributed Database Cluster"

# Create ConfigMap for database configuration
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
data:
  primary_init.sh: |
    #!/bin/bash
    set -e
    if [ ! -f /var/lib/postgresql/data/PG_VERSION ]; then
        echo "Initializing primary database..."
        initdb -D /var/lib/postgresql/data
        echo "host all all 0.0.0.0/0 md5" >> /var/lib/postgresql/data/pg_hba.conf
        echo "listen_addresses = '*'" >> /var/lib/postgresql/data/postgresql.conf
        echo "wal_level = replica" >> /var/lib/postgresql/data/postgresql.conf
        echo "max_wal_senders = 3" >> /var/lib/postgresql/data/postgresql.conf
        echo "max_replication_slots = 3" >> /var/lib/postgresql/data/postgresql.conf
    fi
  replica_init.sh: |
    #!/bin/bash
    set -e
    if [ ! -f /var/lib/postgresql/data/PG_VERSION ]; then
        echo "Initializing replica database..."
        pg_basebackup -h postgres-0.postgres-headless -D /var/lib/postgresql/data -U postgres -v -P -W
        echo "standby_mode = 'on'" >> /var/lib/postgresql/data/recovery.conf
        echo "primary_conninfo = 'host=postgres-0.postgres-headless port=5432 user=postgres'" >> /var/lib/postgresql/
    fi
EOF

# Create headless service for StatefulSet
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: postgres-headless
  labels:
    app: postgres
spec:
```

```
  ports:
  - port: 5432
    name: postgres
  clusterIP: None
  selector:
    app: postgres
---
# Create regular service for client connections (points to primary)
apiVersion: v1
kind: Service
metadata:
  name: postgres-primary
  labels:
    app: postgres
spec:
  ports:
  - port: 5432
    name: postgres
  selector:
    app: postgres
    role: primary
EOF

# Create the PostgreSQL StatefulSet with leader election
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: postgres-headless
  replicas: 3
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:13
        env:
```

```
- name: POSTGRES_PASSWORD
  value: "secretpassword"
- name: POSTGRES_USER
  value: "postgres"
- name: POSTGRES_DB
  value: "testdb"
- name: PGDATA
  value: /var/lib/postgresql/data/pgdata
- name: POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
ports:
- containerPort: 8080
  name: http
  protocol: TCP
env:
- name: ENVIRONMENT
  value: ${ENVIRONMENT}
- name: LOG_LEVEL
  value: ${LOG_LEVEL}
- name: DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: ${APP_NAME}-secrets
      key: database-password
- name: API_KEY
  valueFrom:
    secretKeyRef:
      name: ${APP_NAME}-secrets
      key: api-key
envFrom:
- configMapRef:
    name: ${APP_NAME}-config
resources:
  requests:
    memory: ${MEMORY_REQUEST}
    cpu: ${CPU_REQUEST}
  limits:
    memory: ${MEMORY_LIMIT}
    cpu: ${CPU_LIMIT}
livenessProbe:
  httpGet:
    path: /health
```

```
        port: 8080
      initialDelaySeconds: 30
      periodSeconds: 10
      timeoutSeconds: 5
      failureThreshold: 3
    readinessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5
      timeoutSeconds: 3
      failureThreshold: 3
    volumeMounts:
    - name: app-config
      mountPath: /etc/config
      readOnly: true
    - name: temp-storage
      mountPath: /tmp
  volumes:
  - name: app-config
    configMap:
      name: ${APP_NAME}-config
  - name: temp-storage
    emptyDir: {}
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - ${APP_NAME}
          topologyKey: kubernetes.io/hostname
---
apiVersion: v1
kind: Service
metadata:
  name: ${APP_NAME}-service
  namespace: ${APP_NAME}-${ENVIRONMENT}
  labels:
```

```
    app: ${APP_NAME}
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: ${APP_NAME}
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ${APP_NAME}-ingress
  namespace: ${APP_NAME}-${ENVIRONMENT}
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - ${APP_NAME}-${ENVIRONMENT}.${DOMAIN}
    secretName: ${APP_NAME}-tls
  rules:
  - host: ${APP_NAME}-${ENVIRONMENT}.${DOMAIN}
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ${APP_NAME}-service
            port:
              number: 80
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: ${APP_NAME}-hpa
  namespace: ${APP_NAME}-${ENVIRONMENT}
spec:
  scaleTargetRef:
```

```
    apiVersion: apps/v1
    kind: Deployment
    name: ${APP_NAME}
  minReplicas: ${MIN_REPLICAS}
  maxReplicas: ${MAX_REPLICAS}
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: ${APP_NAME}-pdb
  namespace: ${APP_NAME}-${ENVIRONMENT}
spec:
  minAvailable: 50%
  selector:
    matchLabels:
      app: ${APP_NAME}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ${APP_NAME}-network-policy
  namespace: ${APP_NAME}-${ENVIRONMENT}
spec:
  podSelector:
    matchLabels:
      app: ${APP_NAME}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
```

```
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    - namespaceSelector:
        matchLabels:
          environment: ${ENVIRONMENT}
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to: []
    ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
  - to:
    - namespaceSelector:
        matchLabels:
          name: database
    ports:
    - protocol: TCP
      port: 5432
EOF

# Create deployment pipeline script
cat << 'EOF' > deploy-production-app.sh
#!/bin/bash
set -e

# Configuration
APP_NAME=${1:-"webapp"}
ENVIRONMENT=${2:-"staging"}
APP_VERSION=${3:-"latest"}
CONFIG_FILE=${4:-"deployment-config.env"}

echo "🚀 Production Deployment Pipeline"
echo "================================="
echo "Application: $APP_NAME"
echo "Environment: $ENVIRONMENT"
echo "Version: $APP_VERSION"

# Load environment-specific configuration
if [ -f "$CONFIG_FILE" ]; then
```

```
    source "$CONFIG_FILE"
else
    echo "⚠️  Configuration file not found: $CONFIG_FILE"
    echo "Creating default configuration..."

    case $ENVIRONMENT in
        "development")
            REPLICAS=1
            MIN_REPLICAS=1
            MAX_REPLICAS=3
            CPU_REQUEST="100m"
            CPU_LIMIT="500m"
            MEMORY_REQUEST="128Mi"
            MEMORY_LIMIT="512Mi"
            LOG_LEVEL="debug"
            ;;
        "staging")
            REPLICAS=2
            MIN_REPLICAS=2
            MAX_REPLICAS=5
            CPU_REQUEST="200m"
            CPU_LIMIT="1000m"
            MEMORY_REQUEST="256Mi"
            MEMORY_LIMIT="1Gi"
            LOG_LEVEL="info"
            ;;
        "production")
            REPLICAS=3
            MIN_REPLICAS=3
            MAX_REPLICAS=10
            CPU_REQUEST="500m"
            CPU_LIMIT="2000m"
            MEMORY_REQUEST="512Mi"
            MEMORY_LIMIT="2Gi"
            LOG_LEVEL="warn"
            ;;
    esac

    # Default values
    IMAGE_REGISTRY=${IMAGE_REGISTRY:-"registry.company.com"}
    DOMAIN=${DOMAIN:-"company.com"}
    DB_HOST=${DB_HOST:-"postgres.database.svc.cluster.local"}
    DB_PORT=${DB_PORT:-"5432"}
    CACHE_ENABLED=${CACHE_ENABLED:-"true"}
```

```
    # Generate base64 encoded secrets (in production, use proper secret management)
    DB_PASSWORD_B64=$(echo "secretpassword" | base64 -w 0)
    API_KEY_B64=$(echo "api-key-12345" | base64 -w 0)
fi

# Pre-deployment validations
echo "🔍 Running pre-deployment validations..."

# Check if kubectl is configured
if ! kubectl cluster-info >/dev/null 2>&1; then
    echo "❌ kubectl not configured or cluster not accessible"
    exit 1
fi

# Check if the image exists (mock check)
echo "🐬 Validating container image: ${IMAGE_REGISTRY}/${APP_NAME}:${APP_VERSION}"
# In production, you would check the registry
# docker manifest inspect ${IMAGE_REGISTRY}/${APP_NAME}:${APP_VERSION} >/dev/null || {
#     echo "❌ Image not found in registry"
#     exit 1
# }

# Security scan (mock)
echo "🔒 Running security scan..."
# In production, integrate with tools like Twistlock, Aqua, or Snyk
sleep 2
echo "✅ Security scan passed"

# Generate deployment manifests
echo "📝 Generating deployment manifests..."
TEMP_DIR=$(mktemp -d)
envsubst < production-app-template.yaml > "$TEMP_DIR/deployment.yaml"

# Validate generated YAML
echo "✅ Validating generated manifests..."
kubectl apply --dry-run=client -f "$TEMP_DIR/deployment.yaml" >/dev/null

# Deploy application
echo "🚀 Deploying application..."

# Create/update namespace first
kubectl apply -f - <<EOF
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: ${APP_NAME}-${ENVIRONMENT}
  labels:
    environment: ${ENVIRONMENT}
    managed-by: deployment-pipeline
EOF

# Apply all resources
kubectl apply -f "$TEMP_DIR/deployment.yaml"

# Wait for deployment to complete
echo "⏳ Waiting for deployment to complete..."
kubectl rollout status deployment/${APP_NAME} -n ${APP_NAME}-${ENVIRONMENT} --timeout=600s

# Verify deployment
echo "🔍 Verifying deployment..."

# Check pod status
READY_PODS=$(kubectl get deployment ${APP_NAME} -n ${APP_NAME}-${ENVIRONMENT} -o jsonpath='{.status.read
DESIRED_PODS=$(kubectl get deployment ${APP_NAME} -n ${APP_NAME}-${ENVIRONMENT} -o jsonpath='{.spec.repl

if [ "$READY_PODS" = "$DESIRED_PODS" ]; then
    echo "✅ All pods are ready ($READY_PODS/$DESIRED_PODS)"
else
    echo "❌ Not all pods are ready ($READY_PODS/$DESIRED_PODS)"
    kubectl get pods -n ${APP_NAME}-${ENVIRONMENT}
    exit 1
fi

# Health check
echo "🏥 Running health checks..."
kubectl run health-check-$(date +%s) \
    --image=busybox \
    --rm -i --restart=Never \
    --namespace=${APP_NAME}-${ENVIRONMENT} \
    -- wget -q --spider http://${APP_NAME}-service/health || {
    echo "❌ Health check failed"
    exit 1
}

echo "✅ Health check passed"

# Performance test (basic)
```

```
echo "⚡ Running basic performance test..."
kubectl run perf-test-$(date +%s) \
  --image=busybox \
  --rm -i --restart=Never \
  --namespace=${APP_NAME}-${ENVIRONMENT} \
  -- sh -c "
  for i in \$(seq 1 10); do
    wget -q -O- http://${APP_NAME}-service/ >/dev/null && echo 'Request \$i: OK' || echo 'Request \$i: FAILED'
  done
  " || {
  echo "⚠️  Performance test had some failures"
}

# Generate deployment report
echo "📊 Generating deployment report..."
REPORT_FILE="/tmp/deployment-report-${APP_NAME}-${ENVIRONMENT}-$(date +%Y%m%d-%H%M%S).txt"

cat << REPORT > "$REPORT_FILE"
Deployment Report
================
Application: $APP_NAME
Environment: $ENVIRONMENT
Version: $APP_VERSION
Timestamp: $(date)
Deployed by: $(whoami)
Cluster: $(kubectl config current-context)

Resources Created:
$(kubectl get all -n ${APP_NAME}-${ENVIRONMENT} | grep -v "pod/health-check\|pod/perf-test")

Pod Status:
$(kubectl get pods -n ${APP_NAME}-${ENVIRONMENT})

Service Status:
$(kubectl get service ${APP_NAME}-service -n ${APP_NAME}-${ENVIRONMENT})

Ingress Status:
$(kubectl get ingress ${APP_NAME}-ingress -n ${APP_NAME}-${ENVIRONMENT})

HPA Status:
$(kubectl get hpa ${APP_NAME}-hpa -n ${APP_NAME}-${ENVIRONMENT})

Recent Events:
$(kubectl get events -n ${APP_NAME}-${ENVIRONMENT} --sort-by='.lastTimestamp' | tail -10)
```

```
REPORT

echo "📄 Deployment report saved: $REPORT_FILE"

# Cleanup
rm -rf "$TEMP_DIR"

echo ""
echo "🎉 Deployment completed successfully!"
echo ""
echo "📋 Next Steps:"
echo "  Monitor: kubectl get pods -n ${APP_NAME}-${ENVIRONMENT} -w"
echo "  Logs: kubectl logs -f deployment/${APP_NAME} -n ${APP_NAME}-${ENVIRONMENT}"
echo "  Scale: kubectl scale deployment ${APP_NAME} --replicas=5 -n ${APP_NAME}-${ENVIRONMENT}"
echo "  Rollback: kubectl rollout undo deployment/${APP_NAME} -n ${APP_NAME}-${ENVIRONMENT}"
echo ""
echo "🌐 Application URL: https://${APP_NAME}-${ENVIRONMENT}.${DOMAIN}"
EOF

chmod +x deploy-production-app.sh

# Create sample configuration files
cat << 'EOF' > deployment-config-dev.env
# Development Environment Configuration
ENVIRONMENT=development
REPLICAS=1
MIN_REPLICAS=1
MAX_REPLICAS=3
CPU_REQUEST="100m"
CPU_LIMIT="500m"
MEMORY_REQUEST="128Mi"
MEMORY_LIMIT="512Mi"
LOG_LEVEL="debug"
IMAGE_REGISTRY="registry.company.com"
DOMAIN="dev.company.com"
DB_HOST="postgres-dev.database.svc.cluster.local"
DB_PORT="5432"
CACHE_ENABLED="false"
DB_PASSWORD_B64="ZGV2cGFzc3dvcmQ="  # devpassword
API_KEY_B64="ZGV2LWFwaS1rZXk="      # dev-api-key
EOF

cat << 'EOF' > deployment-config-prod.env
# Production Environment Configuration
```

```
ENVIRONMENT=production
REPLICAS=3
MIN_REPLICAS=3
MAX_REPLICAS=10
CPU_REQUEST="500m"
CPU_LIMIT="2000m"
MEMORY_REQUEST="512Mi"
MEMORY_LIMIT="2Gi"
LOG_LEVEL="warn"
IMAGE_REGISTRY="registry.company.com"
DOMAIN="company.com"
DB_HOST="postgres-prod.database.svc.cluster.local"
DB_PORT="5432"
CACHE_ENABLED="true"
DB_PASSWORD_B64="cHJvZHBhc3N3b3Jk"  # prodpassword
API_KEY_B64="cHJvZC1hcGkta2V5"      # prod-api-key
EOF

echo "✅ Production deployment pipeline created!"
echo ""
echo "📁 Files created:"
echo "  - production-app-template.yaml (Kubernetes manifests template)"
echo "  - deploy-production-app.sh (Deployment script)"
echo "  - deployment-config-dev.env (Development configuration)"
echo "  - deployment-config-prod.env (Production configuration)"
echo ""
echo "🚀 Usage Examples:"
echo "  ./deploy-production-app.sh webapp development v1.0.0 deployment-config-dev.env"
echo "  ./deploy-production-app.sh webapp production v1.0.0 deployment-config-prod.env"
```

Mini-Project: Complete Kubernetes Operations Center

Create a comprehensive operations center with monitoring, alerting, and automated remediation:

bash

```
#!/bin/bash
# save as k8s-operations-center.sh
echo "🎛️ Kubernetes Operations Center Setup"

# Create operations center directory structure
mkdir -p k8s-ops-center/{monitoring,alerting,automation,dashboards,reports}
cd k8s-ops-center

# Create comprehensive monitoring system
cat << 'EOF' > monitoring/cluster-monitor.sh
#!/bin/bash

METRICS_DIR="/tmp/k8s-metrics"
ALERT_THRESHOLD_CPU=80
ALERT_THRESHOLD_MEMORY=85
ALERT_THRESHOLD_PODS=90

mkdir -p "$METRICS_DIR"

collect_metrics() {
    local timestamp=$(date +%s)
    local date_str=$(date '+%Y-%m-%d %H:%M:%S')

    echo "📊 Collecting metrics at $date_str"

    # Node metrics
    if kubectl top nodes >/dev/null 2>&1; then
        kubectl top nodes --no-headers > "$METRICS_DIR/nodes-$timestamp.txt"
        kubectl top pods --all-namespaces --no-headers > "$METRICS_DIR/pods-$timestamp.txt"
    fi

    # Cluster state metrics
    kubectl get nodes -o json > "$METRICS_DIR/nodes-state-$timestamp.json"
    kubectl get pods --all-namespaces -o json > "$METRICS_DIR/pods-state-$timestamp.json"
    kubectl get events --all-namespaces -o json > "$METRICS_DIR/events-$timestamp.json"

    # Resource utilization
    cat << METRICS > "$METRICS_DIR/cluster-summary-$timestamp.txt"
Timestamp: $date_str
Nodes: $(kubectl get nodes --no-headers | wc -l)
Total Pods: $(kubectl get pods --all-namespaces --no-headers | wc -l)
Running Pods: $(kubectl get pods --all-namespaces --no-headers | grep Running | wc -l)
Failed Pods: $(kubectl get pods --all-namespaces --no-headers | grep -E "(Error|Failed|CrashLoopBackOff)" | wc -l)
```

```
Services: $(kubectl get services --all-namespaces --no-headers | wc -l)
Deployments: $(kubectl get deployments --all-namespaces --no-headers | wc -l)
METRICS

    # Cleanup old metrics (keep last 24 hours)
    find "$METRICS_DIR" -name "*.txt" -o -name "*.json" | head -n -100 | xargs rm -f 2>/dev/null || true
}

analyze_trends() {
    echo "📈 Analyzing trends..."

    # CPU usage trend
    if ls "$METRICS_DIR"/nodes-*.txt >/dev/null 2>&1; then
        echo "🖥️ Node CPU Usage Trend (last 10 readings):"
        ls -t "$METRICS_DIR"/nodes-*.txt | head -10 | while read file; do
            timestamp=$(basename "$file" .txt | cut -d'-' -f2)
            date_str=$(date -d "@$timestamp" '+%H:%M:%S' 2>/dev/null || echo "$timestamp")
            avg_cpu=$(awk '{sum+=$3} END {if(NR>0) print sum/NR; else print 0}' "$file" | cut -d'%' -f1)
            echo "  $date_str: ${avg_cpu}% average CPU"
        done
    fi
}

generate_alerts() {
    echo "🚨 Checking alert conditions..."

    # Check for high resource usage
    if [ -f "$METRICS_DIR/nodes-$(ls "$METRICS_DIR" | grep "nodes-" | tail -1 | cut -d'-' -f2)" ]; then
        while read node cpu_usage cpu_percent memory_usage memory_percent; do
            cpu_num=$(echo $cpu_percent | tr -d '%')
            mem_num=$(echo $memory_percent | tr -d '%')

            if [ "$cpu_num" -gt "$ALERT_THRESHOLD_CPU" ]; then
                echo "🔥 ALERT: High CPU on $node: $cpu_percent"
                # In production, send to alerting system
            fi

            if [ "$mem_num" -gt "$ALERT_THRESHOLD_MEMORY" ]; then
                echo "💾 ALERT: High memory on $node: $memory_percent"
                # In production, send to alerting system
            fi
        done < "$METRICS_DIR/nodes-$(ls "$METRICS_DIR" | grep "nodes-" | tail -1 | cut -d'-' -f2)"
    fi
}
```

```
# Main monitoring loop
case "$1" in
    "collect")
        collect_metrics
        ;;
    "analyze")
        analyze_trends
        ;;
    "alert")
        generate_alerts
        ;;
    "daemon")
        echo "🔄 Starting monitoring daemon..."
        while true; do
            collect_metrics
            analyze_trends
            generate_alerts
            sleep 60
        done
        ;;
    *)
        echo "Usage: $0 {collect|analyze|alert|daemon}"
        ;;
esac
EOF

chmod +x monitoring/cluster-monitor.sh

# Create automated remediation system
cat << 'EOF' > automation/auto-remediation.sh
#!/bin/bash

LOG_FILE="/tmp/auto-remediation.log"

log_action() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

restart_failing_pods() {
    log_action "🔄 Checking for failing pods..."

    # Find pods in error states
    kubectl get pods --all-namespaces --no-headers | grep -E "(Error|CrashLoopBackOff|ImagePullBackOff)" | while read
```

```
        log_action "🚨 Found failing pod: $namespace/$pod"

        # Get pod age
        age=$(kubectl get pod "$pod" -n "$namespace" -o jsonpath='{.metadata.creationTimestamp}')
        current_time=$(date -u +%s)
        pod_time=$(date -d "$age" +%s 2>/dev/null || echo $current_time)
        age_minutes=$(( (current_time - pod_time) / 60 ))

        # Only restart if pod has been failing for more than 5 minutes
        if [ "$age_minutes" -gt 5 ]; then
            log_action "♻️ Restarting pod $namespace/$pod (age: ${age_minutes}m)"
            kubectl delete pod "$pod" -n "$namespace"

            # Wait and verify
            sleep 30
            new_status=$(kubectl get pod -l app=$(kubectl get pod "$pod" -n "$namespace" -o jsonpath='{.metadata.labe
            log_action "✅ Pod restart result: $new_status"
        else
            log_action "⏳ Pod is new, waiting before restart (age: ${age_minutes}m)"
        fi
    done
}

scale_deployments() {
    log_action "📊 Checking deployment scaling needs..."

    # Check HPA status and manually scale if needed
    kubectl get hpa --all-namespaces --no-headers | while read namespace hpa target min max replicas rest; do
        current_replicas=$(echo $replicas | cut -d'/' -f1)
        max_replicas=$(echo $replicas | cut -d'/' -f2)

        # If at max capacity, log warning
        if [ "$current_replicas" = "$max_replicas" ]; then
            log_action "⚠️ HPA $namespace/$hpa at maximum capacity ($current_replicas/$max_replicas)"
        fi
    done
}

cleanup_completed_jobs() {
    log_action "🧹 Cleaning up completed jobs..."

    # Clean up completed jobs older than 1 hour
    kubectl get jobs --all-namespaces -o json | jq -r '
    .items[] |
```

```
    select(.status.conditions[]?.type == "Complete" and .status.conditions[]?.status == "True") |
    select((now - (.status.completionTime | fromdateiso8601)) > 3600) |
    "\(.metadata.namespace) \(.metadata.name)"
    ' | while read namespace job; do
        log_action "🗑️  Deleting completed job: $namespace/$job"
        kubectl delete job "$job" -n "$namespace"
    done
}

cleanup_failed_pods() {
    log_action "🧹 Cleaning up failed pods..."

    # Clean up pods in Failed state older than 30 minutes
    kubectl get pods --all-namespaces --field-selector=status.phase=Failed -o json | jq -r '
    .items[] |
    select((now - (.metadata.creationTimestamp | fromdateiso8601)) > 1800) |
    "\(.metadata.namespace) \(.metadata.name)"
    ' | while read namespace pod; do
        log_action "🗑️  Deleting failed pod: $namespace/$pod"
        kubectl delete pod "$pod" -n "$namespace"
    done
}

check_disk_pressure() {
    log_action "💾 Checking for disk pressure..."

    # Check nodes for disk pressure
    kubectl get nodes -o json | jq -r '
    .items[] |
    select(.status.conditions[]? | select(.type == "DiskPressure" and .status == "True")) |
    .metadata.name
    ' | while read node; do
        log_action "🚨 Disk pressure detected on node: $node"

        # Trigger cleanup on the node
        kubectl get pods --all-namespaces --field-selector spec.nodeName="$node" -o json | jq -r '
        .items[] |
        select(.status.phase == "Succeeded" or .status.phase == "Failed") |
        "\(.metadata.namespace) \(.metadata.name)"
        ' | head -5 | while read namespace pod; do
            log_action "🧹 Cleaning up pod on pressured node: $namespace/$pod"
            kubectl delete pod "$pod" -n "$namespace"
        done
    done
```

```
}

# Main remediation functions
case "$1" in
    "pods")
        restart_failing_pods
        ;;
    "scale")
        scale_deployments
        ;;
    "jobs")
        cleanup_completed_jobs
        ;;
    "failed")
        cleanup_failed_pods
        ;;
    "disk")
        check_disk_pressure
        ;;
    "all")
        restart_failing_pods
        scale_deployments
        cleanup_completed_jobs
        cleanup_failed_pods
        check_disk_pressure
        ;;
    "daemon")
        log_action "🤖 Starting auto-remediation daemon..."
        while true; do
            restart_failing_pods
            cleanup_completed_jobs
            cleanup_failed_pods
            check_disk_pressure
            sleep 300  # Run every 5 minutes
        done
        ;;
    *)
        echo "🤖 Auto-Remediation System"
        echo "Usage: $0 {pods|scale|jobs|failed|disk|all|daemon}"
        ;;
esac
EOF

chmod +x automation/auto-remediation.sh
```

```
# Create dashboard generator
cat << 'EOF' > dashboards/generate-dashboard.sh
#!/bin/bash

DASHBOARD_DIR="/tmp/k8s-dashboard"
mkdir -p "$DASHBOARD_DIR"

generate_html_dashboard() {
    local dashboard_file="$DASHBOARD_DIR/k8s-dashboard-$(date +%Y%m%d-%H%M%S).html"

    cat << 'HTML' > "$dashboard_file"
<!DOCTYPE html>
<html>
<head>
    <title>Kubernetes Operations Dashboard</title>
    <meta http-equiv="refresh" content="30">
    <style>
        body {
            font-family: 'Segoe UI', Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background: #f5f5f5;
        }
        .header {
            background: #2196F3;
            color: white;
            padding: 20px;
            text-align: center;
            margin: -20px -20px 20px -20px;
        }
        .metrics-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
            gap: 20px;
            margin-bottom: 20px;
        }
        .metric-card {
            background: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }
        .metric-title {
```

```
            font-size: 18px;
            font-weight: bold;
            margin-bottom: 10px;
            color: #333;
        }
        .metric-value {
            font-size: 24px;
            font-weight: bold;
            color: #2196F3;
        }
        .status-good { color: #4CAF50; }
        .status-warning { color: #FF9800; }
        .status-critical { color: #F44336; }
        .resource-table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 10px;
        }
        .resource-table th, .resource-table td {
            padding: 8px;
            text-align: left;
            border-bottom: 1px solid #ddd;
        }
        .resource-table th {
            background: #f0f0f0;
            font-weight: bold;
        }
        pre {
            background: #f8f8f8;
            padding: 10px;
            border-radius: 4px;
            overflow-x: auto;
            font-size: 12px;
        }
        .section {
            background: white;
            margin: 20px 0;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }
    </style>
</head>
<body>
```

```
<div class="header">
    <h1>🎛️ Kubernetes Operations Dashboard</h1>
    <p>Last Updated: TIMESTAMP_PLACEHOLDER</p>
    <p>Cluster: CLUSTER_PLACEHOLDER</p>
</div>

<div class="metrics-grid">
    <div class="metric-card">
        <div class="metric-title">🖥️ Nodes</div>
        <div class="metric-value">NODES_COUNT</div>
        <div>NODES_READY ready</div>
    </div>

    <div class="metric-card">
        <div class="metric-title">📦 Pods</div>
        <div class="metric-value">PODS_TOTAL</div>
        <div class="status-good">PODS_RUNNING running</div>
        <div class="status-critical">PODS_FAILED failed</div>
    </div>

    <div class="metric-card">
        <div class="metric-title">🚀 Deployments</div>
        <div class="metric-value">DEPLOYMENTS_COUNT</div>
        <div>DEPLOYMENTS_READY ready</div>
    </div>

    <div class="metric-card">
        <div class="metric-title">🌐 Services</div>
        <div class="metric-value">SERVICES_COUNT</div>
        <div>Across NAMESPACES_COUNT namespaces</div>
    </div>
</div>

<div class="section">
    <h2>📊 Resource Usage</h2>
    <table class="resource-table">
        <thead>
            <tr>
                <th>Node</th>
                <th>CPU Usage</th>
                <th>Memory Usage</th>
                <th>Status</th>
            </tr>
        </thead>
```

```
        <tbody>
          NODE_USAGE_ROWS
        </tbody>
      </table>
    </div>

    <div class="section">
      <h2>⚠️ Recent Alerts</h2>
      <pre>RECENT_ALERTS</pre>
    </div>

    <div class="section">
      <h2>📋 Recent Events</h2>
      <pre>RECENT_EVENTS</pre>
    </div>

    <div class="section">
      <h2>🔍 Problematic Pods</h2>
      <pre>PROBLEMATIC_PODS</pre>
    </div>

    <div class="section">
      <h2>💾 Storage Status</h2>
      <pre>STORAGE_STATUS</pre>
    </div>
</body>
</html>
HTML

    # Collect current data
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')
    local cluster=$(kubectl config current-context)

    # Basic metrics
    local nodes_count=$(kubectl get nodes --no-headers | wc -l)
    local nodes_ready=$(kubectl get nodes --no-headers | grep -c Ready)
    local pods_total=$(kubectl get pods --all-namespaces --no-headers | wc -l)
    local pods_running=$(kubectl get pods --all-namespaces --no-headers | grep -c Running)
    local pods_failed=$(kubectl get pods --all-namespaces --no-headers | grep -cE "(Error|Failed|CrashLoopBackOff)")
    local deployments_count=$(kubectl get deployments --all-namespaces --no-headers | wc -l)
    local deployments_ready=$(kubectl get deployments --all-namespaces --no-headers | awk '$3==$4 {count++} END
    local services_count=$(kubectl get services --all-namespaces --no-headers | wc -l)
    local namespaces_count=$(kubectl get namespaces --no-headers | wc -l)
```

```
# Resource usage
local node_usage_rows=""
if kubectl top nodes >/dev/null 2>&1; then
    node_usage_rows=$(kubectl top nodes --no-headers | while read node cpu_usage cpu_percent memory_usage m
        local status_class="status-good"
        local cpu_num=$(echo $cpu_percent | tr -d '%')
        local mem_num=$(echo $memory_percent | tr -d '%')

        if [ "$cpu_num" -gt 80 ] || [ "$mem_num" -gt 80 ]; then
            status_class="status-critical"
        elif [ "$cpu_num" -gt 60 ] || [ "$mem_num" -gt 60 ]; then
            status_class="status-warning"
        fi

        echo "<tr><td>$node</td><td class=\"$status_class\">$cpu_percent</td><td class=\"$status_class\">$mem
    done)
else
    node_usage_rows="<tr><td colspan=\"4\">Metrics server not available</td></tr>"
fi

# Recent alerts (from log file)
local recent_alerts=$(tail -10 /tmp/auto-remediation.log 2>/dev/null | grep ALERT || echo "No recent alerts")

# Recent events
local recent_events=$(kubectl get events --all-namespaces --sort-by='.lastTimestamp' | tail -10)

# Problematic pods
local problematic_pods=$(kubectl get pods --all-namespaces | grep -E "(Error|Failed|CrashLoopBackOff|Pending)" || e

# Storage status
local storage_status=$(kubectl get pv,pvc --all-namespaces 2>/dev/null || echo "No storage resources found")

# Replace placeholders
sed -i "s/TIMESTAMP_PLACEHOLDER/$timestamp/g" "$dashboard_file"
sed -i "s/CLUSTER_PLACEHOLDER/$cluster/g" "$dashboard_file"
sed -i "s/NODES_COUNT/$nodes_count/g" "$dashboard_file"
sed -i "s/NODES_READY/$nodes_ready/g" "$dashboard_file"
sed -i "s/PODS_TOTAL/$pods_total/g" "$dashboard_file"
sed -i "s/PODS_RUNNING/$pods_running/g" "$dashboard_file"
sed -i "s/PODS_FAILED/$pods_failed/g" "$dashboard_file"
sed -i "s/DEPLOYMENTS_COUNT/$deployments_count/g" "$dashboard_file"
sed -i "s/DEPLOYMENTS_READY/$deployments_ready/g" "$dashboard_file"
sed -i "s/SERVICES_COUNT/$services_count/g" "$dashboard_file"
sed -i "s/NAMESPACES_COUNT/$namespaces_count/g" "$dashboard_file"
```

```
    sed -i "s|NODE_USAGE_ROWS|$node_usage_rows|g" "$dashboard_file"

    # Handle multi-line replacements
    echo "$recent_alerts" > /tmp/alerts.txt
    echo "$recent_events" > /tmp/events.txt
    echo "$problematic_pods" > /tmp/pods.txt
    echo "$storage_status" > /tmp/storage.txt

    sed -i "/RECENT_ALERTS/r /tmp/alerts.txt" "$dashboard_file"
    sed -i "s/RECENT_ALERTS//g" "$dashboard_file"

    sed -i "/RECENT_EVENTS/r /tmp/events.txt" "$dashboard_file"
    sed -i "s/RECENT_EVENTS//g" "$dashboard_file"

    sed -i "/PROBLEMATIC_PODS/r /tmp/pods.txt" "$dashboard_file"
    sed -i "s/PROBLEMATIC_PODS//g" "$dashboard_file"

    sed -i "/STORAGE_STATUS/r /tmp/storage.txt" "$dashboard_file"
    sed -i "s/STORAGE_STATUS//g" "$dashboard_file"

    # Clean up temp files
    rm -f /tmp/alerts.txt /tmp/events.txt /tmp/pods.txt /tmp/storage.txt

    echo "📊 Dashboard generated: $dashboard_file"

    # Create symlink to latest
    ln -sf "$dashboard_file" "$DASHBOARD_DIR/latest.html"
    echo "🔗 Latest dashboard: $DASHBOARD_DIR/latest.html"
}

# Main execution
case "$1" in
    "generate")
        generate_html_dashboard
        ;;
    "serve")
        generate_html_dashboard
        echo "🌐 Starting simple HTTP server on port 8080..."
        cd "$DASHBOARD_DIR"
        python3 -m http.server 8080 2>/dev/null || python -m SimpleHTTPServer 8080
        ;;
    "auto")
        echo "🔄 Starting auto-refresh dashboard..."
        while true; do
```

```
        generate_html_dashboard
        sleep 30
      done
      ;;
  *)
      echo "📊 Dashboard Generator"
      echo "Usage: $0 {generate|serve|auto}"
      echo ""
      echo "Commands:"
      echo "  generate - Generate dashboard once"
      echo "  serve    - Generate and serve on HTTP port 8080"
      echo "  auto     - Auto-refresh every 30 seconds"
      ;;
esac
EOF

chmod +x dashboards/generate-dashboard.sh

# Create report generator
cat << 'EOF' > reports/generate-reports.sh
#!/bin/bash

REPORTS_DIR="/tmp/k8s-reports"
mkdir -p "$REPORTS_DIR"

generate_security_report() {
  local report_file="$REPORTS_DIR/security-report-$(date +%Y%m%d).txt"

  cat << REPORT > "$report_file"
Kubernetes Security Report
==========================
Generated: $(date)
Cluster: $(kubectl config current-context)

1. Pod Security Analysis
------------------------
REPORT

  echo "Pods running as root:" >> "$report_file"
  kubectl get pods --all-namespaces -o json | jq -r '
  .items[] |
  select(.spec.securityContext.runAsUser == 0 or .spec.containers[].securityContext.runAsUser == 0) |
  "\(.metadata.namespace)/\(.metadata.name)"
  ' >> "$report_file" || echo "None found" >> "$report_file"
```

```
    echo -e "\nPods with privileged containers:" >> "$report_file"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.spec.containers[].securityContext.privileged == true) |
    "\(.metadata.namespace)/\(.metadata.name)"
    ' >> "$report_file" || echo "None found" >> "$report_file"

    echo -e "\n2. RBAC Analysis" >> "$report_file"
    echo "----------------" >> "$report_file"
    echo "Service Accounts:" >> "$report_file"
    kubectl get serviceaccounts --all-namespaces | wc -l >> "$report_file"

    echo -e "\nCluster Roles:" >> "$report_file"
    kubectl get clusterroles | wc -l >> "$report_file"

    echo -e "\n3. Network Policies" >> "$report_file"
    echo "--------------------" >> "$report_file"
    kubectl get networkpolicies --all-namespaces >> "$report_file" 2>/dev/null || echo "No network policies found" >> "

    echo -e "\n4. Secrets Analysis" >> "$report_file"
    echo "--------------------" >> "$report_file"
    echo "Total secrets:" >> "$report_file"
    kubectl get secrets --all-namespaces | wc -l >> "$report_file"

    echo "✅ Security report generated: $report_file"
}

generate_performance_report() {
    local report_file="$REPORTS_DIR/performance-report-$(date +%Y%m%d).txt"

    cat << REPORT > "$report_file"
Kubernetes Performance Report
==============================
Generated: $(date)
Cluster: $(kubectl config current-context)

1. Resource Utilization
-----------------------
REPORT

    if kubectl top nodes >/dev/null 2>&1; then
        echo "Node Resource Usage:" >> "$report_file"
        kubectl top nodes >> "$report_file"
```

```
        echo -e "\nTop CPU Consuming Pods:" >> "$report_file"
        kubectl top pods --all-namespaces --sort-by=cpu | head -10 >> "$report_file"

        echo -e "\nTop Memory Consuming Pods:" >> "$report_file"
        kubectl top pods --all-namespaces --sort-by=memory | head -10 >> "$report_file"
    else
        echo "Metrics server not available" >> "$report_file"
    fi

    echo -e "\n2. Pod Restart Analysis" >> "$report_file"
    echo "-----------------------" >> "$report_file"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.status.containerStatuses[]?.restartCount > 0) |
    "\(.metadata.namespace)/\(.metadata.name): \(.status.containerStatuses[0].restartCount) restarts"
    ' >> "$report_file"

    echo -e "\n3. Storage Performance" >> "$report_file"
    echo "----------------------" >> "$report_file"
    kubectl get pv -o custom-columns=NAME:.metadata.name,CAPACITY:.spec.capacity.storage,STATUS:.status.phase >>

    echo "✅ Performance report generated: $report_file"
}

generate_inventory_report() {
    local report_file="$REPORTS_DIR/inventory-report-$(date +%Y%m%d).txt"

    cat << REPORT > "$report_file"
Kubernetes Inventory Report
===========================
Generated: $(date)
Cluster: $(kubectl config current-context)

1. Cluster Summary
------------------
Nodes: $(kubectl get nodes --no-headers | wc -l)
Namespaces: $(kubectl get namespaces --no-headers | wc -l)
Pods: $(kubectl get pods --all-namespaces --no-headers | wc -l)
Services: $(kubectl get services --all-namespaces --no-headers | wc -l)
Deployments: $(kubectl get deployments --all-namespaces --no-headers | wc -l)
ConfigMaps: $(kubectl get configmaps --all-namespaces --no-headers | wc -l)
Secrets: $(kubectl get secrets --all-namespaces --no-headers | wc -l)
```

```
2. Node Details
---------------
REPORT

    kubectl get nodes -o wide >> "$report_file"

    echo -e "\n3. Namespace Resource Distribution" >> "$report_file"
    echo "-----------------------------------" >> "$report_file"
    kubectl get pods --all-namespaces --no-headers | awk '{ns[$1]++} END {for(n in ns) print n": "ns[n]" pods"}' | sort >>

    echo -e "\n4. Image Analysis" >> "$report_file"
    echo "-----------------" >> "$report_file"
    kubectl get pods --all-namespaces -o json | jq -r '.items[].spec.containers[].image' | sort | uniq -c | sort -nr | head -20 >

    echo "✅ Inventory report generated: $report_file"
}

generate_compliance_report() {
    local report_file="$REPORTS_DIR/compliance-report-$(date +%Y%m%d).txt"

    cat << REPORT > "$report_file"
Kubernetes Compliance Report
============================
Generated: $(date)
Cluster: $(kubectl config current-context)

1. Resource Limits Compliance
-----------------------------
REPORT

    echo "Pods without resource limits:" >> "$report_file"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.spec.containers[0].resources.limits == null) |
    "\(.metadata.namespace)/\(.metadata.name)"
    ' >> "$report_file"

    echo -e "\n2. Security Context Compliance" >> "$report_file"
    echo "------------------------------" >> "$report_file"
    echo "Pods without security context:" >> "$report_file"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.spec.securityContext == null and .spec.containers[].securityContext == null) |
    "\(.metadata.namespace)/\(.metadata.name)"
```

```
    ' >> "$report_file"

    echo -e "\n3. Label Compliance" >> "$report_file"
    echo "--------------------" >> "$report_file"
    echo "Resources without recommended labels:" >> "$report_file"
    kubectl get deployments --all-namespaces -o json | jq -r '
    .items[] |
    select(.metadata.labels.app == null or .metadata.labels.version == null) |
    "\(.metadata.namespace)/\(.metadata.name)"
    ' >> "$report_file"

    echo "✅ Compliance report generated: $report_file"
}

# Main execution
case "$1" in
    "security")
        generate_security_report
        ;;
    "performance")
        generate_performance_report
        ;;
    "inventory")
        generate_inventory_report
        ;;
    "compliance")
        generate_compliance_report
        ;;
    "all")
        generate_security_report
        generate_performance_report
        generate_inventory_report
        generate_compliance_report
        echo "📊 All reports generated in: $REPORTS_DIR"
        ;;
    *)
        echo "📊 Report Generator"
        echo "Usage: $0 {security|performance|inventory|compliance|all}"
        echo ""
        echo "Commands:"
        echo "  security     - Generate security analysis report"
        echo "  performance  - Generate performance analysis report"
        echo "  inventory    - Generate cluster inventory report"
        echo "  compliance   - Generate compliance check report"
```

```
        echo "  all          - Generate all reports"
        ;;
esac
EOF

chmod +x reports/generate-reports.sh

# Create master operations script
cat << 'EOF' > k8s-ops.sh
#!/bin/bash

OPS_CENTER_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

show_menu() {
    echo "🎛️ Kubernetes Operations Center"
    echo "================================"
    echo ""
    echo "Monitoring:"
    echo "  1) Start monitoring daemon"
    echo "  2) Collect metrics once"
    echo "  3) Analyze trends"
    echo ""
    echo "Automation:"
    echo "  4) Start auto-remediation daemon"
    echo "  5) Run remediation once"
    echo "  6) Clean up resources"
    echo ""
    echo "Dashboard:"
    echo "  7) Generate dashboard"
    echo "  8) Serve dashboard (HTTP)"
    echo "  9) Auto-refresh dashboard"
    echo ""
    echo "Reports:"
    echo "  10) Generate security report"
    echo "  11) Generate performance report"
    echo "  12) Generate all reports"
    echo ""
    echo "  0) Exit"
    echo ""
}

execute_choice() {
    case $1 in
        1)
```

```
    echo "🔄 Starting monitoring daemon..."
    "$OPS_CENTER_DIR/monitoring/cluster-monitor.sh" daemon
    ;;
2)
    echo "📊 Collecting metrics..."
    "$OPS_CENTER_DIR/monitoring/cluster-monitor.sh" collect
    ;;
3)
    echo "📈 Analyzing trends..."
    "$OPS_CENTER_DIR/monitoring/cluster-monitor.sh" analyze
    ;;
4)
    echo "🤖 Starting auto-remediation daemon..."
    "$OPS_CENTER_DIR/automation/auto-remediation.sh" daemon
    ;;
5)
    echo "⚡ Running remediation..."
    "$OPS_CENTER_DIR/automation/auto-remediation.sh" all
    ;;
6)
    echo "🧹 Cleaning up resources..."
    "$OPS_CENTER_DIR/automation/auto-remediation.sh" jobs
    "$OPS_CENTER_DIR/automation/auto-remediation.sh" failed
    ;;
7)
    echo "📊 Generating dashboard..."
    "$OPS_CENTER_DIR/dashboards/generate-dashboard.sh" generate
    ;;
8)
    echo "🌐 Starting dashboard server..."
    "$OPS_CENTER_DIR/dashboards/generate-dashboard.sh" serve
    ;;
9)
    echo "🔄 Starting auto-refresh dashboard..."
    "$OPS_CENTER_DIR/dashboards/generate-dashboard.sh" auto
    ;;
10)
    echo "🔒 Generating security report..."
    "$OPS_CENTER_DIR/reports/generate-reports.sh" security
    ;;
11)
    echo "⚡ Generating performance report..."
    "$OPS_CENTER_DIR/reports/generate-reports.sh" performance
    ;;
```

```
        12)
            echo "📊 Generating all reports..."
            "$OPS_CENTER_DIR/reports/generate-reports.sh" all
            ;;
        0)
            echo "👋 Goodbye!"
            exit 0
            ;;
        *)
            echo "❌ Invalid choice. Please try again."
            ;;
    esac
}

# Interactive mode
if [ "$1" = "" ]; then
    while true; do
        clear
        show_menu
        read -p "Enter your choice [0-12]: " choice
        echo ""
        execute_choice "$choice"
        echo ""
        read -p "Press Enter to continue..."
    done
else
    # Direct command mode
    execute_choice "$1"
fi
EOF

chmod +x k8s-ops.sh

# Create README for the operations center
cat << 'EOF' > README.md
# ☸️ Kubernetes Operations Center

A comprehensive operations center for monitoring, managing, and maintaining Kubernetes clusters.

## Features

### 🔍 Monitoring
- Real-time cluster metrics collection
- Resource usage trend analysis
```

- Automated alerting for threshold breaches
- Historical data retention

🤖 Automation
- Auto-remediation of failing pods
- Cleanup of completed jobs and failed resources
- Disk pressure management
- Scaling recommendations

📊 Dashboards
- Real-time HTML dashboard
- Resource utilization visualization
- Alert status display
- Auto-refreshing capabilities

📋 Reports
- Security compliance analysis
- Performance assessment
- Resource inventory
- Compliance checking

Quick Start

```bash
# Interactive mode
./k8s-ops.sh

# Direct commands
./k8s-ops.sh 1    # Start monitoring daemon
./k8s-ops.sh 4    # Start auto-remediation
./k8s-ops.sh 7    # Generate dashboard
./k8s-ops.sh 12   # Generate all reports
```

Directory Structure

```
k8s-ops-center/
├── monitoring/          # Monitoring scripts
├── automation/          # Auto-remediation scripts
├── dashboards/          # Dashboard generation
├── reports/             # Report generators
├── k8s-ops.sh           # Main operations script
└── README.md            # This file
```

Components

Monitoring System

- **cluster-monitor.sh**: Collects metrics and analyzes trends
- Stores metrics in `/tmp/k8s-metrics/`
- Provides alerting based on configurable thresholds

Auto-Remediation

- **auto-remediation.sh**: Automated problem resolution
- Restarts failing pods
- Cleans up resources
- Manages disk pressure

Dashboard

- **generate-dashboard.sh**: Creates HTML dashboards
- Real-time cluster status
- Resource utilization graphs
- Serves via HTTP on port 8080

Reports

- **generate-reports.sh**: Comprehensive reporting
- Security analysis
- Performance metrics
- Compliance checking
- Resource inventory

Configuration

Edit the scripts to customize:

- Alert thresholds
- Cleanup intervals
- Report formats
- Dashboard refresh rates

Dependencies

- kubectl configured and working
- jq for JSON processing
- Python for HTTP server
- Standard Unix tools (awk, sed, grep)

Best Practices

1. Run monitoring and auto-remediation as background services
2. Review reports regularly for trends
3. Customize thresholds based on your environment
4. Integrate with external alerting systems
5. Regular backup of configurations and data

Troubleshooting

- Check kubectl connectivity: `kubectl cluster-info`
- Verify metrics server: `kubectl top nodes`
- Check log files in `/tmp/` directory
- Ensure proper RBAC permissions

Support

For issues and improvements, check the script comments and customize for your environment.
EOF

cd ..

echo "✅ Kubernetes Operations Center Setup Complete!"
echo ""
echo "📁 Directory Structure:"
find k8s-ops-center -type f | head -15

echo ""
echo "🚀 Getting Started:"
echo " cd k8s-ops-center"
echo " ./k8s-ops.sh   # Interactive mode"

```
echo "  ./k8s-ops.sh 7              # Generate dashboard"
echo "  ./k8s-ops.sh 12             # Generate all reports"

echo ""
echo "🎛️ Operations Center Features:"
echo "  ✅ Real-time monitoring"
echo "  ✅ Auto-remediation"
echo "  ✅ Interactive dashboards"
echo "  ✅ Comprehensive reporting"
echo "  ✅ Security analysis"
echo "  ✅ Performance monitoring"

echo ""
echo "📖 See k8s-ops-center/README.md for detailed documentation"
```

Conclusion

This comprehensive Kubernetes CLI guide has covered everything from basic concepts to advanced production operations. You now have:

🎯 **Core Skills**
- Essential kubectl commands and workflows
- Pod, Service, and Deployment management
- Storage and networking configuration
- Security with RBAC and Secrets

🚀 **Advanced Capabilities**
- Helm chart development and management
- Multi-environment deployment strategies
- Resource optimization and monitoring
- Automated troubleshooting and remediation

🏭 **Production-Ready Tools**
- Complete CI/CD pipeline templates
- Operations center with monitoring and alerting
- Security and compliance reporting
- Performance analysis and optimization

📚 **Best Practices**
- Declarative configuration management
- Resource quotas and limits
- Health checks and monitoring
- Backup and disaster recovery

The examples, demos, and mini-projects provide hands-on experience with real-world scenarios you'll encounter in production Kubernetes environments. Each section builds upon previous knowledge while introducing new concepts and tools.

Remember to:
- Always test in development environments first
- Implement proper RBAC and security measures
- Monitor resource usage and set appropriate limits
- Maintain regular backups and documentation
- Stay updated with Kubernetes releases and best practices

This guide serves as both a learning resource and a reference for ongoing Kubernetes operations. The scripts and tools provided can be customized and extended to meet your specific requirements and environments.5432

```
        name: postgres
      volumeMounts:
      - name: postgres-data
        mountPath: /var/lib/postgresql/data
      - name: config
        mountPath: /scripts
      lifecycle:
        postStart:
          exec:
            command:
            - /bin/bash
            - -c
            - |
              if [[ "${POD_NAME}" == "postgres-0" ]]; then
                echo "Setting up as primary"
                kubectl label pod ${POD_NAME} role=primary --overwrite
                /scripts/primary_init.sh
              else
                echo "Setting up as replica"
                kubectl label pod ${POD_NAME} role=replica --overwrite
                /scripts/replica_init.sh
              fi
      resources:
        requests:
          memory: 256Mi
          cpu: 250m
        limits:
          memory: 512Mi
          cpu: 500m
    volumes:
    - name: config
      configMap:
        name: postgres-config
        defaultMode: 0755
  volumeClaimTemplates:
  - metadata:
      name: postgres-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 5Gi
EOF
```

```
echo "⏳ Waiting for StatefulSet deployment..."
kubectl wait --for=condition=ready pod -l app=postgres --timeout=300s

echo "🔍 Checking cluster status:"
kubectl get pods -l app=postgres --show-labels
kubectl get pvc -l app=postgres

echo "💉 Testing database cluster:"
# Test primary database
echo "Testing primary (postgres-0):"
kubectl exec postgres-0 -- psql -U postgres -c "CREATE TABLE IF NOT EXISTS test_data (id SERIAL, data TEXT, created_at TIMESTAMP DEFAULT NOW());"
kubectl exec postgres-0 -- psql -U postgres -c "INSERT INTO test_data (data) VALUES ('Primary data entry');"

# Test replica access (after replication setup)
echo "Testing replica (postgres-1):"
kubectl exec postgres-1 -- psql -U postgres -c "SELECT * FROM test_data;" 2>/dev/null || echo "Replica not ready yet"

echo "✅ Distributed database cluster deployed!"
echo "📊 Monitor with:"
echo "  kubectl logs -f postgres-0"
echo "  kubectl exec postgres-0 -- psql -U postgres -c 'SELECT * FROM pg_stat_replication;'"
echo "🧹 Cleanup:"
echo "  kubectl delete statefulset postgres"
echo "  kubectl delete service postgres-headless postgres-primary"
echo "  kubectl delete configmap postgres-config"
echo "  kubectl delete pvc -l app=postgres"
```

18. Resource Management

Resource management ensures efficient cluster utilization and prevents resource starvation. Understanding requests, limits, and quotas is crucial for production deployments.

18.1 Resource Requests and Limits

```yaml
```

```yaml
# resource-management-example.yaml
apiVersion: v1
kind: Pod
metadata:
  name: resource-demo
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:  # Minimum resources guaranteed to container
        memory: "64Mi"    # 64 Megabytes of RAM
        cpu: "250m"       # 250 millicores (0.25 CPU cores)
      limits:    # Maximum resources container can use
        memory: "128Mi"   # Container killed if it exceeds this
        cpu: "500m"       # Container throttled if it exceeds this
      # Best Practice: Always set both requests and limits
      # Requests help scheduler make placement decisions
      # Limits prevent containers from consuming excessive resources
```

Example: Quality of Service Classes

Understanding how Kubernetes assigns QoS classes based on resource specifications:

```bash
```

```
#!/bin/bash
# save as qos-demo.sh
echo "🎯 Quality of Service Classes Demo"

# Create pods with different QoS classes

# Guaranteed QoS - requests = limits
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: guaranteed-pod
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        memory: "256Mi"
        cpu: "500m"
      limits:
        memory: "256Mi"
        cpu: "500m"
EOF

# Burstable QoS - requests < limits or only requests specified
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: burstable-pod
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        memory: "128Mi"
        cpu: "250m"
      limits:
        memory: "256Mi"
        cpu: "500m"
EOF
```

```
# BestEffort QoS - no requests or limits specified
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: besteffort-pod
spec:
  containers:
  - name: app
    image: nginx
EOF

echo "⏳ Waiting for pods to be ready..."
kubectl wait --for=condition=ready pod --all --timeout=60s

echo "🔍 Checking QoS classes:"
for pod in guaranteed-pod burstable-pod besteffort-pod; do
    qos=$(kubectl get pod $pod -o jsonpath='{.status.qosClass}')
    echo "$pod: $qos"
done

echo "📊 Resource allocation:"
kubectl describe pods | grep -A 10 -B 2 "QoS Class"

echo "💡 QoS Class Implications:"
echo "  Guaranteed: Highest priority, last to be evicted"
echo "  Burstable: Medium priority, evicted before Guaranteed"
echo "  BestEffort: Lowest priority, first to be evicted"

echo "🧹 Cleanup:"
echo "kubectl delete pod guaranteed-pod burstable-pod besteffort-pod"
```

Demo: Resource Monitoring and Alerting

Create a comprehensive resource monitoring system:

bash

```bash
#!/bin/bash
# save as resource-monitor.sh
NAMESPACE=${1:-default}
THRESHOLD_CPU=80
THRESHOLD_MEMORY=80

echo "📊 Resource Monitoring for namespace: $NAMESPACE"

# Function to check if metrics-server is available
check_metrics_server() {
    if ! kubectl top nodes >/dev/null 2>&1; then
        echo "❌ Metrics server not available. Installing..."
        kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
        echo "⏳ Waiting for metrics server to start..."
        kubectl wait --for=condition=available deployment metrics-server -n kube-system --timeout=120s
    fi
}

# Function to analyze resource usage
analyze_resources() {
    echo "🔍 Current Resource Usage:"

    # Node-level analysis
    echo "--- Node Resources ---"
    kubectl top nodes

    # Pod-level analysis
    echo -e "\n--- Pod Resources in $NAMESPACE ---"
    kubectl top pods -n $NAMESPACE --sort-by=cpu

    # Check for resource-constrained pods
    echo -e "\n🚨 Resource Alerts:"

    # Get pod resource usage and compare with limits
    kubectl get pods -n $NAMESPACE -o custom-columns=NAME:.metadata.name,CPU_REQUEST:.spec.containers[0].res

    # Check for pods without resource specifications
    echo -e "\n⚠️  Pods without resource limits:"
    kubectl get pods -n $NAMESPACE -o json | jq -r '.items[] | select(.spec.containers[0].resources.limits == null) | .metada

    # Check for pending pods due to resource constraints
    echo -e "\n🔒 Pending Pods (Resource Issues):"
    kubectl get pods -n $NAMESPACE --field-selector=status.phase=Pending -o custom-columns=NAME:.metadata.nam
```

```
}

# Function to generate resource recommendations
generate_recommendations() {
    echo -e "\n💡 Resource Optimization Recommendations:"

    # Check for over-provisioned pods (low usage vs high limits)
    echo "1. Check for over-provisioned resources:"
    echo "   kubectl top pods -n $NAMESPACE --sort-by=memory"
    echo "   Compare with limits using: kubectl describe pods -n $NAMESPACE"

    # Check for under-provisioned pods (high usage, low limits)
    echo "2. Monitor for CPU throttling:"
    echo "   kubectl get --raw '/api/v1/nodes/{node-name}/proxy/stats/summary' | jq '.pods[]'"

    # Resource quota recommendations
    echo "3. Consider implementing resource quotas:"
    echo "   kubectl create quota namespace-quota --hard=requests.cpu=4,requests.memory=8Gi,limits.cpu=8,limits.me
}

# Main execution
check_metrics_server
analyze_resources
generate_recommendations

# Create a resource monitoring pod
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: resource-monitor
  namespace: $NAMESPACE
  labels:
    app: resource-monitor
spec:
  containers:
  - name: monitor
    image: busybox
    command: ["sh", "-c"]
    args:
    - |
      while true; do
        echo "$(date): Monitoring resources..."
        # Simple resource usage simulation
```

```
      dd if=/dev/zero of=/tmp/testfile bs=1M count=50 2>/dev/null
      rm /tmp/testfile
      sleep 60
    done
  resources:
    requests:
      memory: "64Mi"
      cpu: "100m"
    limits:
      memory: "128Mi"
      cpu: "200m"
 restartPolicy: Always
EOF

echo "📈 Resource monitor pod created. Check with:"
echo "  kubectl logs -f resource-monitor -n $NAMESPACE"
echo "  kubectl top pod resource-monitor -n $NAMESPACE"
```

Mini-Project: Automated Resource Right-Sizing

Create a system that analyzes resource usage and suggests optimal resource specifications:

bash

```bash
#!/bin/bash
# save as resource-rightsizing.sh
NAMESPACE=${1:-default}
ANALYSIS_PERIOD=${2:-7} # days

echo "🎯 Automated Resource Right-Sizing Analysis"
echo "Namespace: $NAMESPACE"
echo "Analysis Period: $ANALYSIS_PERIOD days"

# Create a comprehensive resource analysis tool
cat << 'EOF' > analyze-resources.py
#!/usr/bin/env python3
import subprocess
import json
import sys
from datetime import datetime, timedelta

def run_kubectl(cmd):
    """Execute kubectl command and return output"""
    try:
        result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
        return result.stdout if result.returncode == 0 else None
    except Exception as e:
        print(f"Error running command: {e}")
        return None

def get_pod_resources(namespace):
    """Get current resource specifications for all pods"""
    cmd = f"kubectl get pods -n {namespace} -o json"
    output = run_kubectl(cmd)
    if not output:
        return []

    pods_data = json.loads(output)
    pod_resources = []

    for pod in pods_data['items']:
        if pod['status']['phase'] != 'Running':
            continue

        pod_info = {
            'name': pod['metadata']['name'],
            'namespace': pod['metadata']['namespace'],
```

```
            'containers': []
        }

        for container in pod['spec']['containers']:
            container_info = {
                'name': container['name'],
                'requests': container.get('resources', {}).get('requests', {}),
                'limits': container.get('resources', {}).get('limits', {})
            }
            pod_info['containers'].append(container_info)

        pod_resources.append(pod_info)

    return pod_resources

def get_resource_usage(namespace):
    """Get current resource usage for pods"""
    cmd = f"kubectl top pods -n {namespace} --no-headers"
    output = run_kubectl(cmd)
    if not output:
        return {}

    usage = {}
    for line in output.strip().split('\n'):
        if line:
            parts = line.split()
            if len(parts) >= 3:
                pod_name = parts[0]
                cpu_usage = parts[1]
                memory_usage = parts[2]
                usage[pod_name] = {
                    'cpu': cpu_usage,
                    'memory': memory_usage
                }

    return usage

def convert_cpu_to_millicores(cpu_str):
    """Convert CPU string to millicores"""
    if not cpu_str:
        return 0
    if cpu_str.endswith('m'):
        return int(cpu_str[:-1])
    else:
```

```
        return int(float(cpu_str) * 1000)

def convert_memory_to_mi(memory_str):
    """Convert memory string to Mi"""
    if not memory_str:
        return 0
    if memory_str.endswith('Mi'):
        return int(memory_str[:-2])
    elif memory_str.endswith('Gi'):
        return int(memory_str[:-2]) * 1024
    elif memory_str.endswith('Ki'):
        return int(memory_str[:-2]) // 1024
    else:
        return int(memory_str) // (1024 * 1024)

def analyze_and_recommend(namespace):
    """Main analysis function"""
    print(f"🔍 Analyzing resources for namespace: {namespace}")

    pod_resources = get_pod_resources(namespace)
    current_usage = get_resource_usage(namespace)

    recommendations = []

    for pod in pod_resources:
        pod_name = pod['name']
        usage = current_usage.get(pod_name, {})

        if not usage:
            continue

        for container in pod['containers']:
            container_name = container['name']
            requests = container['requests']
            limits = container['limits']

            # Current usage
            current_cpu = convert_cpu_to_millicores(usage.get('cpu', '0'))
            current_memory = convert_memory_to_mi(usage.get('memory', '0'))

            # Current limits
            limit_cpu = convert_cpu_to_millicores(limits.get('cpu', '0'))
            limit_memory = convert_memory_to_mi(limits.get('memory', '0'))
```

```
            # Current requests
            request_cpu = convert_cpu_to_millicores(requests.get('cpu', '0'))
            request_memory = convert_memory_to_mi(requests.get('memory', '0'))

            # Calculate recommendations (usage * 1.2 for headroom)
            recommended_cpu = max(current_cpu * 1.2, 100)  # minimum 100m
            recommended_memory = max(current_memory * 1.2, 64)  # minimum 64Mi

            recommendation = {
                'pod': pod_name,
                'container': container_name,
                'current_usage': {
                    'cpu': f"{current_cpu}m",
                    'memory': f"{current_memory}Mi"
                },
                'current_limits': {
                    'cpu': f"{limit_cpu}m" if limit_cpu else "none",
                    'memory': f"{limit_memory}Mi" if limit_memory else "none"
                },
                'recommended': {
                    'cpu_request': f"{int(recommended_cpu)}m",
                    'cpu_limit': f"{int(recommended_cpu * 1.5)}m",
                    'memory_request': f"{int(recommended_memory)}Mi",
                    'memory_limit': f"{int(recommended_memory * 1.5)}Mi"
                }
            }

            recommendations.append(recommendation)

    return recommendations

def generate_yaml(recommendations):
    """Generate YAML patches for resource recommendations"""
    print("\n📝 Generated Resource Patches:")

    for rec in recommendations:
        print(f"\n# Patch for {rec['pod']} - {rec['container']}")
        print(f"kubectl patch pod {rec['pod']} -p '{{")
        print(f'  "spec": {{')
        print(f'    "containers": [{{')
        print(f'      "name": "{rec["container"]}",')
        print(f'      "resources": {{')
        print(f'        "requests": {{')
        print(f'          "cpu": "{rec["recommended"]["cpu_request"]}",')
```

```
        print(f'          "memory": "{rec["recommended"]["memory_request"]}"')
        print(f'        }},')
        print(f'        "limits": {{')
        print(f'          "cpu": "{rec["recommended"]["cpu_limit"]}",')
        print(f'          "memory": "{rec["recommended"]["memory_limit"]}"')
        print(f'        }}')
        print(f'      }}')
        print(f'    }}]')
        print(f'  }}')
        print(f"}}'")

if __name__ == "__main__":
    namespace = sys.argv[1] if len(sys.argv) > 1 else "default"

    recommendations = analyze_and_recommend(namespace)

    print("\n📊 Resource Right-Sizing Recommendations:")
    print("-" * 80)

    for rec in recommendations:
        print(f"Pod: {rec['pod']} | Container: {rec['container']}")
        print(f"  Current Usage: CPU={rec['current_usage']['cpu']}, Memory={rec['current_usage']['memory']}")
        print(f"  Current Limits: CPU={rec['current_limits']['cpu']}, Memory={rec['current_limits']['memory']}")
        print(f"  Recommended: CPU Request={rec['recommended']['cpu_request']}, Limit={rec['recommended']['cpu_limit'
        print(f"               Memory Request={rec['recommended']['memory_request']}, Limit={rec['recommended']['memory_
        print()

    generate_yaml(recommendations)
EOF

chmod +x analyze-resources.py

# Run the analysis
echo "🚀 Running resource analysis..."
python3 analyze-resources.py $NAMESPACE

# Create deployment with various resource patterns for testing
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resource-test-app
  namespace: $NAMESPACE
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: resource-test
  template:
    metadata:
      labels:
        app: resource-test
    spec:
      containers:
      - name: web
        image: nginx:alpine
        resources:
          requests:
            memory: "64Mi"
            cpu: "100m"
          limits:
            memory: "128Mi"
            cpu: "200m"
      - name: sidecar
        image: busybox
        command: ["sh", "-c", "while true; do echo 'sidecar running'; sleep 30; done"]
        resources:
          requests:
            memory: "32Mi"
            cpu: "50m"
          limits:
            memory: "64Mi"
            cpu: "100m"
EOF

echo "✅ Resource right-sizing analysis complete!"
echo "📈 Monitor resource usage over time with:"
echo "  watch kubectl top pods -n $NAMESPACE"
echo "🧹 Cleanup:"
echo "  kubectl delete deployment resource-test-app -n $NAMESPACE"
echo "  rm analyze-resources.py"
```

19. Helm Basics

Helm is the package manager for Kubernetes, allowing you to define, install, and upgrade complex Kubernetes applications using templates and charts.

19.1 Installing and Using Helm

bash

```
# Install Helm (if not already installed)
curl https://get.helm.sh/helm-v3.12.0-linux-amd64.tar.gz | tar xz
sudo mv linux-amd64/helm /usr/local/bin/

# Verify Helm installation
helm version

# Add popular Helm repositories
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add stable https://charts.helm.sh/stable
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

# Update repository information
helm repo update

# Search for charts
helm search repo nginx
helm search repo database

# Install a chart
helm install my-nginx bitnami/nginx
# Creates a release named "my-nginx" using the bitnami/nginx chart

# List installed releases
helm list
helm list --all-namespaces

# Get information about a release
helm status my-nginx
helm get values my-nginx
helm get manifest my-nginx

# Upgrade a release
helm upgrade my-nginx bitnami/nginx --set service.type=NodePort

# Rollback a release
helm rollback my-nginx 1

# Uninstall a release
helm uninstall my-nginx
```

Example: Creating Your First Helm Chart

Let's create a custom Helm chart for a web application:

bash

```
#!/bin/bash
# save as create-helm-chart.sh
CHART_NAME=${1:-webapp}

echo "📦 Creating Helm Chart: $CHART_NAME"

# Create a new Helm chart
helm create $CHART_NAME

echo "📁 Chart structure created:"
tree $CHART_NAME

# Customize the chart values
cat << EOF > $CHART_NAME/values.yaml
# Default values for $CHART_NAME
replicaCount: 2

image:
  repository: nginx
  pullPolicy: IfNotPresent
  tag: "1.21"

nameOverride: ""
fullnameOverride: ""

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: true
  className: ""
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
  hosts:
    - host: $CHART_NAME.local
      paths:
        - path: /
          pathType: Prefix
  tls: []

resources:
```

```
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 250m
    memory: 256Mi

autoscaling:
  enabled: true
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80

nodeSelector: {}
tolerations: []
affinity: {}

# Custom application configuration
app:
  environment: production
  debug: false
  database:
    host: db.example.com
    port: 5432
    name: myapp
EOF

# Create a ConfigMap template for application configuration
cat << 'EOF' > $CHART_NAME/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ include "webapp.fullname" . }}-config
  labels:
    {{- include "webapp.labels" . | nindent 4 }}
data:
  app.properties: |
    environment={{ .Values.app.environment }}
    debug={{ .Values.app.debug }}
    database.host={{ .Values.app.database.host }}
    database.port={{ .Values.app.database.port | quote }}
    database.name={{ .Values.app.database.name }}
EOF
```

```
# Update deployment template to use ConfigMap
cat << 'EOF' > $CHART_NAME/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "webapp.fullname" . }}
  labels:
    {{- include "webapp.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "webapp.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml") . | sha256sum }}
      labels:
        {{- include "webapp.selectorLabels" . | nindent 8 }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /
              port: http
            initialDelaySeconds: 5
            periodSeconds: 5
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
```

```
        volumeMounts:
          - name: config
            mountPath: /etc/config
            readOnly: true
        env:
          - name: ENVIRONMENT
            value: {{ .Values.app.environment | quote }}
      volumes:
        - name: config
          configMap:
            name: {{ include "webapp.fullname" . }}-config
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
EOF

echo "✅ Helm chart created successfully!"

# Validate the chart
echo "🔍 Validating chart..."
helm lint $CHART_NAME

# Test chart rendering
echo "🧪 Testing chart templates..."
helm template $CHART_NAME ./$CHART_NAME

# Install the chart
echo "🚀 Installing chart..."
helm install $CHART_NAME-release ./$CHART_NAME

echo "📊 Chart deployed! Check status with:"
echo "  helm status $CHART_NAME-release"
echo "  kubectl get all -l app.kubernetes.io/instance=$CHART_NAME-release"

echo "🧹 Cleanup:"
```

```
echo "  helm uninstall $CHART_NAME-release"
echo "  rm -rf $CHART_NAME"
```

Demo: Multi-Environment Deployment with Helm

Create a system for deploying applications across different environments:

bash

```
#!/bin/bash
# save as multi-env-helm-demo.sh
APP_NAME="myapp"
ENVIRONMENTS=("development" "staging" "production")

echo "🌍 Multi-Environment Helm Deployment Demo"

# Create base chart
helm create $APP_NAME

# Create environment-specific values files
create_env_values() {
    local env=$1
    local replicas cpu_limit memory_limit ingress_host

    case $env in
        "development")
            replicas=1
            cpu_limit="200m"
            memory_limit="256Mi"
            ingress_host="dev-$APP_NAME.local"
            ;;
        "staging")
            replicas=2
            cpu_limit="500m"
            memory_limit="512Mi"
            ingress_host="staging-$APP_NAME.local"
            ;;
        "production")
            replicas=3
            cpu_limit="1000m"
            memory_limit="1Gi"
            ingress_host="$APP_NAME.example.com"
            ;;
    esac

    cat << EOF > $APP_NAME/values-$env.yaml
# Values for $env environment
replicaCount: $replicas

image:
  repository: nginx
  tag: "1.21"
```

```
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: true
  className: "nginx"
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  hosts:
    - host: $ingress_host
      paths:
        - path: /
          pathType: Prefix

resources:
  limits:
    cpu: $cpu_limit
    memory: $memory_limit
  requests:
    cpu: $(echo $cpu_limit | sed 's/[0-9]*m//' | awk '{print $1/2"m"}' | sed 's/\..*m/m/')
    memory: $(echo $memory_limit | sed 's/[0-9]*Mi//' | awk '{print $1/2"Mi"}' | sed 's/\..*Mi/Mi/')

autoscaling:
  enabled: $([ "$env" = "production" ] && echo "true" || echo "false")
  minReplicas: $replicas
  maxReplicas: $(($replicas * 3))
  targetCPUUtilizationPercentage: 80

# Environment-specific configuration
env:
  name: $env
  debug: $([ "$env" = "development" ] && echo "true" || echo "false")
  logLevel: $([ "$env" = "production" ] && echo "warn" || echo "info")
EOF
}

# Create environment-specific values
for env in "${ENVIRONMENTS[@]}"; do
    echo " 📝 Creating values for $env environment..."
    create_env_values $env
done
```

```
# Create namespaces for each environment
for env in "${ENVIRONMENTS[@]}"; do
    echo "🏗️ Creating namespace for $env..."
    kubectl create namespace $env --dry-run=client -o yaml | kubectl apply -f -
done

# Deploy to each environment
deploy_to_env() {
    local env=$1
    echo "🚀 Deploying $APP_NAME to $env environment..."

    helm upgrade --install $APP_NAME-$env ./$APP_NAME \
        --namespace $env \
        --values ./$APP_NAME/values-$env.yaml \
        --wait \
        --timeout 300s

    echo "✅ Deployment to $env complete!"
}

# Deploy to all environments
for env in "${ENVIRONMENTS[@]}"; do
    deploy_to_env $env
done

echo "📊 Deployment Summary:"
for env in "${ENVIRONMENTS[@]}"; do
    echo "--- $env Environment ---"
    helm status $APP_NAME-$env -n $env
    kubectl get pods -n $env -l app.kubernetes.io/instance=$APP_NAME-$env
done

echo "🔍 Check deployments:"
echo "  helm list --all-namespaces"
echo "  kubectl get pods --all-namespaces -l app.kubernetes.io/name=$APP_NAME"

echo "🧹 Cleanup all environments:"
for env in "${ENVIRONMENTS[@]}"; do
    echo "  helm uninstall $APP_NAME-$env -n $env"
done
echo "  kubectl delete namespace ${ENVIRONMENTS[*]}"
echo "  rm -rf $APP_NAME"
```

Mini-Project: Helm Chart Repository and CI/CD Pipeline

Create a complete Helm chart repository with automated testing and deployment:

bash

```
#!/bin/bash
# save as helm-cicd-pipeline.sh
REPO_NAME="myapp-charts"
CHART_NAME="webapp"

echo "🏭 Helm Chart Repository and CI/CD Pipeline Setup"

# Create chart repository structure
mkdir -p $REPO_NAME/{charts,docs,scripts,tests}
cd $REPO_NAME

# Initialize git repository
git init
cat << EOF > .gitignore
*.tgz
.DS_Store
Thumbs.db
EOF

# Create base chart
helm create charts/$CHART_NAME

# Create chart testing configuration
cat << EOF > charts/$CHART_NAME/ci/test-values.yaml
# Test values for CI pipeline
replicaCount: 1

image:
  repository: nginx
  tag: "alpine"
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 80

resources:
  limits:
    cpu: 100m
    memory: 128Mi
  requests:
    cpu: 50m
    memory: 64Mi
```

```
# Minimal configuration for testing
autoscaling:
  enabled: false

ingress:
  enabled: false
EOF

# Create comprehensive test suite
cat << 'EOF' > tests/chart-test.sh
#!/bin/bash
set -e

CHART_PATH="charts/webapp"
RELEASE_NAME="test-release"
NAMESPACE="chart-testing"

echo "🧪 Running Helm Chart Tests"

# Create testing namespace
kubectl create namespace $NAMESPACE --dry-run=client -o yaml | kubectl apply -f -

# Lint the chart
echo "📋 Linting chart..."
helm lint $CHART_PATH

# Test template rendering
echo "🎨 Testing template rendering..."
helm template $RELEASE_NAME $CHART_PATH --values $CHART_PATH/ci/test-values.yaml > /tmp/rendered-templates

# Validate generated YAML
echo "✅ Validating generated YAML..."
kubectl apply --dry-run=client -f /tmp/rendered-templates.yaml

# Install chart for integration testing
echo "🚀 Installing chart for testing..."
helm install $RELEASE_NAME $CHART_PATH \
    --namespace $NAMESPACE \
    --values $CHART_PATH/ci/test-values.yaml \
    --wait \
    --timeout 300s

# Run tests
```

```
echo "🔍 Running integration tests..."

# Test deployment status
if kubectl get deployment $RELEASE_NAME-webapp -n $NAMESPACE >/dev/null 2>&1; then
    echo "✅ Deployment created successfully"
else
    echo "❌ Deployment not found"
    exit 1
fi

# Test service creation
if kubectl get service $RELEASE_NAME-webapp -n $NAMESPACE >/dev/null 2>&1; then
    echo "✅ Service created successfully"
else
    echo "❌ Service not found"
    exit 1
fi

# Test pod readiness
echo "⏳ Waiting for pods to be ready..."
kubectl wait --for=condition=ready pod -l app.kubernetes.io/instance=$RELEASE_NAME -n $NAMESPACE --timeout=1
echo "✅ Pods are ready"

# Test HTTP connectivity
echo "🌐 Testing HTTP connectivity..."
kubectl run test-pod --image=busybox --rm -it --restart=Never -n $NAMESPACE -- \
    wget -q --spider http://$RELEASE_NAME-webapp/
echo "✅ HTTP connectivity test passed"

# Cleanup
echo "🧹 Cleaning up test resources..."
helm uninstall $RELEASE_NAME -n $NAMESPACE
kubectl delete namespace $NAMESPACE

echo "🎉 All tests passed!"
EOF

chmod +x tests/chart-test.sh

# Create release automation script
cat << 'EOF' > scripts/release.sh
#!/bin/bash
set -e
```

```
CHART_PATH="charts/webapp"
VERSION=$1

if [ -z "$VERSION" ]; then
    echo "Usage: $0 <version>"
    echo "Example: $0 1.0.0"
    exit 1
fi

echo "📦 Creating Helm Chart Release v$VERSION"

# Update chart version
sed -i "s/version:.*/version: $VERSION/" $CHART_PATH/Chart.yaml
sed -i "s/appVersion:.*/appVersion: \"$VERSION\"/" $CHART_PATH/Chart.yaml

# Run tests
echo "🧪 Running tests before release..."
./tests/chart-test.sh

# Package chart
echo "📦 Packaging chart..."
helm package $CHART_PATH --destination ./docs/

# Update repository index
echo "📑 Updating repository index..."
helm repo index ./docs/ --url https://mycompany.github.io/myapp-charts/

# Create GitHub Pages index
cat << 'HTML' > docs/index.html
<!DOCTYPE html>
<html>
<head>
    <title>MyApp Helm Charts</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 40px; }
        .chart { background: #f5f5f5; padding: 20px; margin: 20px 0; border-radius: 5px; }
        .version { color: #666; }
    </style>
</head>
<body>
    <h1>🚢 MyApp Helm Chart Repository</h1>
    <p>Add this repository to Helm:</p>
    <pre><code>helm repo add myapp-charts https://mycompany.github.io/myapp-charts/</code></pre>
```

```
    <div class="chart">
        <h2>webapp</h2>
        <p class="version">Latest Version: VERSION_PLACEHOLDER</p>
        <p>A production-ready web application chart with configurable environments.</p>
        <pre><code>helm install my-webapp myapp-charts/webapp</code></pre>
    </div>
</body>
</html>
HTML

# Replace version placeholder
sed -i "s/VERSION_PLACEHOLDER/$VERSION/" docs/index.html

# Git operations
git add .
git commit -m "Release v$VERSION"
git tag -a "v$VERSION" -m "Release version $VERSION"

echo "✅ Release v$VERSION created successfully!"
echo "📤 Push to repository with:"
echo "  git push origin main"
echo "  git push origin v$VERSION"
EOF

chmod +x scripts/release.sh

# Create GitHub Actions workflow
mkdir -p .github/workflows
cat << 'EOF' > .github/workflows/ci.yml
name: Chart CI/CD

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

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Set up Helm
```

```
      uses: azure/setup-helm@v3
      with:
        version: '3.12.0'

    - name: Set up kubectl
      uses: azure/setup-kubectl@v3

    - name: Create k8s Kind Cluster
      uses: helm/kind-action@v1.4.0

    - name: Run chart tests
      run: ./tests/chart-test.sh

  release:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
    - uses: actions/checkout@v3
      with:
        token: ${{ secrets.GITHUB_TOKEN }}

    - name: Set up Helm
      uses: azure/setup-helm@v3
      with:
        version: '3.12.0'

    - name: Configure Git
      run: |
        git config user.name "$GITHUB_ACTOR"
        git config user.email "$GITHUB_ACTOR@users.noreply.github.com"

    - name: Release Chart
      run: |
        VERSION=$(date +%Y.%m.%d-%H%M%S)
        ./scripts/release.sh $VERSION

    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./docs
EOF
```

```
# Create comprehensive documentation
cat << EOF > README.md
# 🚢 MyApp Helm Charts

This repository contains Helm charts for MyApp deployments.

## Usage

Add the repository:
\`\`\`bash
helm repo add myapp-charts https://mycompany.github.io/myapp-charts/
helm repo update
\`\`\`

Install a chart:
\`\`\`bash
helm install my-webapp myapp-charts/webapp
\`\`\`

## Charts

### webapp
A production-ready web application chart with:
- Multi-environment support
- Horizontal Pod Autoscaling
- Ingress configuration
- Resource management
- Health checks

## Development

### Testing
Run the test suite:
\`\`\`bash
./tests/chart-test.sh
\`\`\`

### Releasing
Create a new release:
\`\`\`bash
./scripts/release.sh 1.0.0
\`\`\`

## Contributing
```

```
1. Make changes to charts
2. Run tests locally
3. Create pull request
4. Tests run automatically
5. Merge triggers automatic release
EOF

echo "✅ Helm Chart Repository Setup Complete!"
echo ""
echo "📁 Repository Structure:"
find . -type f -name "*.sh" -o -name "*.yaml" -o -name "*.yml" -o -name "*.md" | head -20

echo ""
echo "🚀 Next Steps:"
echo "1. Initialize git: git add . && git commit -m 'Initial commit'"
echo "2. Run tests: ./tests/chart-test.sh"
echo "3. Create release: ./scripts/release.sh 1.0.0"
echo "4. Push to GitHub and enable Pages"

cd ..
```

20. Configuration and Contexts

Managing multiple Kubernetes clusters and contexts is essential for development workflows and multi-environment deployments.

20.1 Managing Contexts and Configurations

bash

```
# View current configuration
kubectl config view
# Shows merged configuration from all kubeconfig files

# List available contexts
kubectl config get-contexts
# Shows all configured clusters, users, and contexts

# Display current context
kubectl config current-context
# Shows which cluster you're currently connected to

# Switch context
kubectl config use-context minikube
kubectl config use-context production-cluster

# Set default namespace for current context
kubectl config set-context --current --namespace=development
# All kubectl commands will use this namespace by default

# Create a new context
kubectl config set-context dev-context \
  --cluster=development-cluster \
  --user=dev-user \
  --namespace=development

# Rename context
kubectl config rename-context old-name new-name

# Delete context
kubectl config delete-context unused-context
```

Example: Multi-Cluster Management Setup

Create a comprehensive multi-cluster management system:

bash

```bash
#!/bin/bash
# save as multi-cluster-setup.sh
echo "🌐 Multi-Cluster Kubernetes Management Setup"

# Create directory for cluster configurations
mkdir -p ~/.kube/clusters
KUBECONFIG_DIR="$HOME/.kube/clusters"

# Function to create mock cluster configurations
create_cluster_config() {
    local cluster_name=$1
    local environment=$2
    local endpoint=$3

    cat << EOF > $KUBECONFIG_DIR/$cluster_name-config
apiVersion: v1
kind: Config
clusters:
- cluster:
    server: $endpoint
    certificate-authority-data: LS0tLS1CRUdJTi... # Mock cert data
  name: $cluster_name
contexts:
- context:
    cluster: $cluster_name
    user: $cluster_name-user
    namespace: default
  name: $cluster_name
current-context: $cluster_name
users:
- name: $cluster_name-user
  user:
    token: mock-token-for-$cluster_name
EOF

    echo "✅ Created config for $cluster_name ($environment)"
}

# Create configurations for different environments
create_cluster_config "dev-cluster" "development" "https://dev-k8s-api.company.com"
create_cluster_config "staging-cluster" "staging" "https://staging-k8s-api.company.com"
create_cluster_config "prod-cluster" "production" "https://prod-k8s-api.company.com"
```

```
# Create context management script
cat << 'EOF' > ~/.kube/kube-context-manager.sh
#!/bin/bash

# Function to display available contexts
show_contexts() {
    echo "📋 Available Kubernetes Contexts:"
    kubectl config get-contexts
}

# Function to switch context with safety checks
switch_context() {
    local target_context=$1

    if [ -z "$target_context" ]; then
        echo "❌ Please specify a context name"
        show_contexts
        return 1
    fi

    # Check if context exists
    if ! kubectl config get-contexts -o name | grep -q "^$target_context$"; then
        echo "❌ Context '$target_context' not found"
        show_contexts
        return 1
    fi

    # Safety check for production
    if [[ "$target_context" == *"prod"* ]]; then
        echo "⚠️  You're about to switch to PRODUCTION context!"
        read -p "Are you sure? (yes/no): " confirmation
        if [ "$confirmation" != "yes" ]; then
            echo "❌ Context switch cancelled"
            return 1
        fi
    fi

    # Switch context
    kubectl config use-context $target_context

    # Display current context info
    echo "✅ Switched to context: $target_context"
    echo "🔍 Current cluster info:"
    kubectl cluster-info --context=$target_context
```

```
    # Set PS1 to show current context
    export PS1="[k8s:$target_context] \u@\h:\w\$ "
}

# Function to create context alias
create_alias() {
    local alias_name=$1
    local context_name=$2

    if [ -z "$alias_name" ] || [ -z "$context_name" ]; then
        echo "Usage: create_alias <alias> <context>"
        return 1
    fi

    echo "alias k-$alias_name='kubectl config use-context $context_name'" >> ~/.bashrc
    echo "✅ Created alias 'k-$alias_name' for context '$context_name'"
    echo "Run 'source ~/.bashrc' to activate"
}

# Function to backup current kubeconfig
backup_config() {
    local backup_name="kubeconfig-backup-$(date +%Y%m%d-%H%M%S)"
    cp ~/.kube/config ~/.kube/$backup_name
    echo "✅ Kubeconfig backed up to ~/.kube/$backup_name"
}

# Function to merge kubeconfig files
merge_configs() {
    echo "🔀 Merging kubeconfig files from ~/.kube/clusters/"

    # Backup current config
    backup_config

    # Merge all configs
    export KUBECONFIG=~/.kube/config:$(find ~/.kube/clusters -name "*-config" | tr '\n' ':')
    kubectl config view --flatten > ~/.kube/config-merged
    mv ~/.kube/config-merged ~/.kube/config

    echo "✅ Configurations merged successfully"
    show_contexts
}

# Main command handler
```

```
case "$1" in
    "list"|"ls")
        show_contexts
        ;;
    "switch"|"use")
        switch_context $2
        ;;
    "alias")
        create_alias $2 $3
        ;;
    "backup")
        backup_config
        ;;
    "merge")
        merge_configs
        ;;
    "current")
        echo "Current context: $(kubectl config current-context)"
        kubectl config get-contexts $(kubectl config current-context)
        ;;
    *)
        echo "🚀 Kubernetes Context Manager"
        echo "Usage: $0 {list|switch|alias|backup|merge|current}"
        echo ""
        echo "Commands:"
        echo "  list                - Show available contexts"
        echo "  switch <context>    - Switch to specified context"
        echo "  alias <name> <context>  - Create alias for context"
        echo "  backup              - Backup current kubeconfig"
        echo "  merge               - Merge configs from clusters directory"
        echo "  current             - Show current context"
        ;;
esac
EOF

chmod +x ~/.kube/kube-context-manager.sh

# Create convenient aliases
echo "# Kubernetes context management aliases" >> ~/.bashrc
echo "alias kctx='~/.kube/kube-context-manager.sh'" >> ~/.bashrc
echo "alias k-dev='kubectl config use-context dev-cluster'" >> ~/.bashrc
echo "alias k-staging='kubectl config use-context staging-cluster'" >> ~/.bashrc
echo "alias k-prod='kubectl config use-context prod-cluster'" >> ~/.bashrc
```

```
echo "✅ Multi-cluster setup complete!"
echo ""
echo "🔧 Usage:"
echo "  kctx list              # Show all contexts"
echo "  kctx switch dev-cluster # Switch to dev cluster"
echo "  k-dev                  # Quick switch to dev"
echo "  k-staging              # Quick switch to staging"
echo "  k-prod                 # Quick switch to production"
echo ""
echo "📝 Run 'source ~/.bashrc' to activate aliases"
```

Demo: Environment-Specific Configuration Management

Create a system for managing different configurations per environment:

bash

```
#!/bin/bash
# save as env-config-demo.sh
echo "🔧 Environment-Specific Configuration Management Demo"

# Create environment configurations
ENVIRONMENTS=("development" "staging" "production")

create_env_config() {
    local env=$1
    mkdir -p configs/$env

    # Environment-specific kubectl config
    cat << EOF > configs/$env/kubeconfig
apiVersion: v1
kind: Config
clusters:
- cluster:
    server: https://$env-api.example.com
  name: $env-cluster
contexts:
- context:
    cluster: $env-cluster
    user: $env-user
    namespace: $env
  name: $env
current-context: $env
users:
- name: $env-user
  user:
    token: $env-token-12345
EOF

    # Environment-specific resource quotas
    case $env in
        "development")
            cat << EOF > configs/$env/resource-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: $env-quota
  namespace: $env
spec:
  hard:
```

```
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
    pods: "20"
    services: "10"
EOF
            ;;
        "staging")
            cat << EOF > configs/$env/resource-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: $env-quota
  namespace: $env
spec:
  hard:
    requests.cpu: "8"
    requests.memory: 16Gi
    limits.cpu: "16"
    limits.memory: 32Gi
    pods: "50"
    services: "20"
EOF
            ;;
        "production")
            cat << EOF > configs/$env/resource-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: $env-quota
  namespace: $env
spec:
  hard:
    requests.cpu: "20"
    requests.memory: 40Gi
    limits.cpu: "40"
    limits.memory: 80Gi
    pods: "100"
    services: "50"
EOF
            ;;
    esac
```

```
    # Environment-specific network policies
    cat << EOF > configs/$env/network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: $env-network-policy
  namespace: $env
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          environment: $env
  egress:
  - to: []
EOF

    echo "✅ Created configuration for $env environment"
}

# Create configurations for all environments
for env in "${ENVIRONMENTS[@]}"; do
    create_env_config $env
done

# Create environment switcher script
cat << 'EOF' > switch-env.sh
#!/bin/bash
ENV=$1

if [ -z "$ENV" ]; then
    echo "Usage: $0 {development|staging|production}"
    exit 1
fi

if [ ! -d "configs/$ENV" ]; then
    echo "❌ Environment '$ENV' not found"
    exit 1
fi
```

```
echo "🔄 Switching to $ENV environment..."

# Backup current kubeconfig
cp ~/.kube/config ~/.kube/config.backup

# Set environment-specific kubeconfig
export KUBECONFIG=~/.kube/config:configs/$ENV/kubeconfig
kubectl config view --flatten > ~/.kube/config

# Switch to environment context
kubectl config use-context $ENV

# Create namespace if it doesn't exist
kubectl create namespace $ENV --dry-run=client -o yaml | kubectl apply -f -

# Apply environment-specific configurations
echo "📝 Applying environment configurations..."
kubectl apply -f configs/$ENV/resource-quota.yaml
kubectl apply -f configs/$ENV/network-policy.yaml

# Set default namespace
kubectl config set-context --current --namespace=$ENV

echo "✅ Successfully switched to $ENV environment"
echo "📊 Current context: $(kubectl config current-context)"
echo "🏷️  Default namespace: $(kubectl config view --minify | grep namespace | awk '{print $2}')"

# Update shell prompt
export PS1="[$ENV-k8s] \u@\h:\w\$ "
EOF

chmod +x switch-env.sh

echo "📋 Environment configurations created:"
find configs -name "*.yaml" | head -10

echo ""
echo "🚀 Usage:"
echo "  ./switch-env.sh development  # Switch to dev environment"
echo "  ./switch-env.sh staging      # Switch to staging environment"
echo "  ./switch-env.sh production   # Switch to production environment"

echo ""
echo "🔍 Verify configuration:"
```

```
echo "  kubectl config get-contexts"
echo "  kubectl get resourcequota --all-namespaces"
```

Mini-Project: Advanced Kubeconfig Management System

Create a comprehensive system for managing complex multi-cluster, multi-user scenarios:

bash

```
#!/bin/bash
# save as advanced-kubeconfig-manager.sh
echo "🎛️ Advanced Kubeconfig Management System"

# Create directory structure
mkdir -p ~/.kube/{contexts,users,clusters,backups,profiles}

# Create user management functions
create_user_profile() {
    local username=$1
    local email=$2
    local role=$3

    cat << EOF > ~/.kube/users/$username.yaml
apiVersion: v1
kind: Config
preferences: {}
users:
- name: $username
  user:
    client-certificate-data: $(echo "cert-data-for-$username" | base64)
    client-key-data: $(echo "key-data-for-$username" | base64)
metadata:
  email: $email
  role: $role
  created: $(date -Iseconds)
EOF

    echo "👤 Created user profile for $username ($role)"
}

# Create cluster profiles
create_cluster_profile() {
    local cluster_name=$1
    local environment=$2
    local endpoint=$3
    local region=$4

    cat << EOF > ~/.kube/clusters/$cluster_name.yaml
apiVersion: v1
kind: Config
preferences: {}
clusters:
```

```
- cluster:
    certificate-authority-data: $(echo "ca-data-for-$cluster_name" | base64)
    server: $endpoint
  name: $cluster_name
metadata:
  environment: $environment
  region: $region
  created: $(date -Iseconds)
EOF

    echo "🏗️ Created cluster profile for $cluster_name ($environment - $region)"
}

# Create context templates
create_context_template() {
    local context_name=$1
    local cluster=$2
    local user=$3
    local namespace=$4
    local description=$5

    cat << EOF > ~/.kube/contexts/$context_name.yaml
apiVersion: v1
kind: Config
preferences: {}
contexts:
- context:
    cluster: $cluster
    user: $user
    namespace: $namespace
  name: $context_name
current-context: $context_name
metadata:
  description: "$description"
  created: $(date -Iseconds)
EOF

    echo "🔗 Created context template for $context_name"
}

# Advanced kubeconfig manager script
cat << 'EOF' > ~/.kube/advanced-kubeconfig-manager.sh
#!/bin/bash
```

```
KUBE_DIR="$HOME/.kube"
BACKUP_DIR="$KUBE_DIR/backups"
PROFILES_DIR="$KUBE_DIR/profiles"

# Function to create comprehensive backup
create_backup() {
    local backup_name="full-backup-$(date +%Y%m%d-%H%M%S)"
    local backup_path="$BACKUP_DIR/$backup_name"

    mkdir -p "$backup_path"

    # Backup current config
    cp "$KUBE_DIR/config" "$backup_path/config" 2>/dev/null || true

    # Backup all profiles
    cp -r "$KUBE_DIR"/{contexts,users,clusters,profiles} "$backup_path/" 2>/dev/null || true

    # Create backup manifest
    cat << MANIFEST > "$backup_path/manifest.yaml"
backup_name: $backup_name
created: $(date -Iseconds)
kubectl_version: $(kubectl version --client --short 2>/dev/null || echo "unknown")
contexts_count: $(kubectl config get-contexts --no-headers 2>/dev/null | wc -l || echo "0")
current_context: $(kubectl config current-context 2>/dev/null || echo "none")
MANIFEST

    echo "💾 Backup created: $backup_path"
}

# Function to build kubeconfig from profiles
build_config() {
    local profile_name=${1:-"default"}
    local temp_config="/tmp/kubeconfig-build-$"

    echo "🔨 Building kubeconfig profile: $profile_name"

    # Start with empty config
    cat << CONFIG > "$temp_config"
apiVersion: v1
kind: Config
clusters: []
contexts: []
users: []
preferences: {}
```

```
current-context: ""
CONFIG

    # Read profile configuration
    local profile_file="$PROFILES_DIR/$profile_name.yaml"
    if [ ! -f "$profile_file" ]; then
        echo "❌ Profile '$profile_name' not found"
        return 1
    fi

    # Source profile settings
    source "$profile_file"

    # Merge configurations
    export KUBECONFIG="$temp_config"

    # Add clusters
    for cluster_file in $CLUSTERS; do
        if [ -f "$KUBE_DIR/clusters/$cluster_file.yaml" ]; then
            export KUBECONFIG="$KUBECONFIG:$KUBE_DIR/clusters/$cluster_file.yaml"
        fi
    done

    # Add users
    for user_file in $USERS; do
        if [ -f "$KUBE_DIR/users/$user_file.yaml" ]; then
            export KUBECONFIG="$KUBECONFIG:$KUBE_DIR/users/$user_file.yaml"
        fi
    done

    # Add contexts
    for context_file in $CONTEXTS; do
        if [ -f "$KUBE_DIR/contexts/$context_file.yaml" ]; then
            export KUBECONFIG="$KUBECONFIG:$KUBE_DIR/contexts/$context_file.yaml"
        fi
    done

    # Flatten and save
    kubectl config view --flatten > "$KUBE_DIR/config-$profile_name"

    # Set default context if specified
    if [ ! -z "$DEFAULT_CONTEXT" ]; then
        KUBECONFIG="$KUBE_DIR/config-$profile_name" kubectl config use-context "$DEFAULT_CONTEXT"
    fi
```

```
    echo "✅ Profile '$profile_name' built successfully"
    echo "📄 Config saved to: $KUBE_DIR/config-$profile_name"

    rm "$temp_config"
}

# Function to activate profile
activate_profile() {
    local profile_name=$1

    if [ -z "$profile_name" ]; then
        echo "❌ Please specify a profile name"
        list_profiles
        return 1
    fi

    local config_file="$KUBE_DIR/config-$profile_name"

    if [ ! -f "$config_file" ]; then
        echo "🔨 Profile config not found, building..."
        build_config "$profile_name"
    fi

    # Backup current config
    cp "$KUBE_DIR/config" "$KUBE_DIR/config.previous" 2>/dev/null || true

    # Activate profile
    cp "$config_file" "$KUBE_DIR/config"

    echo "✅ Activated profile: $profile_name"
    echo "📊 Available contexts:"
    kubectl config get-contexts

    # Update shell prompt
    export KUBECONFIG_PROFILE="$profile_name"
    export PS1="[k8s:$profile_name] \u@\h:\w\$ "
}

# Function to list profiles
list_profiles() {
    echo "📋 Available Profiles:"
    if [ -d "$PROFILES_DIR" ]; then
        for profile in "$PROFILES_DIR"/*.yaml; do
```

```
            if [ -f "$profile" ]; then
                local name=$(basename "$profile" .yaml)
                local desc=$(grep "DESCRIPTION=" "$profile" 2>/dev/null | cut -d'=' -f2- | tr -d '"')
                echo "  $name - $desc"
            fi
        done
    else
        echo "  No profiles found"
    fi
}

# Function to create new profile
create_profile() {
    local profile_name=$1
    local description=$2

    if [ -z "$profile_name" ]; then
        echo "Usage: create_profile <name> <description>"
        return 1
    fi

    cat << PROFILE > "$PROFILES_DIR/$profile_name.yaml"
# Profile: $profile_name
DESCRIPTION="$description"
CLUSTERS=""
USERS=""
CONTEXTS=""
DEFAULT_CONTEXT=""

# Add cluster/user/context references here
# Example:
# CLUSTERS="dev-cluster prod-cluster"
# USERS="dev-user prod-user"
# CONTEXTS="dev-context prod-context"
# DEFAULT_CONTEXT="dev-context"
PROFILE

    echo "✅ Created profile template: $profile_name"
    echo "📝 Edit: $PROFILES_DIR/$profile_name.yaml"
}

# Function to validate configuration
validate_config() {
    local config_file=${1:-"$KUBE_DIR/config"}
```

```
    echo "🔍 Validating kubeconfig: $config_file"

    # Check if file exists
    if [ ! -f "$config_file" ]; then
        echo "❌ Config file not found: $config_file"
        return 1
    fi

    # Validate YAML syntax
    if ! kubectl config view --kubeconfig="$config_file" >/dev/null 2>&1; then
        echo "❌ Invalid YAML syntax"
        return 1
    fi

    # Check for required fields
    local temp_kubeconfig="$config_file"
    export KUBECONFIG="$temp_kubeconfig"

    echo "📊 Configuration Summary:"
    echo "  Clusters: $(kubectl config get-clusters --no-headers | wc -l)"
    echo "  Users: $(kubectl config get-users --no-headers | wc -l)"
    echo "  Contexts: $(kubectl config get-contexts --no-headers | wc -l)"
    echo "  Current Context: $(kubectl config current-context 2>/dev/null || echo 'none')"

    # Test connectivity to current context
    if kubectl cluster-info --request-timeout=5s >/dev/null 2>&1; then
        echo "✅ Cluster connectivity: OK"
    else
        echo "⚠️  Cluster connectivity: Failed (may be expected for mock configs)"
    fi

    echo "✅ Configuration validation complete"
}

# Function to show detailed context info
show_context_info() {
    local context_name=${1:-$(kubectl config current-context 2>/dev/null)}

    if [ -z "$context_name" ]; then
        echo "❌ No context specified or current context found"
        return 1
    fi
```

```
    echo "🔍 Context Information: $context_name"

    # Get context details
    local cluster=$(kubectl config view -o jsonpath="{.contexts[?(@.name=='$context_name')].context.cluster}")
    local user=$(kubectl config view -o jsonpath="{.contexts[?(@.name=='$context_name')].context.user}")
    local namespace=$(kubectl config view -o jsonpath="{.contexts[?(@.name=='$context_name')].context.namespace}"

    echo "  Cluster: $cluster"
    echo "  User: $user"
    echo "  Namespace: ${namespace:-default}"

    # Get cluster endpoint
    local endpoint=$(kubectl config view -o jsonpath="{.clusters[?(@.name=='$cluster')].cluster.server}")
    echo "  Endpoint: $endpoint"

    # Test permissions
    echo ""
    echo "🔐 Permission Check:"
    kubectl auth can-i get pods --context="$context_name" 2>/dev/null && echo "  ✅ Can get pods" || echo "  ❌ Can
    kubectl auth can-i create deployments --context="$context_name" 2>/dev/null && echo "  ✅ Can create deployme
    kubectl auth can-i '*' '*' --context="$context_name" 2>/dev/null && echo "  ✅ Has admin access" || echo "  ⚠️  Lim
}

# Main command dispatcher
case "$1" in
    "backup")
        create_backup
        ;;
    "build")
        build_config $2
        ;;
    "activate")
        activate_profile $2
        ;;
    "list")
        list_profiles
        ;;
    "create")
        create_profile $2 "$3"
        ;;
    "validate")
        validate_config $2
        ;;
    "info")
```

```
        show_context_info $2
        ;;
    "clean")
        echo "🧹 Cleaning up temporary files..."
        rm -f /tmp/kubeconfig-build-*
        echo "✅ Cleanup complete"
        ;;
    *)
        echo "🎛️ Advanced Kubeconfig Manager"
        echo ""
        echo "Commands:"
        echo "  backup                  - Create full backup"
        echo "  build <profile>         - Build config from profile"
        echo "  activate <profile>      - Activate profile"
        echo "  list                    - List available profiles"
        echo "  create <name> <desc>    - Create new profile"
        echo "  validate [config]       - Validate configuration"
        echo "  info [context]          - Show context information"
        echo "  clean                   - Clean temporary files"
        echo ""
        echo "Examples:"
        echo "  $0 create dev-profile 'Development environment'"
        echo "  $0 build dev-profile"
        echo "  $0 activate dev-profile"
        ;;
esac
EOF

chmod +x ~/.kube/advanced-kubeconfig-manager.sh

# Create sample user profiles
create_user_profile "dev-user" "dev@company.com" "developer"
create_user_profile "ops-user" "ops@company.com" "admin"
create_user_profile "readonly-user" "viewer@company.com" "viewer"

# Create sample cluster profiles
create_cluster_profile "dev-cluster" "development" "https://dev-k8s.company.com" "us-west-2"
create_cluster_profile "staging-cluster" "staging" "https://staging-k8s.company.com" "us-east-1"
create_cluster_profile "prod-cluster" "production" "https://prod-k8s.company.com" "us-east-1"

# Create sample context templates
create_context_template "dev-context" "dev-cluster" "dev-user" "development" "Development environment context"
create_context_template "staging-context" "staging-cluster" "ops-user" "staging" "Staging environment context"
create_context_template "prod-context" "prod-cluster" "ops-user" "production" "Production environment context"
```

```
# Create sample profiles
cat << 'EOF' > ~/.kube/profiles/developer.yaml
# Profile: developer
DESCRIPTION="Developer profile with access to dev and staging"
CLUSTERS="dev-cluster staging-cluster"
USERS="dev-user"
CONTEXTS="dev-context staging-context"
DEFAULT_CONTEXT="dev-context"
EOF

cat << 'EOF' > ~/.kube/profiles/operator.yaml
# Profile: operator
DESCRIPTION="Operations profile with full access"
CLUSTERS="dev-cluster staging-cluster prod-cluster"
USERS="ops-user"
CONTEXTS="dev-context staging-context prod-context"
DEFAULT_CONTEXT="staging-context"
EOF

cat << 'EOF' > ~/.kube/profiles/readonly.yaml
# Profile: readonly
DESCRIPTION="Read-only access to all environments"
CLUSTERS="dev-cluster staging-cluster prod-cluster"
USERS="readonly-user"
CONTEXTS="dev-context staging-context prod-context"
DEFAULT_CONTEXT="dev-context"
EOF

# Create convenient wrapper script
cat << 'EOF' > ~/.kube/kconfig
#!/bin/bash
# Convenient wrapper for advanced kubeconfig manager
~/.kube/advanced-kubeconfig-manager.sh "$@"
EOF

chmod +x ~/.kube/kconfig

# Add to PATH
echo 'export PATH="$HOME/.kube:$PATH"' >> ~/.bashrc

echo "✅ Advanced Kubeconfig Management System Setup Complete!"
echo ""
echo "📁 Directory Structure:"
```

```
find ~/.kube -type f -name "*.yaml" -o -name "*.sh" -o -name "kconfig" | head -15

echo ""
echo "🚀 Usage Examples:"
echo "  kconfig list                  # List available profiles"
echo "  kconfig create myprofile 'desc' # Create new profile"
echo "  kconfig build developer       # Build developer profile"
echo "  kconfig activate developer    # Activate developer profile"
echo "  kconfig info                  # Show current context info"
echo "  kconfig backup                # Create full backup"

echo ""
echo "📝 Next Steps:"
echo "1. Edit profile files in ~/.kube/profiles/"
echo "2. Build and activate a profile"
echo "3. Verify with: kubectl config get-contexts"
```

21. Advanced Tips

This final section covers advanced kubectl techniques, performance optimization, and production best practices.

21.1 Advanced kubectl Techniques

bash

```
# JSON Path queries - extract specific data from resources
kubectl get pods -o jsonpath='{.items[*].metadata.name}'
# Get all pod names

kubectl get pods -o jsonpath='{.items[*].status.podIP}'
# Get all pod IP addresses

kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'
# Get external IP addresses of nodes

# Custom columns - create custom output formats
kubectl get pods -o custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spec.nodeName,IP:.status.pc

kubectl get deployments -o custom-columns=NAME:.metadata.name,REPLICAS:.spec.replicas,READY:.status.readyRepli

# Field selectors - filter resources by field values
kubectl get pods --field-selector=status.phase=Running
kubectl get pods --field-selector=spec.nodeName=worker-1
kubectl get events --field-selector=type=Warning

# Label selectors - complex label-based filtering
kubectl get pods -l 'environment in (production,staging)'
kubectl get pods -l 'app=web,version!=v1'
kubectl get pods -l 'tier,environment notin (test)'

# Sort resources by various fields
kubectl get pods --sort-by=.metadata.creationTimestamp
kubectl get pods --sort-by=.status.startTime
kubectl get nodes --sort-by=.metadata.name

# Watch resources with specific output formats
kubectl get pods -w -o custom-columns=NAME:.metadata.name,STATUS:.status.phase

# Server-side apply - more reliable than client-side apply
kubectl apply --server-side -f deployment.yaml

# Dry-run with diff - see what would change before applying
kubectl diff -f deployment.yaml
kubectl apply --dry-run=server -f deployment.yaml

# Resource usage and limits
kubectl describe nodes | grep -A 5 "Allocated resources"
```

```
kubectl top pods --sort-by=cpu --all-namespaces
kubectl top pods --sort-by=memory --containers
```

Example: Advanced Resource Management and Automation

Create a comprehensive resource management and automation system:

bash

```
#!/bin/bash
# save as advanced-k8s-automation.sh
echo "🚀 Advanced Kubernetes Automation and Management"

# Create comprehensive resource analyzer
cat << 'EOF' > k8s-resource-analyzer.sh
#!/bin/bash

# Advanced resource analysis with multiple metrics
analyze_cluster_resources() {
    echo "📊 Comprehensive Cluster Resource Analysis"
    echo "=========================================="

    # Node resource analysis
    echo "🖥️ Node Resources:"
    kubectl describe nodes | awk '
    /^Name:/ { node=$2 }
    /cpu:/ {
        if ($1 ~ /Allocated/) {
            allocated_cpu=$4
            total_cpu=$6
            cpu_percent=$7
            gsub(/[()%]/, "", cpu_percent)
        }
    }
    /memory:/ {
        if ($1 ~ /Allocated/) {
            allocated_mem=$4
            total_mem=$6
            mem_percent=$7
            gsub(/[()%]/, "", mem_percent)
            printf "  %s: CPU %s/%s (%s%%), Memory %s/%s (%s%%)\n",
                node, allocated_cpu, total_cpu, cpu_percent,
                allocated_mem, total_mem, mem_percent
        }
    }'

    echo ""
    echo "📦 Pod Resource Distribution:"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.spec.containers[]?.resources.requests) |
    "\(.metadata.namespace)/\(.metadata.name): CPU=\(.spec.containers[0].resources.requests.cpu // "none"), Memory=\
```

```
    ' | head -10

    echo ""
    echo "⚠️ Pods Without Resource Limits:"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.spec.containers[0].resources.limits == null) |
    "\(.metadata.namespace)/\(.metadata.name)"
    ' | head -5

    echo ""
    echo "🔥 High Resource Usage Pods:"
    if kubectl top pods --all-namespaces >/dev/null 2>&1; then
        kubectl top pods --all-namespaces --sort-by=cpu | head -5
        echo ""
        kubectl top pods --all-namespaces --sort-by=memory | head -5
    else
        echo "  Metrics server not available"
    fi
}

# Network analysis
analyze_network() {
    echo ""
    echo "🌐 Network Analysis:"
    echo "==================="

    echo "📡 Services and Endpoints:"
    kubectl get services --all-namespaces -o wide | head -10

    echo ""
    echo "🔗 Ingress Resources:"
    kubectl get ingress --all-namespaces 2>/dev/null | head -5 || echo "  No ingress resources found"

    echo ""
    echo "🔒 Network Policies:"
    kubectl get networkpolicies --all-namespaces 2>/dev/null | head -5 || echo "  No network policies found"
}

# Storage analysis
analyze_storage() {
    echo ""
    echo "💾 Storage Analysis:"
    echo "==================="
```

```
    echo "📀 Persistent Volumes:"
    kubectl get pv -o custom-columns=NAME:.metadata.name,CAPACITY:.spec.capacity.storage,STATUS:.status.phase,CL

    echo ""
    echo "📋 Persistent Volume Claims:"
    kubectl get pvc --all-namespaces -o custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,ST

    echo ""
    echo "🏪 Storage Classes:"
    kubectl get storageclass -o custom-columns=NAME:.metadata.name,PROVISIONER:.provisioner,RECLAIM:.reclaimPo
}

# Security analysis
analyze_security() {
    echo ""
    echo "🔐 Security Analysis:"
    echo "====================="

    echo "👥 Service Accounts:"
    kubectl get serviceaccounts --all-namespaces | head -10

    echo ""
    echo "🛡️ RBAC Roles:"
    kubectl get roles,clusterroles --all-namespaces 2>/dev/null | head -10

    echo ""
    echo "🔑 Secrets:"
    kubectl get secrets --all-namespaces -o custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name
}

# Performance analysis
analyze_performance() {
    echo ""
    echo "⚡ Performance Analysis:"
    echo "========================"

    echo "🎯 Pod Restart Analysis:"
    kubectl get pods --all-namespaces -o json | jq -r '
    .items[] |
    select(.status.containerStatuses[]?.restartCount > 0) |
    "\(.metadata.namespace)/\(.metadata.name): \(.status.containerStatuses[0].restartCount) restarts"
    ' | head -5
```

```
    echo ""
    echo "⏱️ Long-running Pods:"
    kubectl get pods --all-namespaces --sort-by=.metadata.creationTimestamp | head -5

    echo ""
    echo "🚨 Failed Pods:"
    kubectl get pods --all-namespaces --field-selector=status.phase=Failed 2>/dev/null | head -5
}

# Main execution
case "$1" in
    "resources")
        analyze_cluster_resources
        ;;
    "network")
        analyze_network
        ;;
    "storage")
        analyze_storage
        ;;
    "security")
        analyze_security
        ;;
    "performance")
        analyze_performance
        ;;
    "all"|"")
        analyze_cluster_resources
        analyze_network
        analyze_storage
        analyze_security
        analyze_performance
        ;;
    *)
        echo "Usage: $0 {resources|network|storage|security|performance|all}"
        ;;
esac
EOF

chmod +x k8s-resource-analyzer.sh

# Create cluster health monitor
cat << 'EOF' > k8s-health-monitor.sh
#!/bin/bash
```

```
ALERT_THRESHOLD_CPU=80
ALERT_THRESHOLD_MEMORY=80
ALERT_THRESHOLD_DISK=85
LOG_FILE="/tmp/k8s-health-monitor.log"

log_message() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

check_node_health() {
    log_message " Starting Node Health Check"

    while read -r node status; do
        if [[ "$status" != "Ready" ]]; then
            log_message " ALERT: Node $node is not ready (Status: $status)"
        else
            log_message " Node $node is healthy"
        fi
    done < <(kubectl get nodes --no-headers -o custom-columns=NAME:.metadata.name,STATUS:.status.conditions[-1]
}

check_pod_health() {
    log_message " Starting Pod Health Check"

    # Check for pods in problematic states
    local problematic_pods=$(kubectl get pods --all-namespaces --no-headers | grep -E "(Error|CrashLoopBackOff|Imag

    if [ "$problematic_pods" -gt 0 ]; then
        log_message " ALERT: $problematic_pods pods in problematic state"
        kubectl get pods --all-namespaces | grep -E "(Error|CrashLoopBackOff|ImagePullBackOff|Pending)" | head -5 | whil
            log_message "  $line"
        done
    else
        log_message " All pods are healthy"
    fi
}

check_resource_usage() {
    log_message " Starting Resource Usage Check"

    if kubectl top nodes >/dev/null 2>&1; then
        # Check node resource usage
        kubectl top nodes --no-headers | while read node cpu_usage cpu_percent memory_usage memory_percent; do
```

```
            cpu_num=$(echo $cpu_percent | tr -d '%')
            mem_num=$(echo $memory_percent | tr -d '%')

            if [ "$cpu_num" -gt "$ALERT_THRESHOLD_CPU" ]; then
                log_message " 🚨 ALERT: High CPU usage on $node: $cpu_percent"
            fi

            if [ "$mem_num" -gt "$ALERT_THRESHOLD_MEMORY" ]; then
                log_message " 🚨 ALERT: High memory usage on $node: $memory_percent"
            fi
        done
    else
        log_message " ⚠️ Metrics server not available - cannot check resource usage"
    fi
}

check_storage_health() {
    log_message " 💾 Starting Storage Health Check"

    # Check PVC status
    local pending_pvcs=$(kubectl get pvc --all-namespaces --no-headers | grep -c "Pending" || echo "0")
    if [ "$pending_pvcs" -gt 0 ]; then
        log_message " ⚠️ ALERT: $pending_pvcs PVCs in Pending state"
    fi

    # Check PV availability
    local available_pvs=$(kubectl get pv --no-headers | grep -c "Available" || echo "0")
    log_message " ℹ️ Available PVs: $available_pvs"
}

check_service_health() {
    log_message " 🌐 Starting Service Health Check"

    # Check services with no endpoints
    kubectl get endpoints --all-namespaces -o json | jq -r '
    .items[] |
    select(.subsets == null or .subsets == []) |
    "\(.metadata.namespace)/\(.metadata.name)"
    ' | while read service; do
        if [ ! -z "$service" ]; then
            log_message " ⚠️ ALERT: Service $service has no endpoints"
        fi
    done
}
```

```
generate_health_report() {
    local report_file="/tmp/k8s-health-report-$(date +%Y%m%d-%H%M%S).html"

    cat << 'HTML' > "$report_file"
<!DOCTYPE html>
<html>
<head>
    <title>Kubernetes Health Report</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .healthy { color: green; }
        .warning { color: orange; }
        .error { color: red; }
        .section { margin: 20px 0; padding: 15px; border: 1px solid #ddd; }
        pre { background: #f5f5f5; padding: 10px; overflow-x: auto; }
    </style>
</head>
<body>
    <h1>💾 Kubernetes Cluster Health Report</h1>
    <p>Generated: $(date)</p>

    <div class="section">
        <h2>📊 Cluster Overview</h2>
        <pre>$(kubectl get nodes -o wide)</pre>
    </div>

    <div class="section">
        <h2>🔍 Pod Status Summary</h2>
        <pre>$(kubectl get pods --all-namespaces | head -20)</pre>
    </div>

    <div class="section">
        <h2>⚠️ Recent Alerts</h2>
        <pre>$(tail -20 "$LOG_FILE" 2>/dev/null || echo "No alerts logged")</pre>
    </div>

    <div class="section">
        <h2>📈 Resource Usage</h2>
        <pre>$(kubectl top nodes 2>/dev/null || echo "Metrics not available")</pre>
    </div>
</body>
</html>
HTML
```

```
    log_message " 📋 Health report generated: $report_file"
    echo "Health report: $report_file"
}

# Main monitoring loop
monitor_cluster() {
    log_message " 🚀 Starting Kubernetes Health Monitor"

    while true; do
        check_node_health
        check_pod_health
        check_resource_usage
        check_storage_health
        check_service_health

        log_message " 😴 Sleeping for 60 seconds..."
        sleep 60
    done
}

# Command dispatcher
case "$1" in
    "nodes")
        check_node_health
        ;;
    "pods")
        check_pod_health
        ;;
    "resources")
        check_resource_usage
        ;;
    "storage")
        check_storage_health
        ;;
    "services")
        check_service_health
        ;;
    "report")
        generate_health_report
        ;;
    "monitor")
        monitor_cluster
        ;;
```

```
    *)
        echo "🏥 Kubernetes Health Monitor"
        echo "Usage: $0 {nodes|pods|resources|storage|services|report|monitor}"
        echo ""
        echo "Commands:"
        echo "  nodes     - Check node health"
        echo "  pods      - Check pod health"
        echo "  resources - Check resource usage"
        echo "  storage   - Check storage health"
        echo "  services  - Check service health"
        echo "  report    - Generate HTML health report"
        echo "  monitor   - Start continuous monitoring"
        ;;
esac
EOF

chmod +x k8s-health-monitor.sh

echo "✅ Advanced resource analyzer created!"
echo "✅ Health monitor created!"

# Run the analyzers
echo ""
echo "🔍 Running cluster analysis..."
./k8s-resource-analyzer.sh resources

echo ""
echo "🏥 Running health check..."
./k8s-health-monitor.sh pods
```

Demo: Production-Ready Deployment Pipeline

Create a complete production deployment pipeline with all best practices:

bash

```
#!/bin/bash
# save as production-deployment-pipeline.sh
echo "🏭 Production-Ready Kubernetes Deployment Pipeline"

# Create production-grade deployment template
cat << 'EOF' > production-app-template.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: ${APP_NAME}-${ENVIRONMENT}
  labels:
    environment: ${ENVIRONMENT}
    managed-by: deployment-pipeline
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ${APP_NAME}-sa
  namespace: ${APP_NAME}-${ENVIRONMENT}
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: ${APP_NAME}-config
  namespace: ${APP_NAME}-${ENVIRONMENT}
data:
  app.properties: |
    environment=${ENVIRONMENT}
    log.level=${LOG_LEVEL}
    database.host=${DB_HOST}
    database.port=${DB_PORT}
    cache.enabled=${CACHE_ENABLED}
---
apiVersion: v1
kind: Secret
metadata:
  name: ${APP_NAME}-secrets
  namespace: ${APP_NAME}-${ENVIRONMENT}
type: Opaque
data:
  database-password: ${DB_PASSWORD_B64}
  api-key: ${API_KEY_B64}
---
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${APP_NAME}
  namespace: ${APP_NAME}-${ENVIRONMENT}
  labels:
    app: ${APP_NAME}
    version: ${APP_VERSION}
    environment: ${ENVIRONMENT}
spec:
  replicas: ${REPLICAS}
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  selector:
    matchLabels:
      app: ${APP_NAME}
  template:
    metadata:
      labels:
        app: ${APP_NAME}
        version: ${APP_VERSION}
        environment: ${ENVIRONMENT}
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8080"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: ${APP_NAME}-sa
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 2000
      containers:
      - name: ${APP_NAME}
        image: ${IMAGE_REGISTRY}/${APP_NAME}:${APP_VERSION}
        imagePullPolicy: Always
        ports:
        - containerPort:
```