# Kubernetes Workshop

## Part 2: Advanced Topics

From Development to Production

# Part 2 Overview

**Duration:** 4 hours (with breaks)

**Focus:** Advanced Kubernetes concepts for production deployments

**Topics:**

- Ingress Controllers
- Helm Package Management
- GitOps with Flux
- Monitoring and Observability
- Advanced Deployment Strategies
- Autoscaling
- Security and RBAC
- Multi-Cluster Management

# Prerequisites

**From Part 1:**

- Kubernetes fundamentals (Pods, Deployments, Services)
- kubectl proficiency
- Container and Docker Compose knowledge
- Basic YAML understanding

**New Requirements:**

- Helm CLI installed
- Git and GitHub account
- Understanding of CI/CD concepts

# Workshop Structure

| Section | Topic | Duration |
|---|---|---|
| 01 | Ingress Controllers | 45 min |
| 02 | Helm | 50 min |
| 03 | GitOps with Flux | 50 min |
| 04 | Monitoring | 45 min |
| 05 | Advanced Deployments | 40 min |
| 06 | Autoscaling | 45 min |
| 07 | Security and RBAC | 50 min |
| 08 | Multi-Cluster | 45 min |

# Section 1: Ingress Controllers

**Why Ingress?**

- Expose multiple services through single external IP
- HTTP/HTTPS routing based on hostnames and paths
- TLS termination
- Load balancing

**Services Recap:**

- ClusterIP - Internal only
- NodePort - External access on each node
- LoadBalancer - Cloud provider load balancer

**Ingress provides** intelligent HTTP routing

# Ingress Architecture

```
Internet
   |
[External LoadBalancer]
   |
[Ingress Controller] (nginx, traefik, etc.)
   |
[Ingress Rules]
   |
+---+---+---+
|   |   |   |
v   v   v   v
Svc1 Svc2 Svc3 (ClusterIP Services)
```

**Ingress Controller** implements the rules

**Ingress Resource** defines the rules

# Path-Based Routing

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
spec:
  rules:
  - http:
      paths:
      - path: /app1
        pathType: Prefix
        backend:
          service:
            name: app1-service
            port:
              number: 80
      - path: /app2
        pathType: Prefix
        backend:
          service:
            name: app2-service
            port:
              number: 80
```

# Host-Based Routing

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: multi-host-ingress
spec:
  rules:
  - host: web.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: web-service
            port:
              number: 80
  - host: api.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 8080
```

# TLS Termination

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: <base64-encoded-cert>
  tls.key: <base64-encoded-key>
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-ingress
spec:
  tls:
  - hosts:
    - secure.example.com
    secretName: tls-secret
  rules:
  - host: secure.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: secure-app
            port:
              number: 80
```

# Ingress Annotations

### Common annotations (NGINX):

```
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/rate-limit: "100"
    nginx.ingress.kubernetes.io/cors-allow-origin: "*"
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: basic-auth
```

**Annotations provide** advanced features beyond basic routing

# Section 2: Helm

### What is Helm?

- Package manager for Kubernetes
- Manages collections of YAML files as "charts"
- Templating engine for reusable configurations
- Version control for deployments

### Why Helm?

- Reduces YAML duplication
- Environment-specific configurations
- Rollback capabilities
- Chart repositories for sharing

# Helm Architecture

### Helm 3 Components:

- Helm CLI - Client tool
- Charts - Package format
- Releases - Installed chart instances
- Repositories - Chart distribution

**No Tiller** in Helm 3 (removed for security)

### Chart Structure:

```
mychart/
  Chart.yaml          # Chart metadata
  values.yaml         # Default configuration
  templates/          # Kubernetes manifests with templates
    deployment.yaml
    service.yaml
```

# Helm Templating

### values.yaml:

```yaml
replicaCount: 3
image:
  repository: nginx
  tag: "1.25.3"
service:
  type: ClusterIP
  port: 80
```

### templates/deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: {{ .Values.service.port }}
```

# Helm Commands

```
# Repository management
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
helm search repo nginx

# Install chart
helm install my-release bitnami/nginx
helm install my-release ./mychart --values prod-values.yaml

# Manage releases
helm list
helm status my-release
helm upgrade my-release bitnami/nginx --set replicaCount=5
helm rollback my-release 1

# Chart development
helm create mychart
helm lint mychart
helm package mychart
```

# Helm Dependencies

**Chart.yaml:**

```
apiVersion: v2
name: webapp
dependencies:
- name: redis
  version: "18.0.0"
  repository: https://charts.bitnami.com/bitnami
  condition: redis.enabled
- name: postgresql
  version: "12.0.0"
  repository: https://charts.bitnami.com/bitnami
  condition: postgresql.enabled
```

**Update dependencies:**

```
helm dependency update ./webapp
```

**Benefits:** Manage complex application stacks as single unit

# Section 3: GitOps with Flux

**GitOps Principles:**

1. **Declarative** - System state described declaratively
2. **Versioned** - State stored in Git
3. **Pulled** - Software agents pull desired state
4. **Reconciled** - Agents ensure actual = desired

**Benefits:**

- Git as single source of truth
- Audit trail in Git history
- Easy rollbacks via Git revert
- Automated deployments

# Flux Architecture

```
Git Repository
   |
   | (watches)
   v
[Source Controller] - Polls Git for changes
   |
   v
[Kustomize Controller] - Applies Kubernetes manifests
[Helm Controller] - Manages Helm releases
[Notification Controller] - Sends alerts
```

**Flux continuously** reconciles cluster state with Git

# GitRepository Source

```yaml
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 1m0s
  ref:
    branch: main
  url: https://github.com/your-org/your-repo
```

## Kustomization:

```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: apps
  namespace: flux-system
spec:
  interval: 5m
  path: ./apps/production
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
```

# HelmRelease with Flux

```yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: bitnami
  namespace: flux-system
spec:
  interval: 30m
  url: https://charts.bitnami.com/bitnami
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: redis
  namespace: default
spec:
  interval: 5m
  chart:
    spec:
      chart: redis
      version: '18.x.x'
      sourceRef:
        kind: HelmRepository
        name: bitnami
  values:
    auth:
      enabled: false
```

# Image Automation

### Automatically update images:

```yaml
apiVersion: image.toolkit.fluxcd.io/v1beta2
kind: ImageRepository
metadata:
  name: myapp
spec:
  image: ghcr.io/myorg/myapp
  interval: 1m
---
apiVersion: image.toolkit.fluxcd.io/v1beta2
kind: ImagePolicy
metadata:
  name: myapp
spec:
  imageRepositoryRef:
    name: myapp
  policy:
    semver:
      range: 1.x.x
```

**Flux commits** updated image tags to Git automatically

# Section 4: Monitoring

**Three Pillars of Observability:**

1. **Metrics** - Time-series data (CPU, memory, requests)
2. **Logs** - Event records (application logs, errors)
3. **Traces** - Request flow across services

**Stack Overview:**

- Prometheus - Metrics collection and storage
- Grafana - Visualization and dashboards
- Loki - Log aggregation
- Jaeger - Distributed tracing

# Prometheus Architecture

```
[Application] --expose--> /metrics endpoint
      |
      | (scrape)
      v
[Prometheus Server]
      |
      | (query)
      v
[Grafana] / [AlertManager]
```

**Metric Types:**

- Counter - Increases only (requests, errors)
- Gauge - Current value (memory, connections)
- Histogram - Value distribution (latency buckets)
- Summary - Similar to histogram with quantiles

# ServiceMonitor

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: app-metrics
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: myapp
  endpoints:
  - port: metrics
    path: /metrics
    interval: 30s
```

**ServiceMonitor** tells Prometheus what to scrape

**Prometheus Operator** manages Prometheus instances declaratively

# PromQL Basics

```
# Request rate
rate(http_requests_total[5m])

# Error rate
rate(http_requests_total{status="500"}[5m])

# 95th percentile latency
histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))

# Memory usage
container_memory_usage_bytes{pod="myapp-xxx"}

# Aggregation
sum(rate(http_requests_total[5m])) by (endpoint)
```

# PrometheusRule for Alerts

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: app-alerts
  namespace: monitoring
spec:
  groups:
  - name: app
    interval: 30s
    rules:
    - alert: HighErrorRate
      expr: |
        rate(http_requests_total{status="500"}[5m]) > 0.05
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "High error rate on {{ $labels.pod }}"
        description: "Error rate is {{ $value }} req/s"
    - alert: PodCrashLooping
      expr: |
        rate(kube_pod_container_status_restarts_total[15m]) > 0
      for: 5m
      labels:
        severity: warning
```

# Loki for Logs

## Architecture:

```
[App Pods]
    |
    | (tail logs)
    v
[Promtail DaemonSet]
    |
    | (push)
    v
[Loki]
    |
    | (query)
    v
[Grafana]
```

## LogQL Query:

```
{namespace="production", app="api"} |= "error"
| json
| level="error"
| line_format "{{.timestamp}} {{.message}}"
```

# Section 5: Advanced Deployments

**Deployment Strategies:**

- **Recreate** - Stop all, deploy new (downtime)
- **Rolling Update** - Gradual replacement (default)
- **Blue/Green** - Two complete environments, switch
- **Canary** - Gradual traffic shift to new version
- **A/B Testing** - Route by user attributes

**Production deployments require** careful rollout strategies

# Rolling Update Configuration

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2          # Add 2 extra pods during update
      maxUnavailable: 1    # Max 1 pod down at a time
  minReadySeconds: 30      # Wait 30s after pod ready
  progressDeadlineSeconds: 600  # Fail after 10 min
```

**maxSurge** - How fast to update

**maxUnavailable** - How safe to update

# Blue/Green Deployment

```yaml
# Blue deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-blue
spec:
  template:
    metadata:
      labels:
        version: blue
---
# Green deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-green
spec:
  template:
    metadata:
      labels:
        version: green
```

### Switch traffic:

```
kubectl patch service myapp -p '{"spec":{"selector":{"version":"green"}}}'
```

# Canary Deployment

**Manual approach:**

```
# Stable: 9 replicas
# Canary: 1 replica (10% traffic)
```

**Service selects both** using common label

**Gradually increase** canary replicas, decrease stable

**Automated with Flagger:**

- Monitors metrics during rollout
- Automatically promotes or rolls back
- Integrates with Istio, Linkerd, NGINX

# Flagger Canary

```yaml
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: myapp
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  service:
    port: 80
  analysis:
    interval: 1m
    threshold: 5
    maxWeight: 50
    stepWeight: 10
    metrics:
    - name: request-success-rate
      thresholdRange:
        min: 99
      interval: 1m
    - name: request-duration
      thresholdRange:
        max: 500
      interval: 1m
```

# Section 6: Autoscaling

**Three Types:**

1. **Horizontal Pod Autoscaler (HPA)**
   - Scales number of pod replicas
   - Based on CPU, memory, custom metrics
2. **Vertical Pod Autoscaler (VPA)**
   - Adjusts resource requests/limits
   - Right-sizes containers
3. **Cluster Autoscaler**
   - Adds/removes nodes
   - Based on pending pods

# Horizontal Pod Autoscaler

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

# HPA Scaling Behavior

```yaml
spec:
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50      # Scale down by max 50%
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 100     # Scale up by max 100%
        periodSeconds: 30
      - type: Pods
        value: 2       # Or add max 2 pods
        periodSeconds: 30
      selectPolicy: Max
```

**Scale up fast, scale down slow**

# KEDA Event-Driven Autoscaling

**KEDA extends HPA** with 50+ scalers:

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: rabbitmq-scaler
spec:
  scaleTargetRef:
    name: message-processor
  minReplicaCount: 0    # Scale to zero!
  maxReplicaCount: 30
  triggers:
  - type: rabbitmq
    metadata:
      queueName: tasks
      queueLength: "10"
  - type: cron
    metadata:
      timezone: America/New_York
      start: 0 8 * * 1-5
      end: 0 18 * * 1-5
      desiredReplicas: "10"
```

# Section 7: Security and RBAC

**Security Layers:**

1. **Authentication** - Who are you? (certificates, tokens)
2. **Authorization** - What can you do? (RBAC)
3. **Admission Control** - Should this be allowed? (policies)
4. **Network Policies** - Who can talk to whom?
5. **Pod Security** - How secure are pods?

**Defense in depth** - Multiple security layers

# RBAC Core Concepts

**Four Resource Types:**

- **Role** - Permissions in a namespace
- **ClusterRole** - Permissions cluster-wide
- **RoleBinding** - Assigns Role to subjects
- **ClusterRoleBinding** - Assigns ClusterRole cluster-wide

**Subjects:**

- User - Human user
- Group - Collection of users
- ServiceAccount - Pod identity

# Role and RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: myapp-sa
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

# ServiceAccounts

**Every pod** uses a ServiceAccount for API access:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: myapp-sa
automountServiceAccountToken: false  # Disable if not needed
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  template:
    spec:
      serviceAccountName: myapp-sa
      containers:
      - name: app
        image: myapp:latest
```

**Default behavior:** Token mounted at `/var/run/secrets/kubernetes.io/serviceaccount/token`

# NetworkPolicy

```yaml
# Default deny all
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
---
# Allow frontend to backend
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tier: frontend
    ports:
    - protocol: TCP
      port: 8080
```

# Pod Security Standards

**Three levels:**

1. **Privileged** - Unrestricted (not recommended)
2. **Baseline** - Minimally restrictive (prevents known escalations)
3. **Restricted** - Heavily restricted (best practices)

**Enforce on namespace:**

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

# Secure Pod Configuration

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: myapp:latest
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
    volumeMounts:
    - name: tmp
      mountPath: /tmp
  volumes:
  - name: tmp
    emptyDir: {}
```

# Section 8: Multi-Cluster Management

**Why Multiple Clusters?**

- High availability across regions
- Disaster recovery
- Environment separation (dev/staging/prod)
- Geographic data requirements
- Load distribution
- Compliance and isolation

**Trade-off:** Increased complexity vs. improved resilience

# Multi-Cluster Patterns

### 1. Federation

- Single control plane
- Unified policy management
- Tools: kubefed

### 2. Independent Clusters

- Separate clusters with service mesh
- Cross-cluster communication
- Tools: Istio, Cilium Cluster Mesh

### 3. Hub and Spoke

- Central hub manages spoke clusters
- Centralized monitoring and deployments
- Tools: Rancher, Argo CD ApplicationSets

# Context Management

```
# List contexts
kubectl config get-contexts

# Switch context
kubectl config use-context production-cluster

# Create aliases
alias k-prod='kubectl --context=production-cluster'
alias k-dev='kubectl --context=dev-cluster'

# Use both
k-prod get pods
k-dev get pods

# Or use kubectx
kubectx production-cluster
kubectx dev-cluster
```

# Cross-Cluster Communication

### ExternalName Service:

```
apiVersion: v1
kind: Service
metadata:
  name: api-cluster1
spec:
  type: ExternalName
  externalName: api.cluster1.example.com
```

### Cilium Cluster Mesh:

- Connect multiple clusters
- Transparent cross-cluster communication
- Service discovery across clusters

# GitOps for Multi-Cluster

**Repository structure:**

```
fleet-infra/
├── apps/
│   ├── base/
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   ├── dev/
│   │   ├── kustomization.yaml
│   │   └── values.yaml
│   ├── staging/
│   │   └── kustomization.yaml
│   └── production/
│       └── kustomization.yaml
└── clusters/
    ├── dev-cluster/
    ├── staging-cluster/
    └── production-cluster/
```

**Each cluster** runs Flux, syncs from same repo, different paths

# Disaster Recovery with Velero

```
# Install Velero
velero install \
  --provider aws \
  --bucket k8s-backups \
  --backup-location-config region=us-east-1

# Create backup
velero backup create production-backup \
  --include-namespaces production

# Schedule regular backups
velero schedule create daily-backup \
  --schedule="0 2 * * *" \
  --include-namespaces production

# Restore to another cluster
velero restore create \
  --from-backup production-backup
```

# Multi-Cluster Monitoring

### Prometheus Federation:

```
scrape_configs:
- job_name: 'federate-cluster-1'
  honor_labels: true
  metrics_path: '/federate'
  params:
    'match[]':
      - '{job="kubernetes-pods"}'
  static_configs:
  - targets:
    - 'prometheus.cluster-1:9090'
    labels:
      cluster: 'cluster-1'
```

**Central Prometheus** scrapes from cluster Prometheus instances

**Grafana** shows unified view across all clusters

# Best Practices Summary

**Ingress:**

- Use TLS for all external services
- Implement rate limiting
- Monitor ingress controller metrics

**Helm:**

- Use named templates for reusability
- Separate values per environment
- Version and test charts before deployment

**GitOps:**

- Protect main branch
- Require pull request reviews
- Use automated testing in CI

# Best Practices Summary (2)

**Monitoring:**

- Alert on symptoms, not causes
- Use SLOs for alerting
- Keep dashboard count reasonable
- Label metrics consistently

**Deployments:**

- Always use health checks
- Set resource requests and limits
- Test rollback procedures
- Monitor during rollouts

# Best Practices Summary (3)

**Autoscaling:**

- Scale up fast, down slow
- Set conservative limits
- Use PodDisruptionBudgets
- Test under load

**Security:**

- Least privilege principle
- Enable Pod Security Standards
- Use NetworkPolicies
- Rotate secrets regularly
- Scan images for vulnerabilities

# Best Practices Summary (4)

**Multi-Cluster:**

- Consistent configuration across clusters
- Automate with GitOps
- Regular backup testing
- Document failover procedures
- Monitor costs per cluster

# Production Readiness Checklist

**Infrastructure:**

- [ ] Resource limits on all containers
- [ ] Health checks configured
- [ ] PodDisruptionBudgets for critical apps
- [ ] HPA configured for scalable workloads

**Security:**

- [ ] RBAC configured
- [ ] NetworkPolicies in place
- [ ] Pod Security Standards enforced
- [ ] Secrets encrypted at rest
- [ ] TLS for all external services

# Production Readiness Checklist (2)

**Monitoring:**

- [ ] Prometheus monitoring all services
- [ ] Key metrics dashboards in Grafana
- [ ] AlertManager configured
- [ ] Logs aggregated in Loki
- [ ] On-call procedures documented

**Operations:**

- [ ] GitOps implemented
- [ ] Automated backups configured
- [ ] Disaster recovery tested
- [ ] Runbooks for common issues
- [ ] Incident response plan

# Kubernetes Ecosystem Tools

**Package Management:**

- Helm - Chart manager
- Kustomize - Configuration management

**GitOps:**

- Flux CD - Continuous deployment
- Argo CD - Declarative GitOps

**Service Mesh:**

- Istio - Traffic management, security
- Linkerd - Lightweight service mesh

**Monitoring:**

- Prometheus - Metrics
- Grafana - Visualization
- Loki - Logs

# Kubernetes Ecosystem Tools (2)

**Security:**

- OPA/Gatekeeper - Policy enforcement
- Falco - Runtime security
- Trivy - Vulnerability scanning
- Sealed Secrets - Encrypted secrets

**CI/CD:**

- Tekton - Kubernetes-native pipelines
- Jenkins X - GitOps CI/CD
- Argo Workflows - Workflow engine

**Cluster Management:**

- Rancher - Multi-cluster management
- Lens - Kubernetes IDE

# Common Pitfalls to Avoid

**Resource Management:**

- Not setting resource limits → OOMKilled pods
- Over-provisioning → Wasted resources
- No HPA → Manual scaling burden

**Security:**

- Using default ServiceAccount with full permissions
- No NetworkPolicies → Unrestricted pod communication
- Running as root → Security risk

**Operations:**

- Manual deployments → Configuration drift
- No monitoring → Blind to issues
- No backups → Data loss risk

# Debugging Advanced Issues

### Ingress not routing:

```
kubectl describe ingress myapp
kubectl logs -n ingress-nginx deployment/ingress-nginx-controller
kubectl get events --sort-by='.lastTimestamp'
```

### Helm release failing:

```
helm status myrelease
helm get manifest myrelease
helm get values myrelease
kubectl describe pod -l app.kubernetes.io/instance=myrelease
```

### Flux not reconciling:

```
flux get all
flux logs
kubectl describe kustomization apps -n flux-system
```

# Debugging Advanced Issues (2)

### HPA not scaling:

```
kubectl get hpa
kubectl describe hpa myapp
kubectl top pods  # Verify metrics-server working
kubectl get --raw /apis/metrics.k8s.io/v1beta1/pods
```

### RBAC permission denied:

```
kubectl auth can-i <verb> <resource> --as=<user>
kubectl auth can-i --list --as=system:serviceaccount:default:myapp
kubectl describe role,rolebinding -n default
```

### NetworkPolicy blocking traffic:

```
kubectl describe networkpolicy
kubectl run test --rm -it --image=busybox -- wget -O- http://service
```

# Performance Optimization

**Pod Performance:**

- Right-size resource requests/limits
- Use VPA for auto-tuning
- Enable CPU throttling awareness
- Optimize readiness probes

**Cluster Performance:**

- Node autoscaling for demand
- Use node affinity for workload placement
- Implement pod priorities
- Use local storage for high I/O workloads

**Application Performance:**

- Enable HTTP/2 in Ingress
- Use persistent connections

# Cost Optimization

**Strategies:**

1. **Right-sizing** - Use VPA recommendations
2. **Spot instances** - For fault-tolerant workloads
3. **Cluster autoscaling** - Scale down unused nodes
4. **Resource quotas** - Prevent over-provisioning
5. **KEDA scale-to-zero** - For intermittent workloads
6. **Multi-tenancy** - Share clusters across teams

**Monitor costs:**

- Kubecost - Kubernetes cost monitoring
- Cloud provider cost analysis
- Alert on unexpected increases

# Future of Kubernetes

**Trending Topics:**

**WebAssembly (Wasm):**

- Lightweight container alternative
- Faster startup times
- Better resource utilization

**Platform Engineering:**

- Internal developer platforms
- Self-service infrastructure
- Backstage, Crossplane

**AI/ML Workloads:**

- GPU scheduling
- Model serving (KServe)
- Training job management (Kubeflow)

# Continued Learning

**Official Resources:**

- kubernetes.io/docs
- Cloud Native Computing Foundation (CNCF)
- Kubernetes Slack and forums

**Certifications:**

- CKA (Certified Kubernetes Administrator)
- CKAD (Certified Kubernetes Application Developer)
- CKS (Certified Kubernetes Security Specialist)

**Practice:**

- Set up home lab with kind/k3s
- Contribute to open source projects
- Practice on killercoda.com

# Lab Time!

**Access labs at:**

- Section materials in workshop repository
- Each section has hands-on exercises
- Solutions provided for reference

**Today's Focus:**

- Complete at least 5 sections
- Bonus challenges for extra practice
- Ask questions anytime!

**Remember:** Best way to learn is by doing

# Q&A and Wrap-up

**What we covered:**

- Advanced networking with Ingress
- Package management with Helm
- GitOps workflows with Flux
- Production monitoring and observability
- Advanced deployment strategies
- Autoscaling patterns
- Security and RBAC
- Multi-cluster management

**You're now ready** to deploy production-grade Kubernetes applications!

# Thank You!

**Resources:**

- Workshop repository: github.com/your-repo/compose-to-kubernetes
- Kubernetes documentation: kubernetes.io
- CNCF Landscape: landscape.cncf.io

**Stay Connected:**

- Join Kubernetes Slack
- Follow Cloud Native community
- Practice regularly

**Questions?** Let's discuss!

# Extra Resources

**Books:**

- "Kubernetes: Up and Running" - Kelsey Hightower
- "The Kubernetes Book" - Nigel Poulton
- "Production Kubernetes" - Josh Rosso & Rich Lander

**Podcasts:**

- Kubernetes Podcast from Google
- The Cloudcast
- Software Engineering Daily

**YouTube Channels:**

- CNCF
- DevOps Toolkit
- TechWorld with Nana

**Practice Platforms:**