MODULE 9

KUBERNETES (K8s)

ZERO TO HERO

Master Container Orchestration from Basics to Advanced

Deploy
Automated Deployment

Scale
Auto-scaling Magic

Heal
Self-healing Systems

Learning Journey

01

Introduction & Core Concepts

- What is Kubernetes?
- Problems K8s Solves
- Container Orchestration Features
- Core Components Overview

02

Architecture & Components

- Master Node vs Worker Node
- K8s Architecture Deep Dive
- Local Setup with Minikube
- YAML Configuration

03

Practical Deployment

- Kubectl Essential Commands
- Demo Project: Fenrir App
- Namespaces & Ingress
- Data Persistence & Services

04

Advanced Ecosystem

- Helm Package Manager
- ArgoCD for GitOps
- Istio Service Mesh
- Real DevOps Workflow

FOUNDATION

What is Kubernetes?

Definition

Kubernetes (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Docker **runs** containers → K8s **manages** containers

Key Facts

- ✓ Works with Docker, containerd
- ✓ Used by Google, Netflix, Amazon
- ✓ Manages clusters of machines

Problems Kubernetes Solves

✗ Manual deployment	✓ Automatic scaling
✗ No auto-scaling	✓ Self-healing
✗ Difficult networking	✓ Load balancing
✗ Downtime in updates	✓ Zero downtime

Container Orchestration Features

- Auto-scheduling
- Auto-scaling
- Self-healing
- Load balancing
- Rolling updates
- Config management
- Service discovery
- Network management

ARCHITECTURE

Core Components & Architecture

Pod

Smallest deployable unit containing one or more containers.

Service

Exposes pods and provides stable networking.

Ingress

Manages external HTTP/HTTPS access.

ConfigMap

Stores configuration data.

Secret

Stores sensitive data like passwords.

Volume

Provides persistent storage.

Deployment

Manages stateless apps with auto-scaling.

StatefulSet

Manages stateful apps like databases.

Node

A machine (VM/physical) running pods.

Master Node (Control Plane)

API Server	Entry point
Scheduler	Assigns pods
Controller Manager	Maintains state
ETCD	Cluster database

Worker Node

Kubelet	Communicates
Kube Proxy	Networking
Container Runtime	Runs containers
Pods	Your apps

GETTING STARTED

Local Setup & YAML Configuration

Minikube

Runs a local K8s cluster on your laptop for development.

```
minikube start
```

Kubectl

The CLI tool to interact with your cluster.

```
kubectl get nodes
```

YAML Configuration Structure

metadata
Name, labels

spec
Desired state

status
Auto-generated

Example Pod YAML Configuration

```
apiVersion: v1
kind: Pod
metadata:
name: mypod
labels:
app: nginx
spec:
containers:
- name: nginx
image: nginx
ports:
- containerPort: 80
```

CLI MASTERY

Essential Kubectl Commands

Get Resources

```
kubectl get pods
```

List all pods

```
kubectl get nodes
```

List all nodes

```
kubectl get services
```

List all services

Create Resources

```
kubectl apply -f app.yaml
```

Create from YAML

```
kubectl create namespace dev
```

Create namespace

Describe Resources

```
kubectl describe pod mypod
```

Detailed pod info

```
kubectl explain pod
```

Resource schema

Logs & Debug

```
kubectl logs mypod
```

Pod logs

```
kubectl exec -it mypod -- bash
```

Access pod shell

```
kubectl top nodes
```

Resource usage

Delete Resources

```
kubectl delete pod mypod
```

Delete pod

```
kubectl delete -f app.yaml
```

Delete from YAML

Pro Tips

Aliases: k=kubectl

Shortcuts: po=pods, svc=services, deploy=deployments

HANDS-ON LAB

Demo Project: Deploy Fenrir App

1 **Build** Create Docker image

2 **Push** Upload to registry

3 **Deploy** Create Deployment

4 **Expose** Create Service

5 **Access** Via NodePort

1. Build & Push Docker Image

```
docker build -t fenrir-app .
```

```
docker tag fenrir-app username/fenrir-app
```

```
docker push username/fenrir-app
```

2. Create Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata: name: fenrir-deployment
spec: replicas: 3
selector: matchLabels: app: fenrir
template:
metadata: labels: app: fenrir
spec:
containers:
- name: fenrir
image: username/fenrir-app
ports:
- containerPort: 3000
```

3. Create Service

```
apiVersion: v1
kind: Service
metadata: name: fenrir-service
spec:
selector: app: fenrir
type: NodePort
ports:
```

4. Deploy & Verify

```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

```
kubectl get pods
```

ADVANCED CONCEPTS

Namespaces & Ingress

What is a Namespace?

Namespaces provide logical separation within a single cluster, like folders in a filesystem.

- **default** – Your resources
- **kube-system** – System resources
- **kube-public** – Public info

Create Namespace

```
kubectl create namespace dev
```

- ✓ Environment isolation
- ✓ Resource limits
- ✓ Security boundaries

Ingress vs Service

Service (L4)

TCP/UDP, internal/external

Ingress (L7)

HTTP/HTTPS, external

Ingress Controller Required

Ingress resources need an Ingress Controller (like nginx or traefik) to process the routing rules.

Routing Example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata: name: app-ingress
spec:
rules:
- host: example.com
http:
paths:
- path: /api
backend: service: name: backend (port: 80)
- path: /ui
backend: service: name: frontend (port: 80)
```

STORAGE & NETWORKING

Data Persistence & Services

Persistent Volume
Actual storage

PVC
Storage request

Storage Class
Dynamic provisioning

PVC Example

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata: name: myclaim
spec:
accessModes: ["ReadWriteOnce"]
resources: requests: storage: 5Gi
```

Using in Pod

```
spec: containers:
- name: app
image: myapp
volumeMounts:
- name: storage
mountPath: /data
volumes:
- name: storage
persistentVolumeClaim:
claimName: myclaim
```

Service Types

ClusterIP
Internal only

NodePort
External (Dev)

LoadBalancer
Cloud LB

Headless
Direct Pod IP

ADVANCED ECOSYSTEM

Helm Package Manager & ArgoCD

Helm

The package manager for Kubernetes.

Helm Chart

A bundle of K8s resources.

- Reusability
- Easy upgrades
- Configurable values

ArgoCD

GitOps continuous delivery tool.

Core Principle

Git is the source of truth.

- Auto-sync cluster
- Continuous deployment
- Drift detection

Helm Chart Structure

```
chart/
├── templates/
│   ├── deployment.yaml
│   ├── service.yaml
├── values.yaml
├── Chart.yaml
```

Workflow

1. Push to Git
2. ArgoCD detects change
3. Auto-sync to cluster

PRODUCTION WORKFLOW

Istio & Real DevOps K8s Workflow

Istio Service Mesh

A service mesh for traffic management, security, and observability.

Traffic Mgmt
Routing, load balancing

Security
mTLS, policies

Observability
Metrics, tracing

Key Benefits

Canary Deployments: Gradual rollout

Circuit Breakers: Fail fast & recover

Zero-Trust Network: mTLS encryption

Real DevOps K8s Workflow

Code Push
Git

Build Image
CI/CD

Push Registry
Docker Hub

Detect Change
ArgoCD

Deploy App
K8s

Expose Service
Ingress

Monitor
Prometheus

Verify
Tests

CAPSTONE & CAREER

Mini Project & Interview Prep

Mini Project: Flask + MongoDB

Deploy a complete stack: Flask (frontend) + MongoDB (database).

- Deployment (Flask)
- StatefulSet (Mongo)
- Service
- ConfigMap
- PVC
- Ingress

Top Interview Questions

Q1: What is a Pod?

Smallest deployable unit. Can contain one or more containers.

Q2: Deployment vs StatefulSet?

Deployment: Stateless apps. **StatefulSet:** Stateful apps like DBs.

Q3: What is ETCD?

Distributed key-value store for cluster state.

Project Commands

```
# Deploy all
kubectl apply -f .
# Check status
kubectl get all
# Access via Ingress
curl http://myapp.local
```

More Key Questions

Q4: What is Ingress?

Manages external HTTP/HTTPS access.

Q5: What is Helm?

K8s package manager for resource bundles.

Your K8s Journey Continues

You now have the foundation to orchestrate containers like a pro.

Remember: Practice is key to mastery.

Keep Experimenting

Build, break, and rebuild. True learning comes from doing.

Join the Community

Connect, share, and grow with fellow K8s enthusiasts.

Go Build Amazing

The cloud-native world is yours to architect.

🚀 Happy Orchestrating! 🚀