Day 14: Kubernetes & Container Orchestration

This module builds **core Kubernetes fundamentals with production context**. The focus is not just YAML, but *why Kubernetes exists*, *how its control plane works*, and *how applications are safely deployed, scaled, and updated*.

---

1. Background: Why Kubernetes Exists

The Problem Kubernetes Solves

Before Kubernetes: - Manual container deployment - No self-healing - No standardized scaling - Fragile deployments

Kubernetes provides: - **Scheduling** of containers - **Self-healing** (restart, reschedule) - **Service discovery & load balancing** - **Declarative desired state**

> Kubernetes is an **orchestrator**, not a container runtime.

---

2. Kubernetes Architecture (Core Concepts)

Control Plane Components

Component	Responsibility
API Server	Entry point for all operations
etcd	Distributed key-value store (cluster state)
Scheduler	Decides where Pods run
Controller Manager	Maintains desired state

Worker Node Components

Component	Responsibility
kubelet	Ensures containers are running
kube-proxy	Networking & service routing
Container Runtime	Runs containers (containerd)

---

3. Local Kubernetes Cluster Setup

Option 1: Minikube

```
minikube start
kubectl get nodes
```

Option 2: Kind

```
kind create cluster
kubectl cluster-info
```

✔️ Checklist: Kubernetes cluster operational

---

4. Kubernetes Core Objects

Pod

- Smallest deployable unit
- One or more containers

Deployment

- Manages ReplicaSets
- Enables rolling updates & rollbacks

Service

- Stable network endpoint
- Load-balancing across Pods

Ingress

- HTTP/HTTPS routing
- External access

---

5. Deployment Manifest (Application)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-app
```

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
      - name: demo
        image: demo/app:1.0
        ports:
        - containerPort: 8000
```

Apply:

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

✔️ Checklist: Deployment applied successfully

---

6. Service Manifest (Expose Application)

```
apiVersion: v1
kind: Service
metadata:
  name: demo-service
spec:
  selector:
    app: demo
  ports:
  - port: 80
    targetPort: 8000
  type: ClusterIP
```

Verify:

```
kubectl get svc
```

✔️ Checklist: Service exposing application

---

7. Ingress (External Access)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-ingress
spec:
  rules:
  - host: demo.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: demo-service
            port:
              number: 80
```

Enable ingress controller (Minikube):

```
minikube addons enable ingress
```

---

8. ConfigMaps & Secrets (Environment Management)

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_ENV: production
  LOG_LEVEL: info
```

Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
type: Opaque
data:
  DB_PASSWORD: cGFzc3dvcmQ=
```

Mount as environment variables:

```
envFrom:
- configMapRef:
    name: app-config
- secretRef:
    name: app-secrets
```

✔️ Checklist: ConfigMaps & Secrets configured

---

9. Horizontal Pod Autoscaling (HPA)

Why HPA

- Handle traffic spikes automatically
- Optimize cost

Metrics Server

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/
latest/download/components.yaml
```

HPA Manifest

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: demo-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```
    name: demo-app
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
```

Verify:

```
kubectl get hpa
```

✔️ Checklist: HPA scaling based on metrics

---

10. Rolling Updates (Zero Downtime Deployments)

How Rolling Updates Work

- Gradually replace old Pods
- Maintain service availability

Update Image

```
kubectl set image deployment/demo-app demo=demo/app:2.0
```

Monitor rollout:

```
kubectl rollout status deployment/demo-app
```

Rollback if needed:

```
kubectl rollout undo deployment/demo-app
```

✔️ Checklist: Rolling update performed successfully

Day 14 Completion Status

- ☑️ Local Kubernetes cluster running
- ☑️ Deployments applied
- ☑️ Services & Ingress configured
- ☑️ ConfigMaps & Secrets used
- ☑️ HPA scaling verified
- ☑️ Rolling updates completed

Key Interview Insight

> Kubernetes is about **desired state management**. You describe *what you want* — Kubernetes figures out *how to keep it running*.

Strong Next Steps

- Kubernetes networking deep dive
- Helm charts
- Kubernetes security (RBAC, NetworkPolicies)
- GitOps with ArgoCD

Tell me what you want to continue with.