KCNA - Container Orchestration

Container Orchestration (22%)

Download PDF Version

This domain covers container fundamentals and how Kubernetes orchestrates containerized workloads.

Container Fundamentals

What is a Container?

A container is a lightweight, standalone, executable package that includes everything needed to run a piece of software:

- Code
- Runtime
- System tools
- System libraries
- Settings

Container vs Virtual Machine

Aspect	Container	Virtual Machine
Size	Megabytes	Gigabytes
Startup	Seconds	Minutes
Isolation	Process-level	Hardware-level
OS	Shares host kernel	Full OS per VM
Resource Usage	Lightweight	Heavy

Container Runtime

The container runtime is responsible for running containers. Kubernetes supports several runtimes through the Container Runtime Interface (CRI):

- **containerd** - Industry-standard container runtime
- **CRI-O** - Lightweight container runtime for Kubernetes
- **Docker Engine** - Popular container platform (via cri-dockerd)

Container Images

Image Layers

Container images are built in layers:

```
 ┌─────────────────────────────┐
│   Application Code          │   <- Your code
├─────────────────────────────┤
│   Dependencies              │   <- npm, pip packages
├─────────────────────────────┤
│   Runtime                   │   <- Node.js, Python
├─────────────────────────────┤
│   Base OS                   │   <- Alpine, Ubuntu
└─────────────────────────────┘
```

Image Registries

Container images are stored in registries:

- **Docker Hub** - Public registry
- **Google Container Registry (GCR)**
- **Amazon Elastic Container Registry (ECR)**
- **Azure Container Registry (ACR)**
- **Harbor** - Open-source private registry

Image Naming Convention

```
 registry/repository:tag

Examples:
docker.io/library/nginx:1.21
gcr.io/my-project/my-app:v1.0.0
my-registry.com/team/service:latest
```

Kubernetes Scheduling

How Scheduling Works

1. User creates a Pod
2. API Server stores Pod in etcd (status: Pending)
3. Scheduler watches for unscheduled Pods
4. Scheduler selects a suitable node
5. Scheduler updates Pod with node assignment

6. Kubelet on the node creates the container

Scheduling Factors

The scheduler considers:

- **Resource requests and limits**
- **Node selectors and affinity**
- **Taints and tolerations**
- **Pod topology spread constraints**
- **Available resources on nodes**

Node Selector

Simple way to constrain Pods to nodes with specific labels:

```
 apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeSelector:
    disktype: ssd
  containers:
  - name: nginx
    image: nginx
```

Node Affinity

More expressive way to specify node constraints:

```
 apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-west-1a
            - us-west-1b
  containers:
  - name: nginx
    image: nginx
```

Taints and Tolerations

Taints allow nodes to repel certain Pods:

```
 # Add taint to node
kubectl taint nodes node1 key=value:NoSchedule
```

Tolerations allow Pods to schedule on tainted nodes:

```
 apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "value"
    effect: "NoSchedule"
  containers:
  - name: nginx
    image: nginx
```

Resource Management

Resource Requests and Limits

```
 apiVersion: v1
kind: Pod
metadata:
  name: resource-demo
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- **Requests**: Minimum resources guaranteed
- **Limits**: Maximum resources allowed

Quality of Service (QoS) Classes

QoS Class	Condition
Guaranteed	Requests = Limits for all containers
Burstable	At least one container has requests < limits
BestEffort	No requests or limits specified

LimitRange

Sets default resource constraints for a namespace:

```
 apiVersion: v1
kind: LimitRange
metadata:
  name: default-limits
spec:
  limits:
  - default:
      cpu: "500m"
      memory: "256Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    type: Container
```

ResourceQuota

Limits total resource consumption in a namespace:

```
 apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
spec:
  hard:
    requests.cpu: "4"
    requests.memory: "8Gi"
    limits.cpu: "8"
    limits.memory: "16Gi"
    pods: "10"
```

Scaling

Horizontal Pod Autoscaler (HPA)

Automatically scales the number of Pods based on metrics:

```
 apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

Vertical Pod Autoscaler (VPA)

Automatically adjusts resource requests and limits.

Cluster Autoscaler

Automatically adjusts the size of the Kubernetes cluster.

Key Concepts to Remember

1. **Containers share the host kernel** - Unlike VMs
2. **Images are immutable** - Changes create new layers
3. **Scheduler uses filtering and scoring** - To find the best node
4. **Resource requests affect scheduling** - Limits affect runtime
5. **QoS determines eviction priority** - BestEffort evicted first

Practice Questions

1. What is the difference between a container and a virtual machine?
2. What is the role of the kube-scheduler?
3. How do taints and tolerations work together?
4. What is the difference between resource requests and limits?
5. What are the three QoS classes in Kubernetes?

← Previous: Kubernetes Fundamentals | Back to KCNA Overview | Next: Cloud Native Architecture →