1 CKAD

Generated on: 2026-01-13 15:04:29 **Version:** 1.0

1.1 Table of Contents

1.2 Overview

CNCF CKAD

The **Certified Kubernetes Application Developer (CKAD)** exam certifies that candidates can design, build, and deploy cloud native applications for Kubernetes.

1.3 Exam Overview

Detail	Information
Exam Format	Performance-based (hands-on)
Number of Questions	15-20
Duration	2 hours
Passing Score	66%
Certification Validity	3 years
Cost	$395 USD

Detail	Information
Retake Policy	1 free retake
Kubernetes Version	1.30

1.4 Exam Domains & Weights

Domain	Weight
Application Design and Build	20%
Application Deployment	20%
Application Observability and Maintenance	15%
Application Environment, Configuration and Security	25%
Services and Networking	20%

1.5 Prerequisites

- Basic understanding of Kubernetes concepts
- Familiarity with YAML syntax
- Linux command line proficiency
- Container fundamentals (Docker)

1.6 Study Resources

1.6.1 Official Resources

- CKAD Exam Curriculum
- Kubernetes Documentation
- Kubernetes Tasks

1.6.2 Recommended Courses

- Kubernetes for Developers (LFD259)
- CKAD with Tests - Udemy

1.6.3 Practice Resources

- Killercoda CKAD Scenarios ⭐ **Highly Recommended**
- Kubernetes Playground
- killer.sh CKAD Simulator

1.7 Quick Navigation

- 01 - Application Design and Build
- 02 - Application Deployment
- 03 - Application Observability and Maintenance
- 04 - Application Environment, Configuration and Security
- 05 - Services and Networking
- Sample Practice Questions

1.8 Exam Environment

The CKAD exam provides:

- Access to multiple Kubernetes clusters
- `kubectl` with auto-completion enabled
- Access to Kubernetes documentation (kubernetes.io)
- A Linux terminal environment

1.8.1 Allowed Resources During Exam

- kubernetes.io/docs
- kubernetes.io/blog
- helm.sh/docs

1.9 Exam Tips

1. **Practice kubectl imperative commands** - They save significant time
2. **Master YAML generation** - Use `kubectl run --dry-run=client -o yaml`
3. **Know vim/nano basics** - You'll edit YAML files frequently
4. **Use aliases** - Set up `alias k=kubectl` and enable auto-completion
5. **Bookmark important docs** - Prepare bookmarks for quick access
6. **Time management** - Don't spend too long on any single question
7. **Practice on Killercoda** - Free hands-on scenarios at killercoda.com/ckad

1.10 Useful kubectl Commands

```bash
# Set alias
alias k=kubectl

# Enable auto-completion
source <(kubectl completion bash)
complete -o default -F __start_kubectl k

# Generate YAML templates
k run nginx --image=nginx --dry-run=client -o yaml > pod.yaml
k create deployment nginx --image=nginx --dry-run=client -o yaml
        > deploy.yaml
```

```
k create service clusterip nginx --tcp=80:80 --dry-run=client -o
        yaml > svc.yaml

# Quick pod creation
k run nginx --image=nginx --port=80 --labels=app=web

# Expose deployment
k expose deployment nginx --port=80 --target-port=80 --
        type=ClusterIP

# Scale deployment
k scale deployment nginx --replicas=3

# Set resources
k set resources deployment nginx --limits=cpu=200m,memory=512Mi

# Rollout commands
k rollout status deployment/nginx
k rollout history deployment/nginx
k rollout undo deployment/nginx
```

1.11 Registration

Register for CKAD Exam

1.12 Application Design and Build

Download PDF Version

This domain covers designing and building cloud native applications for Kubernetes.

1.13 Container Images

1.13.1 Building Container Images

```dockerfile
# Example Dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .
```

```
EXPOSE 8080

USER 1000

CMD ["python", "app.py"]
```

1.13.2 Multi-stage Builds

```
# Build stage
FROM golang:1.21 AS builder
WORKDIR /app
COPY . .
RUN CGO_ENABLED=0 go build -o myapp

# Runtime stage
FROM alpine:3.18
COPY --from=builder /app/myapp /myapp
ENTRYPOINT ["/myapp"]
```

1.13.3 Image Best Practices

- Use specific image tags, not `latest`
- Use minimal base images (alpine, distroless)
- Run as non-root user
- Use multi-stage builds to reduce image size
- Scan images for vulnerabilities

1.14 Jobs and CronJobs

1.14.1 Job

A Job creates one or more Pods and ensures they complete successfully.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job
spec:
  completions: 3
  parallelism: 2
  backoffLimit: 4
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34
        command: ["perl", "-Mbignum=bpi", "-wle", "print
        bpi(2000)"]
      restartPolicy: Never
```

Key Fields:

Field	Description
`completions`	Number of successful completions required
`parallelism`	Number of pods running in parallel
`backoffLimit`	Number of retries before marking as failed
`activeDeadlineSeconds`	Maximum time for the job
`restartPolicy`	Must be `Never` or `OnFailure`

1.14.2 CronJob

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: backup-cronjob
spec:
  schedule: "0 2 * * *"  # Daily at 2 AM
  concurrencyPolicy: Forbid
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: backup-tool:v1
            command: ["/bin/sh", "-c", "backup.sh"]
          restartPolicy: OnFailure
```

Cron Schedule Format:

```
┌───────────── minute (0 - 59)
│ ┌───────────── hour (0 - 23)
│ │ ┌───────────── day of month (1 - 31)
│ │ │ ┌───────────── month (1 - 12)
│ │ │ │ ┌───────────── day of week (0 - 6)
│ │ │ │ │
* * * * *
```

Concurrency Policies:

Policy	Description
`Allow`	Allow concurrent jobs (default)
`Forbid`	Skip new job if previous is still running

Policy	Description
`Replace`	Replace currently running job with new one

1.15 Multi-Container Pods

1.15.1 Sidecar Pattern

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-sidecar
spec:
  containers:
  - name: app
    image: myapp:v1
    volumeMounts:
    - name: logs
      mountPath: /var/log/app
  - name: log-shipper
    image: fluentd:v1
    volumeMounts:
    - name: logs
      mountPath: /var/log/app
  volumes:
  - name: logs
    emptyDir: {}
```

1.15.2 Init Containers

Init containers run before app containers start:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-init
spec:
  initContainers:
  - name: init-db
    image: busybox:1.36
    command: ['sh', '-c', 'until nc -z db-service 5432; do sleep
        2; done']
  - name: init-config
    image: busybox:1.36
    command: ['sh', '-c', 'wget -O /config/app.conf http://config-
        server/app.conf']
    volumeMounts:
    - name: config
      mountPath: /config
  containers:
  - name: app
```

```
    image: myapp:v1
    volumeMounts:
    - name: config
      mountPath: /config
  volumes:
  - name: config
    emptyDir: {}
```

1.15.3 Ambassador Pattern

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-ambassador
spec:
  containers:
  - name: app
    image: myapp:v1
    env:
    - name: DB_HOST
      value: "localhost"
    - name: DB_PORT
      value: "5432"
  - name: ambassador
    image: ambassador-proxy:v1
    ports:
    - containerPort: 5432
```

1.15.4 Adapter Pattern

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-adapter
spec:
  containers:
  - name: app
    image: legacy-app:v1
    volumeMounts:
    - name: logs
      mountPath: /var/log
  - name: adapter
    image: log-adapter:v1
    volumeMounts:
    - name: logs
      mountPath: /var/log
```

1.16 Volumes

1.16.1 emptyDir

Temporary storage that exists for the Pod's lifetime:

```
volumes:
- name: cache
  emptyDir: {}
- name: memory-cache
  emptyDir:
    medium: Memory
    sizeLimit: 100Mi
```

1.16.2 hostPath

Mount a file or directory from the host node:

```
volumes:
- name: host-data
  hostPath:
    path: /data
    type: DirectoryOrCreate
```

1.16.3 PersistentVolumeClaim

```
volumes:
- name: data
  persistentVolumeClaim:
    claimName: my-pvc
```

1.17 Kubectl Imperative Commands

1.17.1 Create Resources Quickly

```
# Create a pod
kubectl run nginx --image=nginx --port=80

# Create a deployment
kubectl create deployment nginx --image=nginx --replicas=3

# Create a job
kubectl create job pi --image=perl -- perl -Mbignum=bpi -wle
        'print bpi(2000)'

# Create a cronjob
kubectl create cronjob backup --image=backup --schedule="0 2 * *
        *" -- /bin/sh -c 'backup.sh'
```

```
# Generate YAML
kubectl run nginx --image=nginx --dry-run=client -o yaml >
        pod.yaml
```

1.18 Key Concepts to Remember

1. **Jobs** - Run to completion, use `restartPolicy: Never` or `OnFailure`
2. **CronJobs** - Scheduled jobs using cron syntax
3. **Init containers** - Run before main containers, must complete successfully
4. **Sidecar pattern** - Helper container alongside main app
5. **Multi-stage builds** - Reduce image size

1.19 Practice Questions

1. How do you create a Job that runs 5 completions with 2 parallel pods?
2. What is the difference between init containers and sidecar containers?
3. How do you generate YAML for a pod without creating it?
4. What cron schedule runs every Monday at 3 AM?
5. What happens if a CronJob's concurrencyPolicy is set to Forbid?

Back to CKAD Overview | Next: Application Deployment →

1.20 Application Deployment

Download PDF Version

This domain covers deploying applications in Kubernetes using various strategies and tools.

1.21 Deployments

1.21.1 Creating Deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
```

1.21.2 Imperative Commands

```
# Create deployment
kubectl create deployment nginx --image=nginx:1.21 --replicas=3

# Scale deployment
kubectl scale deployment nginx --replicas=5

# Update image
kubectl set image deployment/nginx nginx=nginx:1.22

# Edit deployment
kubectl edit deployment nginx
```

1.22 Deployment Strategies

1.22.1 Rolling Update (Default)

```
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
```

Parameter	**Description**
`maxSurge`	Max pods above desired count during update
`maxUnavailable`	Max pods unavailable during update

1.22.2 Recreate

```yaml
spec:
  strategy:
    type: Recreate
```

All existing pods are killed before new ones are created.

1.23 Rollouts

1.23.1 Rollout Commands

```bash
# Check rollout status
kubectl rollout status deployment/nginx

# View rollout history
kubectl rollout history deployment/nginx

# View specific revision
kubectl rollout history deployment/nginx --revision=2

# Undo rollout (rollback)
kubectl rollout undo deployment/nginx

# Rollback to specific revision
kubectl rollout undo deployment/nginx --to-revision=2

# Pause rollout
kubectl rollout pause deployment/nginx

# Resume rollout
kubectl rollout resume deployment/nginx

# Restart deployment
kubectl rollout restart deployment/nginx
```

1.23.2 Recording Changes

```bash
# Record the command in revision history
kubectl set image deployment/nginx nginx=nginx:1.22 --record
```

1.24 Blue-Green Deployment

Manual blue-green deployment using Services:

```yaml
# Blue deployment (current)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      version: blue
  template:
    metadata:
      labels:
        app: myapp
        version: blue
    spec:
      containers:
      - name: app
        image: myapp:v1
---
# Green deployment (new)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      version: green
  template:
    metadata:
      labels:
        app: myapp
        version: green
    spec:
      containers:
      - name: app
        image: myapp:v2
---
# Service - switch selector to change versions
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
    version: blue  # Change to 'green' to switch
  ports:
  - port: 80
    targetPort: 8080
```

1.25 Canary Deployment

```yaml
# Stable deployment (90% traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-stable
spec:
  replicas: 9
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: app
        image: myapp:v1
---
# Canary deployment (10% traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-canary
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: app
        image: myapp:v2
---
# Service routes to both (based on replica ratio)
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp  # Matches both deployments
  ports:
  - port: 80
    targetPort: 8080
```

1.26 Helm

1.26.1 Helm Basics

```bash
# Add repository
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update

# Search charts
helm search repo nginx
helm search hub wordpress

# Install chart
helm install my-nginx bitnami/nginx

# Install with custom values
helm install my-nginx bitnami/nginx -f values.yaml
helm install my-nginx bitnami/nginx --set service.type=NodePort

# List releases
helm list
helm list -A  # All namespaces

# Upgrade release
helm upgrade my-nginx bitnami/nginx --set replicaCount=3

# Rollback
helm rollback my-nginx 1

# Uninstall
helm uninstall my-nginx

# Show chart info
helm show values bitnami/nginx
helm show chart bitnami/nginx
```

1.26.2 Helm Chart Structure

```
mychart/
├── Chart.yaml          # Chart metadata
├── values.yaml         # Default configuration values
├── charts/             # Chart dependencies
├── templates/          # Template files
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── _helpers.tpl    # Template helpers
│   └── NOTES.txt       # Post-install notes
└── .helmignore         # Files to ignore
```

1.26.3 Chart.yaml

```yaml
apiVersion: v2
name: mychart
description: A Helm chart for my application
type: application
version: 0.1.0
appVersion: "1.0.0"
```

1.26.4 values.yaml

```yaml
replicaCount: 3

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

service:
  type: ClusterIP
  port: 80

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

1.26.5 Template Example

```yaml
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-deployment
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:
        {{ .Values.image.tag }}"
        ports:
        - containerPort: 80
```

1.27 Kustomize

1.27.1 Kustomize Structure

```
├── base/
│   ├── deployment.yaml
│   ├── service.yaml
│   └── kustomization.yaml
└── overlays/
    ├── dev/
    │   └── kustomization.yaml
    └── prod/
        └── kustomization.yaml
```

1.27.2 Base kustomization.yaml

```yaml
# base/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - deployment.yaml
  - service.yaml
```

1.27.3 Overlay kustomization.yaml

```yaml
# overlays/prod/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - ../../base

namePrefix: prod-

replicas:
  - name: myapp
    count: 5

images:
  - name: myapp
    newTag: v2.0.0

patches:
  - patch: |-
      - op: replace
```

```
        path: /spec/template/spec/containers/0/resources/limits/
         memory
        value: 512Mi
    target:
      kind: Deployment
      name: myapp
```

1.27.4 Kustomize Commands

```
# Preview output
kubectl kustomize overlays/prod/

# Apply
kubectl apply -k overlays/prod/

# Delete
kubectl delete -k overlays/prod/
```

1.28 Key Concepts to Remember

1. **Rolling Update** - Default strategy, gradual replacement
2. **Recreate** - Kill all, then create new
3. **Rollout commands** - status, history, undo, pause, resume
4. **Helm** - Package manager for Kubernetes
5. **Kustomize** - Template-free configuration customization

1.29 Practice Questions

1. How do you rollback a deployment to revision 3?
2. What is the difference between maxSurge and maxUnavailable?
3. How do you install a Helm chart with custom values?
4. What command shows the rollout history of a deployment?
5. How do you apply a Kustomize overlay?

← Previous: Application Design and Build | Back to CKAD Overview | Next: Application Observability and Maintenance →

1.30 Application Observability and Maintenance

Download PDF Version

This domain covers monitoring, debugging, and maintaining applications in Kubernetes.

1.31 Probes

1.31.1 Liveness Probe

Determines if a container is running. If it fails, the container is restarted.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-liveness
spec:
  containers:
  - name: app
    image: myapp:v1
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 10
      timeoutSeconds: 5
      failureThreshold: 3
      successThreshold: 1
```

1.31.2 Readiness Probe

Determines if a container is ready to receive traffic.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-readiness
spec:
  containers:
  - name: app
    image: myapp:v1
    readinessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 5
      failureThreshold: 3
```

1.31.3 Startup Probe

Used for slow-starting containers. Disables liveness/readiness until it succeeds.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-startup
spec:
  containers:
  - name: app
    image: myapp:v1
    startupProbe:
      httpGet:
        path: /healthz
        port: 8080
      failureThreshold: 30
      periodSeconds: 10
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      periodSeconds: 10
```

1.31.4 Probe Types

Type	Description
`httpGet`	HTTP GET request to specified path and port
`tcpSocket`	TCP connection to specified port
`exec`	Execute command in container
`grpc`	gRPC health check

```yaml
# TCP Socket probe
livenessProbe:
  tcpSocket:
    port: 3306
  initialDelaySeconds: 15
  periodSeconds: 10

# Exec probe
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5

# gRPC probe
livenessProbe:
  grpc:
    port: 50051
  initialDelaySeconds: 10
```

1.31.5 Probe Parameters

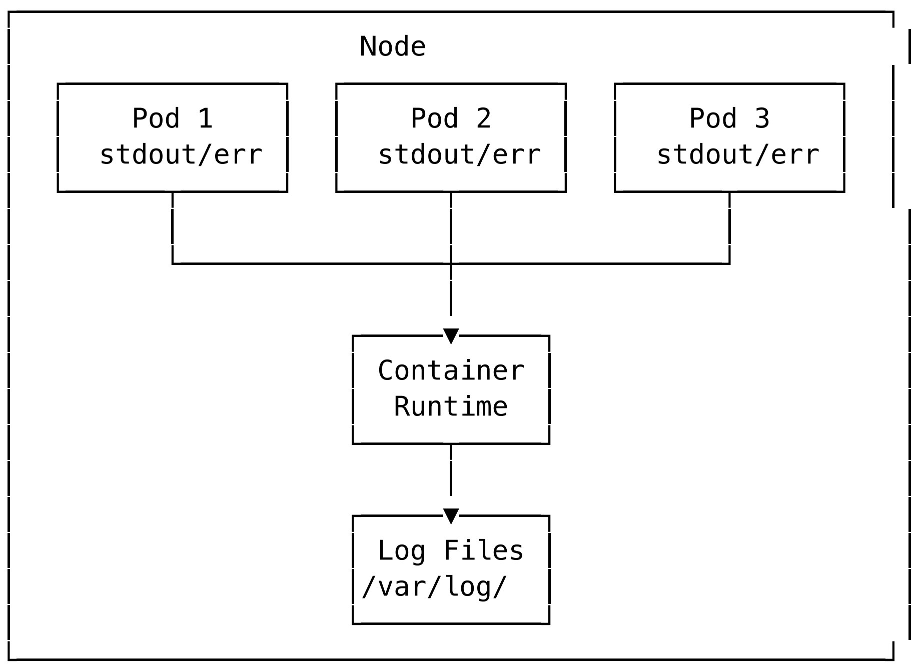
Parameter	Description	Default
`initialDelaySeconds`	Delay before first probe	0
`periodSeconds`	How often to probe	10
`timeoutSeconds`	Probe timeout	1
`failureThreshold`	Failures before action	3
`successThreshold`	Successes to be considered healthy	1

1.32 Logging

1.32.1 Viewing Logs

```
# View pod logs
kubectl logs nginx

# View specific container logs
kubectl logs nginx -c sidecar

# Follow logs
kubectl logs -f nginx

# View previous container logs (after restart)
kubectl logs nginx --previous

# View last N lines
kubectl logs nginx --tail=100

# View logs since time
kubectl logs nginx --since=1h
kubectl logs nginx --since-time=2024-01-01T00:00:00Z

# View logs from all pods with label
kubectl logs -l app=nginx

# View logs from all containers in pod
kubectl logs nginx --all-containers
```

1.32.2 Logging Architecture



1.33 Debugging

1.33.1 Debug Commands

```
# Describe pod (events, status)
kubectl describe pod nginx

# Get pod details
kubectl get pod nginx -o yaml
kubectl get pod nginx -o wide

# Check events
kubectl get events --sort-by='.lastTimestamp'
kubectl get events --field-selector involvedObject.name=nginx

# Execute command in container
kubectl exec nginx -- ls /app
kubectl exec -it nginx -- /bin/sh

# Copy files to/from container
kubectl cp nginx:/var/log/app.log ./app.log
kubectl cp ./config.yaml nginx:/app/config.yaml

# Port forward
kubectl port-forward pod/nginx 8080:80
kubectl port-forward svc/nginx 8080:80

# Debug with ephemeral container
kubectl debug nginx -it --image=busybox --target=nginx
```

1.33.2 Common Issues

Issue	Debug Steps
ImagePullBackOff	Check image name, registry access, pull secrets
CrashLoopBackOff	Check logs, probe configuration, resource limits
Pending	Check events, node resources, taints/tolerations
OOMKilled	Increase memory limits
CreateContainerConfigError	Check ConfigMaps, Secrets references

1.33.3 Pod Status Phases

Phase	Description
`Pending`	Pod accepted but not running
`Running`	Pod bound to node, containers running
`Succeeded`	All containers terminated successfully
`Failed`	All containers terminated, at least one failed

Phase	Description
`Unknown`	Pod state cannot be determined

1.34 Monitoring

1.34.1 Resource Metrics

```
# View node resource usage
kubectl top nodes

# View pod resource usage
kubectl top pods
kubectl top pods -A
kubectl top pods --containers

# Sort by CPU/memory
kubectl top pods --sort-by=cpu
kubectl top pods --sort-by=memory
```

1.34.2 Metrics Server

Required for `kubectl top` commands:

```
# Check if metrics server is running
kubectl get pods -n kube-system | grep metrics-server

# Install metrics server (if needed)
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
        server/releases/latest/download/components.yaml
```

1.35 Application Maintenance

1.35.1 Updating Applications

```
# Update image
kubectl set image deployment/nginx nginx=nginx:1.22

# Update environment variable
kubectl set env deployment/nginx ENV=production

# Update resources
kubectl set resources deployment/nginx --
        limits=cpu=200m,memory=512Mi

# Patch resource
kubectl patch deployment nginx -p '{"spec":{"replicas":5}}'
```

1.35.2 Scaling

```
# Manual scaling
kubectl scale deployment nginx --replicas=5

# Autoscaling
kubectl autoscale deployment nginx --min=2 --max=10 --cpu-
        percent=80
```

1.35.3 HorizontalPodAutoscaler

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

1.36 Key Concepts to Remember

1. **Liveness** - Is the container running? Restart if not
2. **Readiness** - Is the container ready for traffic?
3. **Startup** - For slow-starting containers
4. **kubectl logs** - View container output
5. **kubectl describe** - Detailed resource info with events
6. **kubectl top** - Resource usage (requires metrics-server)

1.37 Practice Questions

1. What happens when a liveness probe fails?
2. How do you view logs from a previous container instance?
3. What is the difference between readiness and liveness probes?

4. How do you execute a command in a running container?
5. What probe type would you use for a database container?

← Previous: Application Deployment | Back to CKAD Overview | Next: Application Environment, Configuration and Security →

1.38 Application Environment, Configuration and Security

Download PDF Version

This domain covers configuring applications, managing secrets, and implementing security in Kubernetes.

1.39 ConfigMaps

1.39.1 Creating ConfigMaps

```
# From literal values
kubectl create configmap app-config --from-literal=ENV=production
        --from-literal=LOG_LEVEL=info

# From file
kubectl create configmap app-config --from-file=config.properties

# From directory
kubectl create configmap app-config --from-file=config/

# From env file
kubectl create configmap app-config --from-env-file=app.env
```

1.39.2 ConfigMap YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  ENV: production
  LOG_LEVEL: info
  config.json: |
    {
      "database": "mysql",
      "port": 3306
    }
```

1.39.3 Using ConfigMaps

As Environment Variables:

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: myapp:v1
    env:
    - name: ENVIRONMENT
      valueFrom:
        configMapKeyRef:
          name: app-config
          key: ENV
    envFrom:
```

```
    - configMapRef:
        name: app-config
```

As Volume:

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: myapp:v1
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: app-config
      items:
      - key: config.json
        path: app-config.json
```

1.40 Secrets

1.40.1 Creating Secrets

```
# From literal values
kubectl create secret generic db-secret --from-
        literal=username=admin --from-literal=password=secret123

# From file
kubectl create secret generic tls-secret --from-file=tls.crt --
        from-file=tls.key

# Docker registry secret
kubectl create secret docker-registry regcred \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=user \
  --docker-password=pass \
  --docker-email=user@example.com

# TLS secret
kubectl create secret tls tls-secret --cert=tls.crt --key=tls.key
```

1.40.2 Secret YAML

```
apiVersion: v1
kind: Secret
metadata:
```

```yaml
  name: db-secret
type: Opaque
data:
  username: YWRtaW4=      # base64 encoded
  password: c2VjcmV0MTIz  # base64 encoded
---
# Using stringData (auto-encoded)
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
stringData:
  username: admin
  password: secret123
```

1.40.3 Secret Types

Type	Description
`Opaque`	Generic secret (default)
`kubernetes.io/dockerconfigjson`	Docker registry credentials
`kubernetes.io/tls`	TLS certificate and key
`kubernetes.io/basic-auth`	Basic authentication
`kubernetes.io/ssh-auth`	SSH authentication

1.40.4 Using Secrets

As Environment Variables:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: myapp:v1
    env:
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: password
    envFrom:
    - secretRef:
        name: db-secret
```

As Volume:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: myapp:v1
    volumeMounts:
    - name: secret-volume
      mountPath: /etc/secrets
      readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: db-secret
      defaultMode: 0400
```

1.41 ServiceAccounts

1.41.1 Creating ServiceAccounts

```
kubectl create serviceaccount my-sa
```

1.41.2 ServiceAccount YAML

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
automountServiceAccountToken: false
```

1.41.3 Using ServiceAccounts

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  serviceAccountName: my-sa
  automountServiceAccountToken: true
  containers:
  - name: app
    image: myapp:v1
```

1.42 Security Context

1.42.1 Pod-level Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
    runAsNonRoot: true
  containers:
  - name: app
    image: myapp:v1
```

1.42.2 Container-level Security Context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  containers:
  - name: app
    image: myapp:v1
    securityContext:
      runAsUser: 1000
      runAsNonRoot: true
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
        add:
        - NET_BIND_SERVICE
```

1.42.3 Security Context Fields

Field	Level	Description
`runAsUser`	Pod/Container	UID to run as
`runAsGroup`	Pod/Container	GID to run as
`runAsNonRoot`	Pod/Container	Must run as non-root
`fsGroup`	Pod	Group for volumes
`readOnlyRootFilesystem`	Container	Read-only root FS
`allowPrivilegeEscalation`	Container	Prevent privilege escalation

Field	Level	Description
`capabilities`	Container	Linux capabilities

1.43 Resource Requirements

1.43.1 Requests and Limits

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

1.43.2 CPU Units

Value	Description
`1`	1 CPU core
`500m`	0.5 CPU core (500 millicores)
`100m`	0.1 CPU core

1.43.3 Memory Units

Value	Description
`128Mi`	128 Mebibytes
`1Gi`	1 Gibibyte
`256M`	256 Megabytes

1.43.4 QoS Classes

Class	Condition
Guaranteed	requests = limits for all containers
Burstable	At least one request or limit set
BestEffort	No requests or limits set

1.44 LimitRange

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: resource-limits
spec:
  limits:
  - type: Container
    default:
      cpu: "500m"
      memory: "256Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    max:
      cpu: "2"
      memory: "1Gi"
    min:
      cpu: "50m"
      memory: "64Mi"
```

1.45 ResourceQuota

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
spec:
  hard:
    requests.cpu: "4"
    requests.memory: "8Gi"
    limits.cpu: "8"
    limits.memory: "16Gi"
    pods: "10"
    configmaps: "10"
    secrets: "10"
    persistentvolumeclaims: "5"
```

1.46 Admission Controllers

1.46.1 Pod Security Standards

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

Level	Description
`privileged`	Unrestricted
`baseline`	Minimally restrictive
`restricted`	Highly restrictive

1.47 Key Concepts to Remember

1. **ConfigMaps** - Non-sensitive configuration data
2. **Secrets** - Sensitive data (base64 encoded, not encrypted)
3. **SecurityContext** - Pod/container security settings
4. **Resources** - requests (scheduling) vs limits (enforcement)
5. **ServiceAccounts** - Identity for pods

1.48 Practice Questions

1. How do you create a ConfigMap from a file?
2. What is the difference between `data` and `stringData` in Secrets?
3. How do you mount a Secret as a volume with specific permissions?
4. What QoS class is assigned when requests equal limits?
5. How do you prevent a container from running as root?

← Previous: Application Observability and Maintenance | Back to CKAD Overview | Next: Services and Networking →

1.49 Services and Networking

Download PDF Version

This domain covers Kubernetes networking concepts, Services, and Ingress.

1.50 Services

1.50.1 Service Types

Type	Description
`ClusterIP`	Internal cluster IP (default)
`NodePort`	Exposes on each node’s IP at a static port
`LoadBalancer`	External load balancer (cloud provider)
`ExternalName`	Maps to external DNS name

1.50.2 ClusterIP Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
```

1.50.3 NodePort Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30080  # Optional: 30000-32767
```

1.50.4 LoadBalancer Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
```

1.50.5 ExternalName Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: external-db
spec:
  type: ExternalName
  externalName: db.example.com
```

1.50.6 Headless Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless-service
spec:
  clusterIP: None
  selector:
    app: myapp
  ports:
  - port: 80
    targetPort: 8080
```

1.50.7 Creating Services Imperatively

```bash
# Expose deployment
kubectl expose deployment nginx --port=80 --target-port=8080 --
        type=ClusterIP

# Expose pod
kubectl expose pod nginx --port=80 --target-port=8080

# Create service without selector
kubectl create service clusterip my-svc --tcp=80:8080

# Generate YAML
kubectl expose deployment nginx --port=80 --dry-run=client -o
        yaml > svc.yaml
```

1.51 DNS in Kubernetes

1.51.1 Service DNS

```
<service-name>.<namespace>.svc.cluster.local

Examples:
- my-service.default.svc.cluster.local
- my-service.default.svc
- my-service.default
- my-service (within same namespace)
```

1.51.2 Pod DNS

```
<pod-ip-dashed>.<namespace>.pod.cluster.local
```

```
Example:
- 10-244-0-5.default.pod.cluster.local
```

1.51.3 DNS Resolution Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dns-test
spec:
  containers:
  - name: test
    image: busybox:1.36
    command: ['sleep', '3600']
---
# Test DNS
kubectl exec dns-test -- nslookup my-service
kubectl exec dns-test -- nslookup my-
        service.default.svc.cluster.local
```

1.52 Network Policies

1.52.1 Default Behavior

By default, all pods can communicate with all other pods.

1.52.2 Deny All Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

1.52.3 Deny All Egress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-egress
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

1.52.4 Allow Specific Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    - namespaceSelector:
        matchLabels:
          name: production
    ports:
    - protocol: TCP
      port: 8080
```

1.52.5 Allow Egress to Specific Pods

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-db-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Egress
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database
    ports:
    - protocol: TCP
      port: 5432
  - to:  # Allow DNS
    - namespaceSelector: {}
```

```yaml
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

1.52.6 Network Policy Selectors

Selector	Description
`podSelector`	Select pods by labels
`namespaceSelector`	Select namespaces by labels
`ipBlock`	Select by IP CIDR

```yaml
# IP Block example
ingress:
- from:
  - ipBlock:
      cidr: 10.0.0.0/8
      except:
      - 10.0.1.0/24
```

1.53 Ingress

1.53.1 Ingress Resource

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: myapp-service
            port:
              number: 80
```

1.53.2 Multiple Hosts

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

1.53.3 Path-based Routing

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

1.53.4 TLS Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - myapp.example.com
    secretName: tls-secret
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: myapp-service
            port:
              number: 80
```

1.53.5 Path Types

Type	Description
`Exact`	Exact match of the URL path
`Prefix`	Matches based on URL path prefix
`ImplementationSpecific`	Depends on IngressClass

1.53.6 Creating Ingress Imperatively

```bash
# Create ingress
kubectl create ingress my-ingress \
  --rule="myapp.example.com/=myapp-service:80" \
  --class=nginx

# With TLS
kubectl create ingress my-ingress \
  --rule="myapp.example.com/=myapp-service:80,tls=tls-secret" \
  --class=nginx
```

1.54 Port Forwarding

```
# Forward pod port
kubectl port-forward pod/nginx 8080:80

# Forward service port
kubectl port-forward svc/nginx 8080:80

# Forward deployment port
kubectl port-forward deployment/nginx 8080:80

# Listen on all interfaces
kubectl port-forward --address 0.0.0.0 pod/nginx 8080:80
```

1.55 Key Concepts to Remember

1. **ClusterIP** - Default, internal only
2. **NodePort** - External access via node IP:port
3. **LoadBalancer** - Cloud provider load balancer
4. **Network Policies** - Default allow, explicit deny
5. **Ingress** - HTTP/HTTPS routing, requires controller

1.56 Practice Questions

1. What is the default Service type in Kubernetes?
2. How do you create a Service that exposes a deployment on port 80?
3. What happens to traffic if no Network Policy exists?
4. How do you route traffic based on URL path using Ingress?
5. What is the DNS name format for a Service?

← Previous: Application Environment, Configuration and Security | Back to CKAD Overview | Next: Sample Practice Questions →

1.57 Sample Practice Questions

Download PDF Version

> **Disclaimer**: These are sample practice questions created for study purposes only. They are NOT actual exam questions and are designed to help you test your understanding of CKAD concepts. Real exam questions may differ in format and content.

1.58 Practice Resources

Before attempting these questions, we highly recommend practicing on:

- **Killercoda CKAD Scenarios** ⭐ Free hands-on practice environments
- **killer.sh CKAD Simulator** - Included with exam registration

1.59 Instructions

- The CKAD exam is **performance-based** (hands-on), not multiple choice
- Practice these scenarios in a real Kubernetes cluster

- Time yourself - aim for efficiency
- Use imperative commands when possible to save time

1.60 Section 1: Application Design and Build (20%)

1.60.1 Question 1.1 - Create a Job

Create a Job named `pi-calculator` that: - Uses the image `perl:5.34` - Runs the command `perl -Mbignum=bpi -wle 'print bpi(2000)'` - Completes 3 times successfully - Runs 2 pods in parallel - Has a backoff limit of 4

Show Solution

```
# Generate base YAML
kubectl create job pi-calculator --image=perl:5.34 --dry-
        run=client -o yaml -- perl -Mbignum=bpi -wle 'print
        bpi(2000)' > job.yaml

# Edit to add completions, parallelism, backoffLimit

apiVersion: batch/v1
kind: Job
metadata:
  name: pi-calculator
spec:
  completions: 3
  parallelism: 2
  backoffLimit: 4
  template:
    spec:
      containers:
      - name: pi-calculator
        image: perl:5.34
        command: ["perl", "-Mbignum=bpi", "-wle", "print
        bpi(2000)"]
      restartPolicy: Never

kubectl apply -f job.yaml
```

1.60.2 Question 1.2 - Create a CronJob

Create a CronJob named `backup-job` that: - Runs every day at 2:30 AM - Uses image `busybox:1.36` - Runs command `echo "Backup completed at $(date)"` - Keeps 3 successful job history - Keeps 1 failed job history

Show Solution

```
kubectl create cronjob backup-job --image=busybox:1.36 --
        schedule="30 2 * * *" -- /bin/sh -c 'echo "Backup
        completed at $(date)"'

# Or with YAML for history limits:

apiVersion: batch/v1
kind: CronJob
metadata:
  name: backup-job
spec:
  schedule: "30 2 * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: busybox:1.36
            command: ["/bin/sh", "-c", "echo
        \"Backup completed at $(date)\""]
          restartPolicy: OnFailure
```

1.60.3 Question 1.3 - Multi-Container Pod

Create a Pod named `app-with-sidecar` with: - Main container: `nginx:1.21` named `main-app` - Sidecar container: `busybox:1.36` named `log-agent` that runs `tail -f /var/log/nginx/access.log` - Both containers share a volume mounted at `/var/log/nginx`

Show Solution

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-sidecar
spec:
  containers:
  - name: main-app
    image: nginx:1.21
    volumeMounts:
    - name: logs
      mountPath: /var/log/nginx
  - name: log-agent
    image: busybox:1.36
    command: ["tail", "-f", "/var/log/nginx/access.log"]
    volumeMounts:
    - name: logs
      mountPath: /var/log/nginx
  volumes:
```

```yaml
  - name: logs
    emptyDir: {}
```

1.61 Section 2: Application Deployment (20%)

1.61.1 Question 2.1 - Create and Scale Deployment

1. Create a Deployment named `web-app` with image `nginx:1.21` and 3 replicas
2. Update the image to `nginx:1.22`
3. Check the rollout status
4. Rollback to the previous version

Show Solution

```bash
# Create deployment
kubectl create deployment web-app --image=nginx:1.21 --replicas=3

# Update image
kubectl set image deployment/web-app nginx=nginx:1.22

# Check rollout status
kubectl rollout status deployment/web-app

# View history
kubectl rollout history deployment/web-app

# Rollback
kubectl rollout undo deployment/web-app
```

1.61.2 Question 2.2 - Deployment Strategy

Create a Deployment named `rolling-app` with: - Image: `nginx:1.21` - 4 replicas - Rolling update strategy with maxSurge=1 and maxUnavailable=1

Show Solution

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rolling-app
spec:
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: rolling-app
  template:
    metadata:
      labels:
        app: rolling-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
```

1.61.3 Question 2.3 - Helm Operations

1. Add the bitnami repository
2. Search for nginx chart
3. Install nginx chart with release name `my-nginx`
4. List all releases
5. Uninstall the release

Show Solution

```bash
# Add repository
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update

# Search
helm search repo nginx

# Install
helm install my-nginx bitnami/nginx

# List releases
helm list

# Uninstall
helm uninstall my-nginx
```

1.62 Section 3: Application Observability and Maintenance (15%)

1.62.1 Question 3.1 - Configure Probes

Create a Pod named `health-check-pod` with: - Image: `nginx:1.21` - Liveness probe: HTTP GET on path `/` port 80, initial delay 10s, period 5s - Readiness probe: HTTP GET on path `/` port 80, initial delay 5s, period 3s

Show Solution

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: health-check-pod
```

```yaml
spec:
  containers:
  - name: nginx
    image: nginx:1.21
    ports:
    - containerPort: 80
    livenessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 10
      periodSeconds: 5
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 3
```

1.62.2 Question 3.2 - Debugging

A Pod named `broken-pod` is not running correctly. Debug and fix it.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: broken-pod
spec:
  containers:
  - name: app
    image: nginx:latest
    command: ["nginx", "-g", "daemon off;"]
    resources:
      limits:
        memory: "10Mi"
```

Show Solution

```bash
# Check pod status
kubectl get pod broken-pod

# Check events and details
kubectl describe pod broken-pod

# Check logs
kubectl logs broken-pod

# The issue is likely OOMKilled due to low memory limit
# Fix by increasing memory limit:
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: broken-pod
spec:
  containers:
  - name: app
    image: nginx:latest
    resources:
      limits:
        memory: "128Mi"
```

1.62.3 Question 3.3 - View Logs

1. View logs of pod `nginx` in namespace `web`
2. View logs of the previous container instance
3. Follow logs in real-time
4. View last 50 lines

Show Solution

```
# View logs
kubectl logs nginx -n web

# Previous container
kubectl logs nginx -n web --previous

# Follow logs
kubectl logs -f nginx -n web

# Last 50 lines
kubectl logs nginx -n web --tail=50
```

1.63 Section 4: Application Environment, Configuration and Security (25%)

1.63.1 Question 4.1 - ConfigMap and Secret

1. Create a ConfigMap named `app-config` with:
   - `APP_ENV=production`
   - `LOG_LEVEL=info`
2. Create a Secret named `db-secret` with:
   - `DB_USER=admin`
   - `DB_PASS=secret123`
3. Create a Pod that uses both as environment variables

Show Solution

```
# Create ConfigMap
kubectl create configmap app-config --from-
        literal=APP_ENV=production --from-literal=LOG_LEVEL=info
```

```
# Create Secret
kubectl create secret generic db-secret --from-
        literal=DB_USER=admin --from-literal=DB_PASS=secret123
```

```
apiVersion: v1
kind: Pod
metadata:
  name: app-pod
spec:
  containers:
  - name: app
    image: nginx:1.21
    envFrom:
    - configMapRef:
        name: app-config
    - secretRef:
        name: db-secret
```

1.63.2 Question 4.2 - Security Context

Create a Pod named `secure-pod` that: - Runs as user ID 1000 - Runs as group ID 3000 - Has a read-only root filesystem - Cannot escalate privileges

Show Solution

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
  containers:
  - name: app
    image: nginx:1.21
    securityContext:
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
```

1.63.3 Question 4.3 - Resource Limits

Create a Pod named `resource-pod` with: - Image: `nginx:1.21` - CPU request: 100m, limit: 200m - Memory request: 64Mi, limit: 128Mi

Show Solution

```
kubectl run resource-pod --image=nginx:1.21 --dry-run=client -o
        yaml > pod.yaml
# Edit to add resources
```

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: resource-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.21
    resources:
      requests:
        cpu: "100m"
        memory: "64Mi"
      limits:
        cpu: "200m"
        memory: "128Mi"
```

1.64 Section 5: Services and Networking (20%)

1.64.1 Question 5.1 - Create Services

1. Create a Deployment named `web` with image `nginx:1.21` and 3 replicas
2. Expose it as a ClusterIP Service on port 80
3. Expose it as a NodePort Service on port 30080

Show Solution

```
# Create deployment
kubectl create deployment web --image=nginx:1.21 --replicas=3

# ClusterIP service
kubectl expose deployment web --port=80 --target-port=80 --
        name=web-clusterip

# NodePort service
kubectl expose deployment web --port=80 --target-port=80 --
        type=NodePort --name=web-nodeport

# Or specify nodePort:
kubectl create service nodeport web-nodeport --tcp=80:80 --node-
        port=30080
```

1.64.2 Question 5.2 - Network Policy

Create a NetworkPolicy named `api-policy` in namespace `default` that: - Applies to pods with label `app=api` - Allows ingress only from pods with label `app=frontend` - Allows ingress only on port 8080

Show Solution

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: api
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080
```

1.64.3 Question 5.3 - Ingress

Create an Ingress named `web-ingress` that: - Routes `app.example.com/api` to service `api-service` port 80 - Routes `app.example.com/web` to service `web-service` port 80 - Uses ingress class `nginx`

Show Solution

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 80
      - path: /web
        pathType: Prefix
        backend:
          service:
            name: web-service
            port:
              number: 80
```

1.65 Exam Tips

1. **Use aliases**: `alias k=kubectl`
2. **Enable auto-completion**: `source <(kubectl completion bash)`
3. Use **`--dry-run=client -o yaml`** to generate YAML templates
4. **Bookmark important docs** before the exam
5. **Practice on Killercoda** for free hands-on scenarios
6. **Time management**: Don’t spend too long on any single question
7. **Use imperative commands** when possible to save time

1.66 Additional Practice

- Killercoda CKAD Scenarios - Free interactive scenarios
- killer.sh - Exam simulator (included with registration)
- Kubernetes Documentation - Allowed during exam

← Back to CKAD Overview
