CKAD Practice Questions

Certified Kubernetes Application Developer Exam Preparation

Table of Contents

1. Application Design and Build (20%)

Container Images

Question 1: Write a Dockerfile for a Node.js application that: - Uses node:18-alpine as base image - Sets working directory to /app - Copies package.json and installs dependencies - Copies application code - Exposes port 3000 - Runs the application with `npm start`

Question 2: Build a multi-stage Dockerfile for a Go application that: - First stage: builds the binary using golang:1.21 - Second stage: uses alpine:latest for minimal final image - Only copies the compiled binary to the final image

Question 3: Create a Pod that uses an image from a private registry. The registry credentials are stored in a Secret named `regcred`.

Question 4: Modify an existing Deployment to use a different image tag and record the change. Then rollback to the previous version.

Workload Resources

Question 5: Create a Deployment named `web-app` with: - 3 replicas - Image: nginx:1.21 - Label: app=web, tier=frontend - Resource requests: memory=64Mi, cpu=100m - Resource limits: memory=128Mi, cpu=200m

Question 6: Create a DaemonSet named `log-collector` that runs on all nodes except the master. Use image: fluent/fluent-bit:latest and label: app=logging.

Question 7: Create a CronJob named `backup-db` that: - Runs every day at 2 AM (0 2 * * *) - Uses image: mysql:8.0 - Runs command: mysqldump -u root -p$MYSQL_PASSWORD mydb > /backup/db.sql - Keeps 3 successful job completions - Keeps 1 failed job completion

Question 8: Create a Job named `data-processor` that: - Runs 5 times in parallel - Completes successfully 10 times total - Uses image: busybox - Runs command: echo "Processing batch $HOSTNAME"

Question 9: Convert the following Deployment to use a StatefulSet for a database application that needs stable network identities:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
```

Multi-Container Pods

Question 10: Create a Pod named `web-with-git` with: - Init container: clones a git repo (use alpine/git image) into /data volume - Main container: nginx serving files from /usr/share/nginx/html mounted from /data volume - Use emptyDir volume named `git-repo`

Question 11: Create a Pod with the adapter pattern: - Main container: writes custom formatted logs to /var/log/app/custom.log - Adapter container: reads from /var/log/app/custom.log and reformats to JSON, writing to /var/log/app/json.log - Use busybox for both containers with appropriate commands

Question 12: Create a Pod implementing the ambassador pattern: - Main application container on port 8080 - Ambassador container (nginx) that proxies requests to the main container - Ambassador exposes port 80

Question 13: Create a Pod with two init containers that run sequentially: - First init: checks if a ConfigMap named `app-config` exists (use kubectl in init container) - Second init: waits for a Service named `database` to resolve (use nslookup) - Main container: nginx

Volumes

Question 14: Create a Pod with: - A PersistentVolumeClaim requesting 1Gi of storage with ReadWriteOnce access - Mount the PVC at /data in the container - Use nginx:alpine image

Question 15: Create a Pod that uses: - emptyDir volume with memory medium (tmpfs) at /cache - hostPath volume mounting /var/log from the host to /host-logs in container - Image: busybox running `sleep 3600`

Question 16: Create a Pod with multiple containers sharing data: - Container 1: writes timestamp to /shared/data.txt every 5 seconds - Container 2: reads and displays content from /shared/data.txt every 10 seconds - Use emptyDir volume

---

2. Application Deployment (20%)

Deployment Strategies

Question 17: Create a Deployment with a rolling update strategy: - Name: frontend - Replicas: 6 - Image: nginx:1.20 - maxSurge: 2 - maxUnavailable: 1 - Update the image to nginx:1.21 and observe the rollout

Question 18: Implement a blue/green deployment: - Create two Deployments: blue (nginx:1.20) and green (nginx:1.21) - Create a Service that initially points to blue - Switch traffic to green by updating the Service selector

Question 19: Implement a canary deployment: - Main Deployment: 9 replicas with image nginx:1.20, label version=stable - Canary Deployment: 1 replica with image nginx:1.21, label version=canary - Service selects both using app=frontend label - Calculate the traffic percentage to canary

Question 20: Perform a rolling update on a Deployment: - Update image from nginx:1.20 to nginx:1.21 - Pause the rollout after 3 pods are updated - Verify the status - Resume the rollout

Rollouts and Rollbacks

Question 21: A Deployment update has failed. Write commands to: - Check the rollout status - View rollout history - Rollback to the previous version - Rollback to a specific revision number 3

Question 22: Create a Deployment and perform multiple updates while recording each change. Then view the history and rollback to revision 2.

Question 23: Update a Deployment's image but the new image is broken. The readiness probe prevents traffic from reaching the new pods. Rollback the deployment.

Helm

Question 24: Install nginx-ingress using Helm: - Add the ingress-nginx repository - Update the repo - Install the chart in namespace `ingress-nginx` - Create the namespace if it doesn't exist

Question 25: Create a simple Helm chart for a web application: - Chart name: myapp - Include templates for: Deployment, Service, ConfigMap - Add values.yaml with configurable replicas and image - Use template functions to reference values

Question 26: Upgrade a Helm release: - Install nginx chart with 2 replicas - Upgrade to 5 replicas using --set flag - Rollback the release to previous version

Question 27: List all Helm releases in all namespaces and uninstall a specific release named `myapp` from namespace `production`.

Kustomize

Question 28: Create a Kustomize structure:

```
base/
  deployment.yaml
  service.yaml
  kustomization.yaml
overlays/
  dev/
    kustomization.yaml
  prod/
    kustomization.yaml
```

Use Kustomize to deploy with different replica counts for dev (2) and prod (5).

Question 29: Use Kustomize to: - Create a base configuration with a Deployment and ConfigMap - Create dev overlay that adds a `dev` suffix to all resource names - Create prod overlay that adds environment label `env=production`

Question 30: Apply a Kustomize configuration and view the generated manifests before applying:

```
kubectl kustomize overlays/dev
kubectl apply -k overlays/dev
```

---

3. Application Observability and Maintenance (15%)

Probes and Health Checks

Question 31: Create a Pod with all three probe types: - Liveness probe: HTTP GET on /healthz port 8080, fails after 3 attempts - Readiness probe: TCP socket on port 8080, initial delay 5s - Startup probe: exec command `cat /tmp/ready`, period 10s, failure threshold 30

Question 32: Debug a Pod where the liveness probe is failing: - The application takes 60 seconds to start - Current liveness probe has initialDelaySeconds: 10 - Fix the probe configuration

Question 33: Create a Deployment where: - Readiness probe prevents traffic until application is ready - Liveness probe restarts container if application becomes unresponsive - Use httpGet probes on different endpoints

Monitoring and Logging

Question 34: Get logs from: - A specific pod - A specific container in a multi-container pod - Previous instance of a crashed container - Follow logs in real-time - Last 50 lines of logs

Question 35: A pod has multiple containers. Write commands to: - Get logs from all containers - Get logs from a specific container named `sidecar` - Stream logs from the main container

Question 36: Install metrics-server and retrieve: - CPU and memory usage for all pods in a namespace - Resource usage for a specific pod - Top nodes by CPU usage

Question 37: Debug an application that's not responding: - Check pod status - View pod events - Get detailed pod description - Check container logs - Execute commands inside the container to investigate

Container Logs and Debugging

Question 38: A pod is in CrashLoopBackOff state. Write the debugging steps: - Check pod status - View pod events - Get logs from the crashed container - Describe the pod - Check if resource limits are too low

Question 39: Debug a pod stuck in Pending state: - Check if there are resource constraints - View pod events - Check if PVC is bound - Verify node selectors and taints/tolerations

Question 40: An application is running but responding slowly. Debug using: - kubectl top to check resource usage - kubectl exec to run diagnostic commands inside container - Check application logs for errors

API Deprecations

Question 41: You have a manifest using deprecated API version `extensions/v1beta1` for Ingress. Update it to the current stable API version `networking.k8s.io/v1`.

Question 42: Check which API versions are deprecated in your cluster:

```
kubectl api-resources
kubectl explain <resource> --api-version=<version>
```

---

4. Application Environment, Configuration and Security (25%)

Custom Resource Definitions (CRDs)

Question 43: Create a simple CRD for a custom resource called `Website`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: websites.example.com
spec:
  group: example.com
  versions:
  - name: v1
    served: true
    storage: true
  scope: Namespaced
  names:
    plural: websites
    singular: website
    kind: Website
```

Then create an instance of this custom resource.

Question 44: List all CRDs in the cluster and describe a specific CRD named `certificates.cert-manager.io`.

ConfigMaps and Secrets

Question 45: Create a ConfigMap from: - Literal values: `DB_HOST=mysql`, `DB_PORT=3306` - A file containing application.properties - An entire directory of config files

Question 46: Create a Pod that uses ConfigMap data as: - Environment variables - Volume mounted files at /etc/config - A single environment variable from a specific key

Question 47: Create a Secret containing: - Username: admin - Password: super-secret-password Create a Pod that uses this Secret as environment variables.

Question 48: Create a Secret from a TLS certificate and key file:

```
kubectl create secret tls my-tls-secret --cert=path/to/cert --key=path/to/key
```

Mount this Secret in a pod at /etc/tls.

Question 49: Update a ConfigMap without recreating pods: - Modify a ConfigMap - Verify that pods using it as a volume see updated values - Note: environment variable usage doesn't auto-update

Question 50: Create a Pod that uses: - ConfigMap for non-sensitive configuration (database host) - Secret for sensitive data (database password) - Both as environment variables and volume mounts

Resource Requirements and Limits

Question 51: Create a Pod with: - CPU request: 100m, limit: 200m - Memory request: 128Mi, limit: 256Mi - Observe what happens when the pod exceeds memory limit

Question 52: Create a LimitRange in namespace `dev` that sets: - Default CPU request: 100m - Default CPU limit: 500m - Default memory request: 64Mi - Default memory limit: 128Mi

Question 53: Create a ResourceQuota in namespace `production`: - Maximum 10 pods - Total CPU requests: 2 cores - Total memory requests: 4Gi - Maximum 5 services

Question 54: Debug a pod that can't be scheduled due to insufficient resources. Check node capacity and current usage.

ServiceAccounts

Question 55: Create a ServiceAccount named `app-sa` and create a Pod that uses this ServiceAccount.

Question 56: Create a ServiceAccount with an Image Pull Secret: - Create a docker-registry secret - Link it to the ServiceAccount - Use the ServiceAccount in a Pod

Question 57: View the token and certificate of a ServiceAccount:

```
kubectl get serviceaccount <sa-name> -o yaml
kubectl describe secret <sa-token-secret>
```

RBAC (Authentication and Authorization)

Question 58: Create a Role that allows: - Getting, listing, watching pods - Creating, deleting services - In namespace `development`

Question 59: Create a RoleBinding that: - Binds the Role from Question 58 - To a ServiceAccount named `dev-user` - In namespace `development`

Question 60: Create a ClusterRole that allows reading pods across all namespaces and bind it to a user named `readonly-user`.

Question 61: Debug authorization issues: - Check if a ServiceAccount can perform an action using `kubectl auth can-i` - Verify Role and RoleBinding configurations

Security Contexts

Question 62: Create a Pod with security context: - Run as user ID 1000 - Run as group ID 3000 - fsGroup 2000 - Read-only root filesystem - Drop ALL capabilities

Question 63: Create a Pod that: - Runs as non-root user - Has privilege escalation disabled - Uses a specific SELinux level

Question 64: Compare Pod-level vs Container-level security contexts. Create a Pod where: - Pod security context sets runAsUser: 1000 - One container overrides with runAsUser: 2000

Admission Controllers

Question 65: Understand Pod Security Standards: - Create a namespace with label `pod-security.kubernetes.io/enforce=baseline` - Try to create a privileged pod - Observe the admission denial

Question 66: Apply Pod Security Admission: - Set namespace to enforce `restricted` standard - Set namespace to warn on `baseline` violations - Try deploying various workloads

---

5. Services and Networking (20%)

Services

Question 67: Create a ClusterIP Service named `backend-svc`: - Selector: app=backend - Port: 80, targetPort: 8080 - Create a Deployment with matching labels

Question 68: Create a NodePort Service: - Name: web-nodeport - Selector: app=web - Port: 80 - NodePort: 30080 - Access the service from outside the cluster

Question 69: Create a LoadBalancer Service for a cloud environment: - Name: external-lb - Selector: app=frontend - Port: 80, targetPort: 8080

Question 70: Create a headless Service (ClusterIP: None): - For a StatefulSet named `mysql` - Allows direct pod-to-pod communication - DNS returns pod IPs instead of service IP

Question 71: Create an ExternalName Service: - Maps to an external database: `db.example.com` - Name: external-db - Access it from within the cluster

Service Discovery

Question 72: Create two Deployments with Services and demonstrate DNS-based service discovery: - Frontend pod accessing backend service using DNS name - Use both short name and FQDN

Question 73: From a Pod, use nslookup to: - Resolve a service in the same namespace - Resolve a service in a different namespace - Understand the DNS naming pattern

NetworkPolicies

Question 74: Create a NetworkPolicy that: - Denies all ingress traffic to pods with label app=database - Allows ingress only from pods with label app=backend on port 3306 - In namespace `production`

Question 75: Create a NetworkPolicy that: - Denies all egress traffic from pods with label app=frontend - Allows egress to pods with label app=backend on port 8080 - Allows DNS (port 53 UDP)

Question 76: Create a default deny-all NetworkPolicy: - Denies all ingress and egress traffic - Apply to namespace `secure`

Question 77: Implement namespace isolation: - Create NetworkPolicy allowing pods in namespace `frontend` to access pods in namespace `backend` - Use namespaceSelector

Question 78: Debug NetworkPolicy issues: - Pods can't communicate as expected - Check if network plugin supports NetworkPolicies - Verify policy selectors and rules

Ingress

Question 79: Create an Ingress resource: - Host: myapp.example.com - Path: /api routes to service backend-svc:8080 - Path: / routes to service frontend-svc:80

Question 80: Create an Ingress with TLS: - Use Secret `my-tls-secret` for TLS termination - Host: secure.example.com - Backend service: web-svc:443

Question 81: Create path-based routing Ingress: - /app1 → service1:8080 - /app2 → service2:8080 - Default backend → default-svc:80

Question 82: Create host-based routing Ingress: - app1.example.com → service1:80 - app2.example.com → service2:80

Question 83: Add annotations to Ingress for: - URL rewriting - SSL redirect - Custom timeout values - Rate limiting

Question 84: Debug Ingress issues: - Ingress created but not getting IP address - Check ingress controller logs - Verify backend services exist - Check DNS resolution

---

Troubleshooting Scenarios

Question 85: A Deployment is not creating any pods. Debug the issue by checking: - Deployment status - ReplicaSet status - Events - Resource quotas

Question 86: Pods are running but the Service is not routing traffic. Debug: - Check Service endpoints - Verify selector labels match - Test pod connectivity directly - Check NetworkPolicies

Question 87: A pod is running but can't access other services. Debug: - DNS resolution - NetworkPolicies - Service configuration - CoreDNS logs

Question 88: Application is slow or unresponsive: - Check resource usage with kubectl top - Review resource limits - Check application logs - Inspect readiness/liveness probes

Question 89: Persistent storage issues: - PVC stuck in Pending - Check PV availability - Verify StorageClass - Check access modes compatibility

Question 90: Container keeps crashing: - View logs of crashed container - Check resource limits - Verify image pull - Review startup probes - Check volume mounts

---

Command Reference

Essential kubectl Commands for CKAD

```
# Pod management
kubectl run nginx --image=nginx --dry-run=client -o yaml
kubectl get pods -o wide
kubectl describe pod <pod-name>
kubectl logs <pod-name> -c <container-name>
kubectl exec -it <pod-name> -- /bin/sh

# Deployments
kubectl create deployment nginx --image=nginx --replicas=3
kubectl scale deployment nginx --replicas=5
kubectl set image deployment/nginx nginx=nginx:1.21
kubectl rollout status deployment/nginx
kubectl rollout undo deployment/nginx

# Services
kubectl expose deployment nginx --port=80 --target-port=8080
kubectl get svc
kubectl get endpoints

# ConfigMaps and Secrets
kubectl create configmap app-config --from-literal=key=value
kubectl create secret generic app-secret --from-literal=password=secret

# Debugging
kubectl describe pod <pod-name>
kubectl logs <pod-name> --previous
kubectl get events --sort-by='.lastTimestamp'
kubectl top pods
kubectl auth can-i create pods

# Labels and Selectors
kubectl get pods -l app=nginx
kubectl label pod <pod-name> env=prod
kubectl annotate pod <pod-name> description="web server"

# Namespaces
kubectl get pods -n <namespace>
kubectl create namespace dev
kubectl config set-context --current --namespace=dev

# Output formats
kubectl get pods -o yaml
kubectl get pods -o json
kubectl get pods -o wide
kubectl get pods -o jsonpath='{.items[*].metadata.name}'
```

Exam Tips

1. **Time Management**: You have approximately 6-8 minutes per question
2. **Use kubectl Docs**: You can access kubernetes.io documentation during exam
3. **Practice Imperative Commands**: Faster than writing YAML from scratch
4. **Use Dry-run**: Generate YAML quickly with `--dry-run=client -o yaml`
5. **Aliases**: Set up helpful aliases like `alias k=kubectl`
6. **Vi/Vim Skills**: Practice YAML editing in vim
7. **Don't Overthink**: Move on if stuck, come back later
8. **Test Your Solutions**: Always verify your resources are working
9. **Read Questions Carefully**: Pay attention to namespaces, labels, names
10. **Know the Basics Cold**: Pods, Deployments, Services should be automatic

---

Answer Key Notes

This practice exam contains 90 questions covering all CKAD exam domains. For the actual exam: - Questions are performance-based (hands-on) - You work directly in terminal/browser - No multiple choice questions - Partial credit is possible - Focus on speed and accuracy

Study Strategy: 1. Try to answer each question without looking at references 2. Time yourself (6-8 minutes per question) 3. Verify your solutions actually work 4. Review concepts you struggled with 5. Practice with actual Kubernetes clusters

Good luck with your CKAD preparation!