# Kubernetes Questions and Practical Exercises

This document provides a structured set of Kubernetes-related questions and practical exercises, categorized by difficulty (Basic, Intermediate, Advanced) and topic, with detailed answers and solutions. The content is designed to match the depth and style of typical DSA question sets, covering theoretical concepts and hands-on tasks to prepare for interviews or certifications like CKA/CKAD.

# Basic Kubernetes Concepts

## Theory Questions

1. **What is Kubernetes?**
    - **Answer**: Kubernetes (K8s) is an open-source platform for automating the deployment, scaling, and management of containerized applications. It orchestrates containers across a cluster of nodes, providing features like load balancing, self-healing, and auto-scaling. It abstracts infrastructure complexities, enabling consistent application deployment across environments.
2. **What is a Kubernetes pod?**
    - **Answer**: A pod is the smallest deployable unit in Kubernetes, containing one or more containers that share storage, network resources, and a lifecycle. Pods run on nodes and are managed by controllers like Deployments or ReplicaSets.
3. **What is the difference between a container and a pod?**
    - **Answer**: A container is a single, isolated runtime environment for an application and its dependencies (e.g., Docker container). A pod is a Kubernetes abstraction that can contain one or multiple containers, sharing the same network namespace (e.g., localhost communication) and storage volumes.
4. **What is a Kubernetes node?**
    - **Answer**: A node is a worker machine (physical or virtual) in a Kubernetes cluster. It runs pods and includes components like the kubelet (manages containers), kube-proxy (handles networking), and a container runtime (e.g., containerd, CRI-O).
5. **What is the role of the Kubernetes control plane?**
    - **Answer**: The control plane manages the cluster's state and orchestrates workloads. Key components include:
        - **API Server**: Handles RESTful requests and updates cluster state.

        - **etcd**: Distributed key-value store for cluster data.
        - **Scheduler**: Assigns pods to nodes based on resource requirements.
        - **Controller Manager**: Runs controllers to maintain desired state (e.g., ReplicaSet controller).
        - **Cloud Controller Manager**: Integrates with cloud providers (optional).

6. **What is a Kubernetes Deployment?**

    - **Answer**: A Deployment is a controller that manages stateless applications by ensuring a specified number of pod replicas are running. It supports rolling updates, rollbacks, and scaling, defined via YAML or JSON manifests.

7. **What is a Kubernetes Service?**

    - **Answer**: A Service is an abstraction that defines a logical set of pods and a policy to access them (e.g., via load balancing). Types include ClusterIP (internal), NodePort (external), LoadBalancer (cloud-provided), and ExternalName (DNS alias).

8. **What is a Kubernetes namespace?**

    - **Answer**: A namespace is a virtual cluster within a physical Kubernetes cluster, used to organize resources and isolate environments (e.g., dev, prod). It helps manage resource quotas and access control.

9. **What is the difference between `kubectl apply` and `kubectl create`?**

    - **Answer**: `kubectl apply` updates resources declaratively, creating or modifying them based on a manifest file, preserving changes. `kubectl create` creates resources imperatively and fails if the resource already exists.

10. **What are the benefits of using Kubernetes?**

    - **Answer**: Kubernetes provides:
        - Automated scaling and load balancing.
        - Self-healing (restarting failed pods).
        - Consistent deployments across environments.
        - Resource optimization through scheduling.
        - Support for multi-cloud and hybrid environments.
        - Extensibility via custom resources and operators.

# Practical Exercises

1. **Create a pod with a single container**

    - **Task**: Create a pod running an `nginx:latest` container using a YAML file and verify it's running.
    - **Solution**:

```
# nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

```
kubectl apply -f nginx-pod.yaml
kubectl get pods
```

- **Explanation**: The YAML defines a pod with one Nginx container exposing port 80. Apply it with `kubectl apply` and check status with `kubectl get pods`. Expected output shows the pod in `Running` state.

2. **Run a simple Deployment**

- **Task**: Create a Deployment with 3 replicas of a `python:3.9` container running a simple HTTP server.
- **Solution**:

```
# python-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: python-app
  template:
    metadata:
      labels:
        app: python-app
    spec:
      containers:
      - name: python
        image: python:3.9
        command: ["python", "-m", "http.server", "8000"]
        ports:
        - containerPort: 8000
```

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

- **Explanation**: The Deployment ensures 3 pod replicas running a Python HTTP server. Apply the manifest and verify with `kubectl get deployments` and `kubectl get pods`.

3. **Expose a Deployment via a Service**

- **Task**: Expose the above Python Deployment as a ClusterIP Service on port 8000.
- **Solution**:

```
# python-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: python-service
  namespace: default
spec:
  selector:
    app: python-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
  type: ClusterIP
```

```
kubectl apply -f python-service.yaml
kubectl get services
```

- **Explanation**: The Service maps port 80 to the pods' port 8000, allowing internal cluster access. Verify with `kubectl get services`.

# Intermediate Kubernetes Concepts

## Theory Questions

1. **What is a ReplicaSet in Kubernetes?**
   - **Answer**: A ReplicaSet is a controller that ensures a specified number of pod replicas are running at any time. It is typically managed by a Deployment, which handles updates and rollbacks, while the ReplicaSet maintains the desired pod count.
2. **What is a ConfigMap, and how is it used?**
   - **Answer**: A ConfigMap stores configuration data as key-value pairs, decoupling configuration from application code. It can be mounted as environment variables, command-line arguments, or files in a pod's filesystem, enabling dynamic configuration updates.
3. **What is a Kubernetes Secret?**
   - **Answer**: A Secret stores sensitive data (e.g., passwords, API keys) in base64-encoded format. It is used similarly to ConfigMaps but with stricter access controls. Secrets can be mounted as volumes or environment variables in pods.
4. **What is a Kubernetes Ingress?**

- **Answer**: An Ingress is an API object that manages external access to Services, typically via HTTP/HTTPS. It provides load balancing, SSL termination, and path-based routing, requiring an Ingress controller (e.g., NGINX, Traefik) to function.

5. **What is the role of kube-proxy?**

   - **Answer**: kube-proxy runs on each node and manages network rules to enable communication between pods, Services, and external clients. It supports modes like iptables, IPVS, or userspace for load balancing and traffic routing.

6. **What is a PersistentVolume (PV) and PersistentVolumeClaim (PVC)?**

   - **Answer**: A PersistentVolume (PV) is a cluster-wide storage resource provisioned by an administrator or dynamically by a StorageClass. A PersistentVolumeClaim (PVC) is a request for storage by a user, binding to a PV to provide storage to pods.

7. **What is the time complexity of pod scheduling in Kubernetes?**

   - **Answer**: Pod scheduling is complex and depends on the scheduler's algorithm. On average, it's O(n) where n is the number of nodes, as the scheduler evaluates each node's suitability (resource availability, constraints). Optimizations like caching reduce practical complexity.

8. **What are Kubernetes labels and selectors?**

   - **Answer**: Labels are key-value pairs attached to objects (e.g., pods) for identification and grouping. Selectors query objects based on labels, used by controllers (e.g., Deployments) and Services to target specific pods.

9. **What is a Kubernetes StatefulSet?**

   - **Answer**: A StatefulSet is a controller for managing stateful applications, ensuring pods have stable identities and persistent storage. It maintains pod order and unique network identifiers, suitable for databases like MySQL or MongoDB.

10. **What is the difference between a DaemonSet and a Deployment?**

    - **Answer**: A DaemonSet ensures one pod runs on every node in the cluster (or a subset, based on node selectors), ideal for monitoring or logging agents. A Deployment manages stateless pods with a specified replica count, not tied to specific nodes.

# Practical Exercises

1. **Create a ConfigMap and mount it in a pod**

   - **Task**: Create a ConfigMap with key-value pairs and mount it as environment variables in an `alpine` pod.
   - **Solution**:

```
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: default
data:
  APP_ENV: production
  LOG_LEVEL: debug
---
apiVersion: v1
kind: Pod
metadata:
  name: config-pod
  namespace: default
spec:
  containers:
  - name: alpine
    image: alpine:latest
    command: ["sh", "-c", "env && sleep 3600"]
    env:
    - name: APP_ENV
      valueFrom:
        configMapKeyRef:
          name: app-config
          key: APP_ENV
    - name: LOG_LEVEL
      valueFrom:
        configMapKeyRef:
          name: app-config
          key: LOG_LEVEL
```

```
kubectl apply -f configmap.yaml
kubectl logs config-pod
```

- **Explanation**: The ConfigMap stores configuration data, mounted as environment variables in the pod. The pod runs `env` to display variables and sleeps. Check logs to verify `APP_ENV=production` and `LOG_LEVEL=debug`.

2. **Set up a PersistentVolume and PersistentVolumeClaim**

- **Task**: Create a PV and PVC for 1Gi of storage, and use it in a pod running `nginx`.
- **Solution**:

```yaml
# pv-pvc.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pv-pod
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:latest
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: storage
  volumes:
  - name: storage
    persistentVolumeClaim:
      claimName: my-pvc
```

```
kubectl apply -f pv-pvc.yaml
kubectl get pv,pvc,pod
```

- **Explanation**: The PV uses a `hostPath` for simplicity (in production, use cloud storage). The PVC requests 1Gi, and the pod mounts it to Nginx's HTML directory. Verify with `kubectl get`.

3. **Create an Ingress for external access**

- **Task**: Set up an Ingress to route traffic to the Python Service created earlier, assuming an NGINX Ingress controller is installed.
- **Solution**:

```
# python-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: python-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /python
        pathType: Prefix
        backend:
          service:
            name: python-service
            port:
              number: 80
```

```
kubectl apply -f python-ingress.yaml
kubectl get ingress
```

- **Explanation**: The Ingress routes traffic from `/python` to the `python-service`. Requires an NGINX Ingress controller. Access via `http://<ingress-host>/python`.

# Advanced Kubernetes Concepts

## Theory Questions

1. **What is a Kubernetes Operator?**
    - **Answer**: An Operator is a custom controller that extends Kubernetes functionality to manage complex, stateful applications (e.g., databases). It uses Custom Resource Definitions (CRDs) and custom logic to automate tasks like backups, scaling, or upgrades.
2. **What is the role of the Kubernetes scheduler?**
    - **Answer**: The scheduler assigns pods to nodes based on resource requirements, constraints (e.g., node selectors, taints/tolerations), and policies (e.g., affinity/anti-affinity). It optimizes resource utilization and ensures workload distribution.
3. **What are taints and tolerations in Kubernetes?**
    - **Answer**: Taints mark nodes to repel pods unless those pods have matching tolerations. They control pod placement, e.g., reserving nodes for specific workloads. Example: `kubectl taint nodes node1 key=value:NoSchedule`.
4. **What is a Kubernetes Horizontal Pod Autoscaler (HPA)?**
    - **Answer**: The HPA automatically scales the number of pod replicas in a Deployment or ReplicaSet based on metrics like CPU/memory usage or custom metrics. It requires a metrics server and is defined via a YAML manifest.
5. **How does Kubernetes handle container orchestration compared to Docker Swarm?**
    - **Answer**: Kubernetes offers advanced orchestration with features like auto-scaling, self-healing, rolling updates, and complex networking (e.g., Ingress, CNI plugins). Docker Swarm is simpler, with native Docker integration but fewer features for large-scale or stateful apps.
6. **What is the time complexity of Service load balancing in Kubernetes?**
    - **Answer**: Service load balancing (via kube-proxy) is typically O(1) for simple lookups, as it uses iptables or IPVS rules. However, updating rules for large clusters can be O(n) where n is the number of endpoints, depending on the mode.
7. **What are Kubernetes RBAC policies?**
    - **Answer**: Role-Based Access Control (RBAC) policies define permissions for users or service accounts to access Kubernetes resources. Roles/ClusterRoles specify allowed actions, and RoleBindings/ClusterRoleBindings assign them to subjects.
8. **What is a Custom Resource Definition (CRD)?**
    - **Answer**: A CRD extends Kubernetes by defining custom resources, allowing users to create new object types with custom behavior. Operators use CRDs to manage application-specific logic.
9. **What are the security best practices for Kubernetes?**
    - **Answer**:

- Enable RBAC and restrict permissions.
- Use network policies to control pod communication.
- Avoid running pods as root; use security contexts.
- Enable PodSecurityPolicies or PodSecurityAdmission.
- Secure the API server with TLS and authentication.
- Regularly update Kubernetes and scan images for vulnerabilities.

10. **What are the disadvantages of Kubernetes?**

    - **Answer**:
        - Steep learning curve for beginners.
        - High resource overhead for small applications.
        - Complex configuration for networking and storage.
        - Requires additional tools (e.g., monitoring, logging) for production.
        - Managing upgrades and cluster maintenance can be challenging.

# Practical Exercises

1. **Implement a Horizontal Pod Autoscaler**

    - **Task**: Create an HPA to scale the Python Deployment based on 70% CPU utilization, with a minimum of 2 and a maximum of 5 replicas.
    - **Solution**:

```
# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: python-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: python-deployment
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

```
kubectl apply -f hpa.yaml
kubectl get hpa
```

- **Explanation**: The HPA scales the `python-deployment` based on CPU usage. Requires a metrics server. Monitor scaling with `kubectl get hpa`.

2. **Set up a NetworkPolicy to restrict pod communication**

- **Task**: Create a NetworkPolicy to allow only ingress traffic to the Python pods from pods with the label `app=frontend`.
- **Solution**:

```
# network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: python-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: python-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8000
```

```
kubectl apply -f network-policy.yaml
kubectl get networkpolicy
```

- **Explanation**: The policy restricts traffic to `python-app` pods, allowing only TCP port 8000 from `app=frontend` pods. Requires a CNI plugin supporting network policies (e.g., Calico).

3. **Implement a StatefulSet for a MongoDB instance**

- **Task**: Create a StatefulSet for a MongoDB instance with persistent storage and a headless Service.
- **Solution**:

```
# mongo-statefulset.yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
  namespace: default
spec:
  clusterIP: None
  selector:
    app: mongo
  ports:
  - port: 27017
    targetPort: 27017
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongo
  namespace: default
spec:
  serviceName: mongo-service
  replicas: 1
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
      - name: mongo
        image: mongo:latest
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: mongo-storage
          mountPath: /data/db
  volumeClaimTemplates:
  - metadata:
      name: mongo-storage
    spec:
      accessModes: ["ReadWriteOnce"]
```

```
      resources:
        requests:
          storage: 1Gi
```

```
kubectl apply -f mongo-statefulset.yaml
kubectl get statefulset,svc,pvc
```

- **Explanation**: The headless Service (`clusterIP: None`) provides stable DNS for the MongoDB pod. The StatefulSet ensures a single replica with persistent storage via a PVC. Verify with `kubectl get`.

4. **Detect and fix a failing pod**

- **Task**: Create a pod with an invalid image reference, diagnose the issue, and fix it.
- **Solution**:

```
# bad-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: bad-pod
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:invalid
```

```
kubectl apply -f bad-pod.yaml
kubectl describe pod bad-pod
kubectl delete pod bad-pod
# Fix by updating to valid image
# Edit bad-pod.yaml to use image: nginx:latest
kubectl apply -f bad-pod.yaml
kubectl get pods
```

- **Explanation**: The pod fails with an `ImagePullBackOff` error due to the invalid image. Use `kubectl describe pod` to diagnose, delete the pod, update the YAML to `nginx:latest`, and reapply.

5. **Implement a liveness probe**

- **Task**: Add a liveness probe to the Python Deployment to check if the HTTP server is running.
- **Solution**:

```
# python-deployment-liveness.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: python-app
  template:
    metadata:
      labels:
        app: python-app
    spec:
      containers:
      - name: python
        image: python:3.9
        command: ["python", "-m", "http.server", "8000"]
        ports:
        - containerPort: 8000
        livenessProbe:
          httpGet:
            path: /
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 10
```

```
kubectl apply -f python-deployment-liveness.yaml
kubectl get pods
```

- **Explanation**: The liveness probe checks the HTTP server's root path every 10 seconds after a 5-second delay. If the probe fails, Kubernetes restarts the pod. Verify pod status with `kubectl get pods`.

# Guidelines

- **Practice Platforms**: Use Minikube, Kind, or cloud-based clusters (e.g., GKE, EKS, AKS) for hands-on practice. Platforms like KodeKloud Labs or KillerCoda offer Kubernetes sandboxes.
- **Resources**: Refer to Kubernetes official documentation, CNCF's CKA/CKAD curriculum, and YouTube tutorials (e.g., TechWorld with Nana) for practical solutions.

- **Best Practices**: Focus on understanding Kubernetes objects and workflows rather than memorizing YAML syntax. Use `kubectl explain` for resource details and practice debugging with `kubectl describe` and `kubectl logs`.
- **Certifications**: Prepare for CKA/CKAD exams with practice tests from KillerCoda, Udemy, or Whizlabs. Focus on tasks like pod creation, troubleshooting, and scaling.
- **Optimization**: Experiment with resource limits, auto-scaling, and network policies to optimize cluster performance. Regularly update Kubernetes and scan images for vulnerabilities.
- **Security**: Follow best practices like enabling RBAC, using network policies, and running pods with minimal privileges.