# Comprehensive Kubernetes Interview Questions and Answers

**## 1. What is Kubernetes? Why is it used?**

\*\*Kubernetes\*\* is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF).

\*\*Why it's used:\*\*

- \*\*Container orchestration\*\*: Manages the lifecycle of containers across multiple hosts

- \*\*High availability\*\*: Ensures applications are running with minimal downtime

- \*\*Scalability\*\*: Easily scales applications up or down based on demand

- \*\*Resource optimization\*\*: Efficiently utilizes hardware resources

- \*\*Self-healing\*\*: Automatically restarts failed containers and replaces unhealthy ones

- \*\*Service discovery and load balancing\*\*: Distributes network traffic to ensure stable deployments

- \*\*Automated rollouts and rollbacks\*\*: Manages application updates with minimal downtime

- \*\*Portability\*\*: Runs on various environments (on-premises, cloud, hybrid)

## 2. **What are the main components of the Kubernetes architecture?**

Kubernetes has two main categories of components: \*\*Control Plane (Master) components\*\* and \*\*Node components\*\*.

### Control Plane Components:

1. \*\*kube-apiserver\*\*: Front-end for the Kubernetes control plane, exposes the Kubernetes API

2. \*\*etcd\*\*: Consistent and highly-available key-value store for all cluster data

3. \*\*kube-scheduler\*\*: Watches for newly created Pods and selects a node for them to run on

4. \*\*kube-controller-manager\*\*: Runs controller processes (Node Controller, Replication Controller, etc.)

5. \*\*cloud-controller-manager\*\*: Links the cluster to the cloud provider's API (optional)

### Node Components:

1. \*\*kubelet\*\*: Agent that runs on each node, ensures containers are running in a Pod

2. \*\*kube-proxy\*\*: Maintains network rules on nodes for network communication

3. \*\*Container runtime\*\*: Software responsible for running containers (Docker, containerd, etc.)

## 3. **Explain the difference between a Pod, Deployment and Service**.

- \*\*Pod\*\*: The smallest deployable unit in Kubernetes. A Pod represents a single instance of a running process in your cluster and can contain one or more containers that share storage and network resources.

- \*\*Deployment\*\*: A higher-level abstraction that manages Pods and ReplicaSets. It provides declarative updates for Pods and enables features like rolling updates, rollbacks, and scaling.

- \*\*Service\*\*: An abstraction that defines a logical set of Pods and a policy to access them. Services enable network access to a set of Pods, providing stable IP addresses and DNS names even as Pods come and go.

## **4. How does Kubernetes handle scaling?**

Kubernetes provides several scaling mechanisms:

1. \*\*Manual Scaling\*\*: Using `kubectl scale` command to increase/decrease the number of replicas

```bash

kubectl scale deployment/my-app --replicas=5

```

2. \*\*Horizontal Pod Autoscaler (HPA)\*\*: Automatically scales the number of Pods based on CPU utilization or custom metrics

```yaml

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 2

maxReplicas: 10

targetCPUUtilizationPercentage: 80

```

3. \*\*Vertical Pod Autoscaler (VPA)\*\*: Automatically adjusts the CPU and memory requests/limits of Pods

4. \*\*Cluster Autoscaler\*\*: Automatically adjusts the size of the Kubernetes cluster (adds/removes nodes) based on resource needs

## 5. **What's the role of kube-apiserver, kube-scheduler and kube-controller-manager?**

- \*\*kube-apiserver\*\*:

- Primary management component that exposes the Kubernetes API

- Validates and configures data for API objects (Pods, Services, etc.)

- Acts as the front-end to the cluster's shared state (stored in etcd)

- All other components interact with the cluster through the API server

- \*\*kube-scheduler\*\*:

- Watches for newly created Pods that have no node assigned

- Selects an optimal node for a Pod to run on based on:

- Resource requirements

- Hardware/software/policy constraints

- Affinity and anti-affinity specifications

- Data locality

- Deadlines

- \*\*kube-controller-manager\*\*:

- Runs controller processes that regulate the state of the cluster

- Includes several controllers:

- Node Controller: Monitors node status

- Replication Controller: Maintains correct number of Pods

- Endpoints Controller: Populates Endpoints objects

- Service Account & Token Controllers: Create default accounts and API access tokens

## **6. What's a Namespace in Kubernetes and why use it?**

A \*\*Namespace\*\* in Kubernetes is a virtual cluster within a physical cluster. It provides a scope for names and is a way to divide cluster resources between multiple users/teams/projects.

\*\*Why use Namespaces:\*\*

- \*\*Resource isolation\*\*: Separate environments (dev, staging, prod) within the same cluster

- \*\*Access control\*\*: Apply RBAC policies to specific namespaces

- \*\*Resource quotas\*\*: Limit resource usage per namespace

- \*\*Organization\*\*: Group related resources together

- \*\*Name collision avoidance\*\*: The same resource name can be used in different namespaces

Common namespaces:

- `default`: Where resources are created if no namespace is specified

- `kube-system`: For Kubernetes system components

- `kube-public`: For resources that should be readable by all users

## **7. How does a Kubernetes Service enable communication between Pods?**

A Kubernetes \*\*Service\*\* enables communication between Pods and other network endpoints through:

1. \*\*Stable IP Address\*\*: The Service gets a stable cluster-internal IP address (ClusterIP) that doesn't change

2. \*\*DNS Name\*\*: The Service gets a DNS name in the format `<service-name>.<namespace>.svc.cluster.local`

3. \*\*Load Balancing\*\*: Distributes traffic to all healthy Pods matching the Service's selector

4. \*\*Endpoint Tracking\*\*: Automatically maintains an Endpoints object that lists all Pod IPs matching the selector

Example Service definition:

```yaml

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 9376

```

## **8. What's the difference between ClusterIP, NodePort and LoadBalancer services?**

| Type | Description | Use Case | Accessible From |

|---------------|-------------|----------|-----------------|

| \*\*ClusterIP\*\* | Default type. Exposes the Service on a cluster-internal IP. | Internal communication between Pods | Only within the cluster |

| \*\*NodePort\*\* | Exposes the Service on each Node's IP at a static port (NodePort). Also creates a ClusterIP. | Development/testing or when you need to expose a Service directly from a node | Outside the cluster via `<NodeIP>:<NodePort>` |

| \*\*LoadBalancer\*\* | Creates an external load balancer in cloud providers. Also creates NodePort and ClusterIP. | Production workloads that need external access | Outside the cluster via the load balancer's IP |

## **9. Explain the concept of ConfigMaps and Secrets.**

\*\*ConfigMaps\*\*:

- Allow you to decouple configuration artifacts from container images

- Store non-confidential data in key-value pairs

- Can be consumed as:

- Environment variables in a container

- Command-line arguments

- Configuration files in a volume

- Example:

```yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

LOG\_LEVEL: debug

CONFIG\_FILE: |

setting1=value1

setting2=value2

```

\*\*Secrets\*\*:

- Similar to ConfigMaps but for sensitive data (passwords, tokens, keys)

- Stored as base64-encoded (not encrypted by default)

- Best practices:

- Enable encryption at rest for Secrets

- Use RBAC to restrict access

- Consider external secret management solutions (Vault, AWS Secrets Manager)

- Example:

```yaml

apiVersion: v1

kind: Secret

metadata:

name: db-secret

type: Opaque

data:

username: YWRtaW4=

password: MWYyZDFlMmU2N2Rm

```

## **10. How do you check the logs of a running Pod?**

Basic command:

```bash

kubectl logs <pod-name>

```

Options:

- View logs from a specific container in a multi-container Pod:

```bash

kubectl logs <pod-name> -c <container-name>

```

- Stream logs in real-time:

```bash

kubectl logs -f <pod-name>

```

- View logs from a previous instance of a crashed Pod:

```bash

kubectl logs --previous <pod-name>

```

- View logs from all Pods matching a label selector:

```bash

kubectl logs -l app=my-app

```

- View logs from a specific time period:

```bash

kubectl logs <pod-name> --since=1h

```

## **11. What's the difference between a Stateful Set and a Deployment?**

| Feature | Deployment | StatefulSet |

|--------------------|------------|-------------|

| \*\*Pod Identity\*\* | No stable identity (Pods are interchangeable) | Stable, persistent identity (ordinal index) |

| \*\*Pod Naming\*\* | Random hash suffix | Ordered, predictable names (e.g., web-0, web-1) |

| \*\*Storage\*\* | Typically uses non-persistent storage | Each Pod gets its own PersistentVolume |

| \*\*Scaling\*\* | Any order | Ordered creation/deletion (0, 1, 2,...) |

| \*\*Updates\*\* | Rolling updates supported | Rolling updates supported |

| \*\*DNS\*\* | All Pods share the same DNS name | Each Pod gets its own DNS name |

| \*\*Use Case\*\* | Stateless applications | Stateful applications (databases, etc.) |

## **12. How do you update a running deployment in Kubernetes?**

Common update methods:

1. \*\*Update the Deployment's container image\*\*:

```bash

kubectl set image deployment/my-app my-app-container=my-app:v2

```

2. \*\*Edit the Deployment directly\*\*:

```bash

kubectl edit deployment/my-app

```

3. \*\*Apply an updated manifest file\*\*:

```bash

kubectl apply -f deployment.yaml

```

Kubernetes performs a \*\*rolling update\*\* by default:

- Creates new Pods with the updated configuration

- Gradually replaces old Pods with new ones

- Ensures a specified number of Pods are available during the update

You can control the update strategy:

```yaml

spec:

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 25%

maxSurge: 25%

```

**## 13. What's the purpose of a PersistentVolume and PersistentVolumeClaim?**

\*\*PersistentVolume (PV)\*\*:

- A cluster resource that represents physical storage in the cluster

- Can be provisioned:

- Statically: by an administrator

- Dynamically: using StorageClasses

- Has a lifecycle independent of any individual Pod

- Example:

```yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-volume

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

storageClassName: slow

hostPath:

path: /mnt/data

```

\*\*PersistentVolumeClaim (PVC)\*\*:

- A request for storage by a user

- Binds to a PV that matches the request's requirements

- Example:

```yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pv-claim

spec:

storageClassName: slow

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 3Gi

```

## **14. Explain the role of Ingress in Kubernetes**.

\*\*Ingress\*\*:

- An API object that manages external access to services in a cluster

- Provides:

- HTTP/HTTPS routing

- Host-based or path-based routing

- SSL/TLS termination

- Load balancing

- Requires an \*\*Ingress Controller\*\* to implement the rules (e.g., Nginx, Traefik, AWS ALB)

- Example:

```yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: example-ingress

spec:

rules:

- host: myapp.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: web-service

port:

number: 80

```

#**# 15. How do you roll back a deployment in Kubernetes?**

1. \*\*Check the rollout history\*\*:

```bash

kubectl rollout history deployment/my-app

```

2. \*\*View details of a specific revision\*\*:

```bash

kubectl rollout history deployment/my-app --revision=2

```

3. \*\*Roll back to the previous version\*\*:

```bash

kubectl rollout undo deployment/my-app

```

4. \*\*Roll back to a specific revision\*\*:

```bash

kubectl rollout undo deployment/my-app --to-revision=2

```

5. \*\*Watch the rollback progress\*\*:

```bash

kubectl rollout status deployment/my-app

```

## **16. What's the role of the kubectl command-line tool?**

\*\*kubectl\*\* is the primary CLI tool for interacting with Kubernetes clusters. Its roles include:

- \*\*Cluster management\*\*: View cluster info, manage nodes

- \*\*Resource management\*\*: Create, view, update, delete Kubernetes resources

- \*\*Debugging\*\*: View logs, exec into containers, port forwarding

- \*\*Configuration\*\*: Apply manifests, manage namespaces, contexts

- \*\*Monitoring\*\*: Get resource status, view events

- \*\*Scaling\*\*: Scale deployments up/down

- \*\*Updates\*\*: Perform rolling updates and rollbacks

Common commands:

```bash

kubectl get pods

kubectl apply -f config.yaml

kubectl describe pod my-pod

kubectl logs my-pod

kubectl exec -it my-pod -- bash

kubectl scale deployment/my-app --replicas=5

```

**## 17. What is a Daemon Set and when would you use it?**

A \*\*DaemonSet\*\* ensures that all (or some) Nodes run a copy of a Pod. As Nodes are added to the cluster, Pods are added to them. As Nodes are removed, those Pods are garbage collected.

\*\*Use cases\*\*:

- Cluster storage daemons (glusterd, ceph)

- Logs collection daemons (fluentd, logstash)

- Node monitoring daemons (Prometheus Node Exporter)

- Networking plugins (kube-proxy, Calico)

- Any system-level service that needs to run on every node

Example:

```yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd

spec:

selector:

matchLabels:

name: fluentd

template:

metadata:

labels:

name: fluentd

spec:

containers:

- name: fluentd

image: fluentd

```

## **18. Explain the concept of Taints and Tolerations. Explain in detail about the effects in taint.**

\*\*Taints and Tolerations\*\* allow you to repel Pods from specific Nodes unless the Pod has a matching toleration.

\*\*Taint\*\* (applied to Nodes):

```bash

kubectl taint nodes node1 key=value:effect

```

\*\*Toleration\*\* (applied to Pods):

```yaml

tolerations:

- key: "key"

operator: "Equal"

value: "value"

effect: "effect"

```

\*\*Taint Effects\*\*:

1. \*\*NoSchedule\*\*:

- Pods that don't tolerate the taint won't be scheduled on the node

- Existing Pods on the node remain unaffected

2. \*\*PreferNoSchedule\*\*:

- Soft version of NoSchedule

- The scheduler tries to avoid placing Pods that don't tolerate the taint

- But may still place them if no better options exist

3. \*\*NoExecute\*\*:

- Affects both new Pods and existing Pods

- Pods without matching toleration are evicted from the node

- Can specify a tolerationSeconds to delay eviction

## **19. How do you secure a Kubernetes cluster?**

\*\*Cluster Security Best Practices\*\*:

1. \*\*RBAC\*\*: Implement Role-Based Access Control

2. \*\*Network Policies\*\*: Restrict Pod-to-Pod communication

3. \*\*Pod Security Policies/Standards\*\*: Define what Pods can do

4. \*\*Secret Management\*\*: Use proper secret handling (encryption at rest)

5. \*\*Update Regularly\*\*: Keep Kubernetes and components patched

6. \*\*etcd Security\*\*: Enable TLS for etcd communication

7. \*\*API Server Security\*\*:

- Use strong authentication (certificates, tokens)

- Enable audit logging

- Restrict access to the API server

8. \*\*Node Security\*\*:

- Minimize host OS footprint

- Use host-level firewalls

- Restrict container capabilities

9. \*\*Image Security\*\*:

- Scan container images for vulnerabilities

- Use trusted registries

10. \*\*Limit Dashboard Access\*\*: Secure the Kubernetes dashboard

**## 20. What's a Helm chart and how does it help with Kubernetes deployments?**

\*\*Helm\*\* is the package manager for Kubernetes. A \*\*Helm chart\*\* is a collection of files that describe a related set of Kubernetes resources.

\*\*Components of a Helm Chart\*\*:

- `Chart.yaml`: Metadata about the chart

- `values.yaml`: Default configuration values

- `templates/`: Kubernetes manifest templates

- `charts/`: Sub-charts that this chart depends on

\*\*Benefits\*\*:

- \*\*Packaging\*\*: Bundle multiple Kubernetes resources together

- \*\*Configuration\*\*: Parameterize deployments with values

- \*\*Versioning\*\*: Manage different versions of deployments

- \*\*Sharing\*\*: Distribute applications via chart repositories

- \*\*Lifecycle Management\*\*: Install, upgrade, rollback releases

Example commands:

```bash

helm install my-release ./mychart

helm upgrade my-release ./mychart

helm rollback my-release 1

helm list

```

## **21. Explain about Resources (Requests and limits) and Taint and Tolerations**

### Resources (Requests and Limits):

Kubernetes allows you to specify resource requirements for containers:

- \*\*requests\*\*: The minimum amount of CPU/memory the container needs

- Used by the scheduler to place Pods on Nodes

- Guaranteed to the container

- \*\*limits\*\*: The maximum amount of CPU/memory the container can use

- If exceeded, the container might be throttled (CPU) or terminated (memory)

Example:

```yaml

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

```

### Taint and Tolerations (covered in detail in question 18)

## **22. What are different deployment strategies in Kubernetes?**

1. \*\*Rolling Update (default)\*\*:

- Gradually replaces old Pods with new ones

- Ensures continuous availability

- Can control speed with maxUnavailable and maxSurge

2. \*\*Recreate\*\*:

- Terminates all old Pods before creating new ones

- Results in downtime

- Useful when the application doesn't support running multiple versions simultaneously

3. \*\*Blue-Green\*\*:

- Deploy new version alongside old version

- Switch traffic all at once

- Requires additional resources during transition

4. \*\*Canary\*\*:

- Deploy new version to a small subset of users

- Gradually increase traffic to new version

- Allows for testing with real traffic

5. \*\*A/B Testing\*\*:

- Similar to Canary but based on specific conditions (headers, cookies)

- Requires service mesh or advanced ingress controller

## 22. Can you write a pod definition file and Deployment file?

\*\*Pod Definition File\*\*:

```yaml

apiVersion: v1

kind: Pod

metadata:

name: my-pod

labels:

app: my-app

tier: frontend

spec:

containers:

- name: my-container

image: nginx:1.19

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

```

\*\*Deployment File\*\*:

```yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deployment

labels:

app: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 1

maxSurge: 1

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: nginx

image: nginx:1.19

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

```

**## 23. How can you scale up and scale down the pods?**

\*\*Manual Scaling\*\*:

```bash

# Scale a Deployment

kubectl scale deployment/my-app --replicas=5

# Scale a ReplicaSet

kubectl scale rs/my-rs --replicas=3

# Scale a StatefulSet

kubectl scale sts/my-statefulset --replicas=7

```

\*\*Auto Scaling (HPA)\*\*:

```bash

# Create an HPA that maintains 50% CPU utilization

kubectl autoscale deployment/my-app --cpu-percent=50 --min=2 --max=10

```

\*\*Editing the Deployment\*\*:

```bash

kubectl edit deployment/my-app

# Then change spec.replicas

```

## **24. When user enters kubectl get pods what happens?**

1. \*\*Authentication\*\*: kubectl checks kubeconfig for credentials and authenticates with the API server

2. \*\*API Request\*\*: kubectl sends a GET request to the Kubernetes API server at `/api/v1/namespaces/default/pods` (or specified namespace)

3. \*\*Authorization\*\*: API server checks RBAC rules to verify the user has permission to list Pods

4. \*\*etcd Query\*\*: API server queries etcd for Pod resources

5. \*\*Filtering\*\*: Results are filtered based on:

- Namespace (if specified)

- Label selectors (if specified with `-l`)

- Field selectors (if specified with `--field-selector`)

6. \*\*Response\*\*: API server returns the filtered Pod list to kubectl

7. \*\*Display\*\*: kubectl formats and displays the results to the user

## **25. Explain about any 3 controllers in Kubernetes control manager**

1. \*\*ReplicaSet Controller\*\*:

- Ensures the desired number of Pod replicas are running at all times

- Creates/deletes Pods to match the replica count

- Uses label selectors to identify Pods it should manage

- Works in conjunction with Deployments for rolling updates

2. \*\*Deployment Controller\*\*:

- Manages the lifecycle of Deployments

- Handles rolling updates and rollbacks

- Creates and manages ReplicaSets to achieve the desired state

- Provides declarative updates for Pods

3. \*\*Node Controller\*\*:

- Monitors the status of nodes in the cluster

- Handles node failures by:

- Evicting Pods from unreachable nodes

- Marking nodes as NotReady

- Deleting nodes from the cluster if they remain unreachable

- Synchronizes node information with cloud providers (if configured)