Top 44 Kubernetes Interview Questions and Answers in 2025

Ace your Kubernetes interview with this guide covering concepts, essential questions and tips to boost your confidence and expertise.

Contents

Feb 12, 2025 · 15 min read

Patrick Brus

Cloud Engineer specializing in MLOps, DataOps, and Kubernetes, with a background in ML and computer vision.

TOPICS

Kubernetes

MLOps

I still remember preparing for my first Kubernetes interview. While I had a solid understanding of container orchestration, I quickly realized that passing a Kubernetes interview required much more than just theoretical knowledge. It demanded hands-on experience, troubleshooting skills, and the ability to solve real-world challenges.

Now, after working extensively with Kubernetes and navigating multiple interviews, I've gained insights into what truly matters in these discussions.

In this guide, I'll share everything you need to prepare for your Kubernetes interview, including:

- Fundamental Kubernetes concepts you need to know.
- Basic, intermediate, and advanced interview questions.
- Scenario-based problem-solving questions.
- Tips for preparing effectively.

By the end of this article, you'll have a solid roadmap to prepare for your Kubernetes interviews and take your career to the next level!

What is Kubernetes?

Before getting into the interview questions, let's have a quick look at Kubernetes fundamentals. Feel free to skip this section if you are already familiar with these concepts.

Kubernetes (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. Google originally developed it and later donated it to the Cloud Native Computing Foundation (CNCF).

Kubernetes became the industry standard for managing microservices-based applications in cloud environments.

It brings the following features:

- **Automated container orchestration:** No more manual container management.

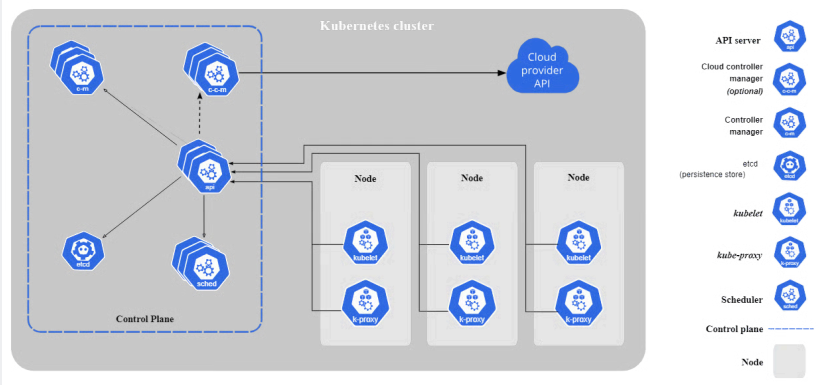
- **Self-healing capabilities:** Automatically restarts failed containers, replaces unresponsive nodes, and reschedules workloads dynamically.
- **Load balancing and service discovery:** Ensures traffic is efficiently distributed between Pods.
- **Declarative configuration management:** Everything is configured via YAML code.
- **Horizontal and vertical scaling:** Can automatically scale applications based on CPU usage, memory usage, or custom metrics.
- **Multi-cloud and hybrid-cloud support:** Works with AWS, Azure, GCP, on-premises, and hybrid environments.

But why is it essential in the first place? It simplifies the deployment and operation of microservices and containers by automating complex tasks like rolling updates, service discovery, and fault tolerance. Kubernetes dynamically schedules workloads across available computing resources and abstracts away these complex principles from the end user.

Core components of Kubernetes

Kubernetes consists of the following core components:

- **Control plane (master node components):**
  - kube-apiserver: Main API gateway for the cluster.
  - etcd: A distributed key-value store that holds the clusters' state and configuration.
  - kube-scheduler: Assigns Pods to nodes based on resource availability.
  - controller-manager: Manages controllers.
- **Worker node components:**
  - kubelet: Agent that runs on each worker node. It ensures that Pods are running as expected.
  - kube-proxy: Handles networking and routes traffic between services.



Core components of Kubernetes. Image by Kubernetes.io

Kubernetes architecture overview

Kubernetes follows a master-worker architecture. The control plane (master node) manages cluster operations while worker nodes run containerized applications. Pods, the smallest deployable unit in Kubernetes, run containers and are assigned to nodes.

Kubernetes ensures the desired state by continuously monitoring and adjusting workloads as required.

Still confused about how Kubernetes and Docker compare? Check out this in-depth **Kubernetes vs. Docker comparison** to understand their roles in containerized environments.

Basic Kubernetes Interview Questions

Let's now start with basic Kubernetes interview questions. These questions cover the foundational knowledge required to understand and work with Kubernetes.

What is a Pod in Kubernetes?

A Pod is the **smallest deployable unit** in Kubernetes. It represents one or more containers running in a shared environment. Containers inside a Pod share networking and storage resources.

Here's a simple Pod definition YAML file:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.21
    ports:
    - containerPort: 80
```


What is the purpose of kubectl?

Kubectl is the primary CLI tool for managing Kubernetes resources and interacting with the cluster. Here are a few common kubectl commands you should be familiar with:

```
kubectl get pods                # list all Pods
kubectl get services            # list all Services
kubectl logs <pod-name>         # view logs of a Pod
kubectl exec -it <pod-name> - /bin/sh # open a shell inside a Pod
```


What is a Deployment in Kubernetes?

A Deployment in Kubernetes is a higher-level abstraction that manages the lifecycle of Pods. It ensures that the desired number of replicas are up and running and provides features like rolling updates, rollbacks, and self-healing.

Here's how a simple Deployment definition YAML file looks like:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

POWERED BY datalab

What is a Kubernetes Service, and why is it needed?

A Service in Kubernetes exposes a group of Pods and allows communication between and to them.

Since Pods are ephemeral, their IPs can change, meaning the application talking to the Pods must also change the IP address. Services, therefore, provide a stable network endpoint with a fixed IP address.

A simple Service YAML definition:

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

POWERED BY datalab

The above Service forwards traffic to Pods that have the label app: my-app .

What service types are available at Kubernetes Services?

Kubernetes provides four main types of Services, each serving a different networking purpose:

- ClusterIP (default): Allows for internal communication of Pods. Only accessible from within the cluster.
- NodePort: This exposes the Service on a static port of each Node, making It accessible from outside the cluster.
- LoadBalancer: Uses a cloud provider's external load balancer. The Service is then accessible via a public IP.
- ExternalName: Maps a Kubernetes Service to an external hostname.

What is the role of ConfigMaps and secrets in Kubernetes?

ConfigMaps stores non-sensitive configuration data, while secrets stores sensitive data like API keys and passwords.

Using secrets allows you to avoid putting secret information into your application code. In contrast, ConfigMaps will enable you to make your applications more configurable, as these values can be easily edited, and you don't need to persist them in your application code.

Example ConfigMap YAML definition:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  database_url: "postgres://db.example.com"
```


Example secret YAML definition (with base64 encoded data):

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  password: cGFzc3dvcmQ=  # "password" encoded in Base64
```


What are Namespaces in Kubernetes?

A Namespace is a virtual cluster within a Kubernetes cluster. It helps to organize workloads in multi-tenant environments by isolating resources within a cluster.

Below, you can find a short code snippet that shows how to create a Namespace using kubectl and how to create and fetch Pods in that Namespace:

```
# create a namespace called “dev”
kubectl create namespace dev

# create a Pod in that namespace
kubectl run nginx --image=nginx --namespace=dev

# get Pods in that namespace
kubectl get pods --namespace=dev
```


What are Labels and Selectors in Kubernetes?

Labels are key/value pairs attached to objects (e.g. Pods). They help to organize Kubernetes objects. Selectors filter resources based on Labels.

Here is an example Pod that has the labels `environment: production` and `app: nginx`:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    environment: production
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx:1.21
    ports:
    - containerPort: 80
```

You could now use the following label selector to fetch all Pods that have the label environment: pod assigned:

```
kubectl get pods -l environment=production
```

POWERED BY datalab

What are Persistent Volumes (PVs) and Persistent Volume Claims (PVCs)?

Persistent Volumes (PVs) provide storage that persists beyond Pod lifecycles. The PV is a storage piece in the cluster that has been provisioned by a cluster administrator or dynamically provisioned using Storage Classes.

A Persistent Volume Claim (PVC) is a request for storage by a user.

Here's an example PV and PVC YAML definition:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data"
```

POWERED BY datalab

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

POWERED BY datalab

Intermediate Kubernetes Interview Questions

Now that you practiced the basics, we can move to intermediate-level questions. Understanding concepts such as networking, security, resource management, and troubleshooting is essential at this level.

What is Kubernetes networking, and how does it work?

Kubernetes networking allows for communication between Pods, Services, and external clients. It follows a flat networking structure, meaning that, by default, all Pods can communicate with each other.

Key networking concepts in Kubernetes include:

- Pod-to-pod communication: Each Pod gets a unique IP assigned and can communicate within the cluster.
- Service-to-pod communication: Services provide a stable network endpoint for a group of Pods, as Pods are ephemeral. Each pod gets a new IP assigned every time it

is created.

- Ingress controllers: Manage external HTTP/HTTPS traffic.
- Network policies: Define rules to restrict or allow communication between Pods.

What is Role-Based Access Control (RBAC) in Kubernetes?

RBAC is a security mechanism that restricts users and services based on their permissions. It consists of:

- Roles and ClusterRoles: Define the actions allowed on resources.
- RoleBindings and ClusterRoleBindings: Assign roles to users or service accounts.

The following YAML shows an example role that only allows read-only access to Pods:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
```


This pod-reader role can now be bound to a user using RoleBinding:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-reader-binding
subjects:
  - kind: User
    name: dummy
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```


How does Kubernetes autoscaling work?

Kubernetes provides three types of autoscaling to optimize resource usage:

1. Horizontal Pod Autoscaler (HPA): Adjusts the number of Pods based on CPU usage, memory usage, or custom metrics.
2. Vertical Pod Autoscaler (VPA): Adjusts the CPU and memory requests for individual Pods.
3. Cluster Autoscaler: Adjusts the number of worker nodes in the cluster based on resource needs.

You can create an HPA using kubectl:

```bash
kubectl autoscale deployment nginx --cpu-percent=50 --min=1 --max=10
```


The above command creates an HPA for a Deployment with the name nginx and tries to keep the average CPU utilization across all Pods at 50%, with a minimum number of replicas

at 1 and a maximum number of replicas at 10.

How do you debug Kubernetes Pods?

When Pods fail, Kubernetes provides multiple ways to debug it:

- Use `kubectl logs` to check container logs for errors.
- Use `kubectl describe pod` to inspect events and recent state changes.
- Use `kubectl exec` to open an interactive shell and investigate from inside the container.
- Use `kubectl get pods --field-selector=status.phase=Failed` to list all failing Pods.

```
# get logs of a specific Pod
kubectl get logs <pod-name>

# describe the Pod and check events
kubectl describe pod <pod-name>

# open an interactive shell inside the Pod
kubectl exec -it <pod-name> -- /bin/sh

# check all failing pods
kubectl get pods --field-selector=status.phase=Failed
```


These commands help identify misconfigurations, resource constraints, or application errors.

How do you perform rolling updates and rollbacks in Kubernetes?

Kubernetes Deployments support rolling updates to avoid downtime. You can perform a rolling update by editing a Deployment or explicitly setting its image to a new version using:

```
kubectl set image deployment/my-deployment nginx=nginx:1.21
```


You can then check the Deployment status:

```
kubectl rollout status deployment my-deployment
```


If you want to roll back to the previous version, you can run:

```
kubectl rollout undo deployment my-deployment
```


What is an Ingress in Kubernetes, and how does it work?

An Ingress is an API object that manages external HTTP/HTTPS access to Services inside a Kubernetes cluster. It allows routing requests based on hostname and paths, acting as a reverse proxy for multiple applications.

Example Ingress YAML definition:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
```

```
metadata:
  name: my-ingress
spec:
  rules:
    - host: my-app.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-service
                port:
                  number: 80
```

POWERED BY datalab

How does Kubernetes handle resource limits and requests?

Kubernetes allows you to set resource requests and limits for Pods to **ensure fair allocation** and avoid the overuse of cluster resources.

- Requests are the minimum amount of CPU and memory a Pod needs. They are permanently assigned to a Pod.
- Limits are the maximum a Pod can use. They are not assigned to the Pod, but if it requires more resources, it can grow until the limit is reached.

Example YAML Pod definition that sets resource requests and limits:

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-limited-pod
spec:
  containers:
    - name: my-container
      image: nginx
      resources:
        requests:
          memory: "256Mi"
          cpu: "250m"
        limits:
          memory: "512Mi"
          cpu: "500m"
```

POWERED BY datalab

What happens if a Pod resource needs to grow beyond the assigned limits?

If a Pod's memory consumption exceeds its assigned memory limit, Kubernetes **immediately kills** the container with an out of memory (OOM) error. The container restarts if a restart policy is defined.

Unlike memory, if a Pod exceeds its assigned CPU limit, it is **not killed**. Instead, Kubernetes throttles CPU usage, causing the application to slow down.

What are init containers, and when should you use them?

Init containers are special containers that **run before the primary containers start**. They help prepare the environment by checking dependencies, loading configuration files, or setting up data.

One example could be an init container that waits for a database to be up and running:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-init
spec:
  initContainers:
    - name: wait-for-db
      image: busybox
      command: ['sh', '-c', 'until nc -z db-service 5432; do sleep 2; done']
  containers:
    - name: my-app
      image: my-app-image
```


How does Kubernetes handle Pod disruptions and high availability?

Kubernetes ensures high availability through Pod Disruption Budgets (PDBs), anti-affinity rules, and self-healing mechanisms. Here's how these mechanisms work:

- **Pod Disruption Budget (PDB):** Ensures a minimum number of Pods remain available during voluntary disruptions (e.g., cluster updates where nodes need to be scaled down).
- **Pod affinity and anti-affinity:** Controls for which Pods can be scheduled together or separately.
- **Node selectors and Taints/Tolerations:** Control how workloads are distributed across Nodes.

Here's an example PDB YAML definition that ensures that at least two Pods remain running during disruptions:

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-app-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: my-app
```


Advanced Kubernetes Interview Questions

This section covers advanced Kubernetes interview questions, focusing on stateful applications, multi-cluster management, security, performance optimization, and troubleshooting. If you're applying to a senior position, make sure to check these out.

What are StatefulSets, and how do they differ from Deployments?

A StatefulSet is used for **stateful applications** that require stable network identities, persistent storage, and ordered deployment. Unlike Deployments, StatefulSets ensure that:

- Pods have stable and unique network identities, where Pods are named like `pod-0`, `pod-1` etc.
- Pods are created, updated, and deleted in order.
- Each Pod retains persistent storage across restarts. Persistent storage is defined as part of the StatefulSet YAML definition.

What is a service mesh, and why is it used in Kubernetes?

A service mesh manages **service-to-service communication**, providing:

- Traffic management (load balancing, canary deployments).
- Security (mTLS encryption between services).
- Observability (tracing, metrics, logging)

All these features are included in the infrastructure layer, so no code changes are required.

Popular Kubernetes service mesh solutions include: Istio, Linkerd and Consul.

How can you secure a Kubernetes cluster?

Follow the 4C security model to secure a Kubernetes cluster:

1. **Cloud provider security:** Use IAM roles and firewall rules.
2. **Cluster security:** Enable RBAC, audit logs, and API server security.
3. **Container security:** Scan images and use non-root users.
4. **Code security:** Implement secrets management and use network policies.

What are Taints and Tolerations in Kubernetes?

Taints and Tolerations control where Pods run:

- **Taints:** Mark nodes to reject Pods.
- **Tolerations:** Allow Pods to ignore certain Taints.

Here's an example for tainting a node only to accept specific workloads:

```
kubectl taint nodes node1 key1=value1:NoSchedule
```

POWERED BY datalab

This means no Pod can be scheduled on `node1` until it has the required Toleration.

A Toleration is added in the `PodSpec` section like the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: “value1”
    effect: "NoSchedule"
```

POWERED BY datalab

The Pod would be allowed to be scheduled on `node1`.

What are Kubernetes sidecar containers, and how are they used?

A sidecar container runs alongside the main application container inside the same Pod. It is commonly used for:

- **Logging and monitoring** (e.g. collecting logs with Fluentd).
- **Security proxies** (e.g. running Istio's Envoy proxy for service mesh).
- **Configuration management** (e.g. syncing application settings).

Example sidecar container for log processing:

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-sidecar
spec:
  containers:
    - name: main-app
      image: my-app
      volumeMounts:
        - mountPath: /var/log
          name: shared-logs
    - name: log-collector
      image: fluentd
      volumeMounts:
        - mountPath: /var/log
          name: shared-logs
  volumes:
    - name: shared-logs
      emptyDir: {}
```


The sidecar container runs Fluentd, which collects logs from the main container via a shared volume.

Name three typical Pod error causes and how they can be fixed.

Pods can get stuck in *Pending*, *CrashLoopBackOff*, or *ImagePullBackOff* states:

- **Pod stuck in Pending:** Check node availability and resource limits. You can check the events of the Pod for further information.
- **CrashLoopBackOff:** Investigate app logs and check misconfigurations.
- **ImagePullBackOff:** Ensure the correct image name and pull credentials. Again, investigate the Pod's events for further information.

You can check the events of the Pod using the `describe` command:

```
kubectl describe pod <pod-name>
```


What is a Kubernetes mutating admission webhook, and how does it work?

A mutating admission webhook allows **real-time modification** of Kubernetes objects before they are applied to the cluster and persisted. It runs a dynamic admission controller in Kubernetes that intercepts API requests before objects are persisted in etcd. It can modify the request payload by injecting, altering, or removing fields before allowing the request to proceed.

They are commonly used for:

- Injecting sidecars.
- Setting default values for Pods, Deployments, or other resources.
- Enforcing best practices (e.g. automatically assigning resource limits).

- Adding security configurations (e.g. requiring labels for audit tracking).

How do you implement zero-downtime Deployments in Kubernetes?

Zero-downtime Deployments ensure that updates **do not interrupt live traffic**. Kubernetes achieves that using:

- Rolling updates (default, gradually replacing old Pods with new ones).
- Canary deployments (testing with a subset of users).
- Blue-green deployments (switching between production and test environments).

Readiness probes help Kubernetes avoid traffic being sent to unready Pods.

What are Kubernetes Custom Resource Definitions (CRDs), and when should you use them?

A Custom Resource Definition (CRD) extends Kubernetes APIs, allowing users to define and manage custom resources. They are used for specific use cases where the Kubernetes API should still be used for managing these resources, such as:

- Managing custom applications (e.g., Machine learning models).
- Enabling Kubernetes operators for self-healing applications.

Example CRD YAML definition:

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: shirts.stable.example.com
spec:
  group: stable.example.com
  scope: Namespaced
  names:
    plural: shirts
    singular: shirt
    kind: Shirt
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              color:
                type: string
              size:
                type: string
    selectableFields:
    - jsonPath: .spec.color
    - jsonPath: .spec.size
    additionalPrinterColumns:
    - jsonPath: .spec.color
      name: Color
      type: string
    - jsonPath: .spec.size
      name: Size
      type: string
```


You could now, for example, retrieve the `shirt` object using kubectl:

```
kubectl get shirts
```

POWERED BY datalab

Discover how to leverage Docker and Kubernetes for machine learning workflows in this hands-on **tutorial on containerization**.

What are Kubernetes operators, and how do they work?

A Kubernetes operator extends Kubernetes functionality by automating the deployment, scaling, and management of complex applications. It is built using CRDs and custom controllers to handle application-specific logic.

Operators work by defining custom resources in Kubernetes and watching for changes in the cluster to automate specific tasks.

These are the key components of an operator:

- **Custom Resource Definition (CRD):** Defines a new Kubernetes API resource.
- **Custom controller:** Watches the CRD and applies automation logic based on the desired state.
- **Reconciliation loop:** Continuously ensures the application state matches the expected state.

Kubernetes Interview Questions for Administrators

Kubernetes administrators maintain, secure, and optimize clusters for production workloads. This section covers Kubernetes interview questions focusing on cluster management.

How do you backup and restore an etcd cluster in Kubernetes?

Etcd is the key-value store that holds the entire cluster state. Regular backups are essential to avoid data loss.

You can create a backup using the below command:

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> \
  snapshot save <backup-file-location>
```

POWERED BY datalab

If you now want to restore from a backup, you can run:

```
etcdutl --data-dir <data-dir-location> snapshot restore snapshot.db
```

POWERED BY datalab

How do you safely upgrade a Kubernetes cluster?

Upgrading a Kubernetes cluster is a multi-step process that requires minimal downtime and maintains cluster stability. Administrators should follow a structured approach to prevent issues during the upgrade:

1. Drain and backup etcd, before the upgrade to ensure that you can restore it in case the upgrade fails.
2. Upgrade the control plane node.
3. Upgrade kubelet and kubectl on control plane nodes.
4. Upgrade worker nodes one by one. Before upgrading, each worker node must be drained to prevent workload disruption.
5. Upgrade cluster add-ons (e.g., Ingress controllers, monitoring tools, ...).

How do you monitor a Kubernetes cluster?

Kubernetes administrators must monitor CPU, memory, disk, networking, and application health. The following tools are recommended for these tasks:

- **Prometheus + Grafana:** Collect and visualize cluster metrics. Create real-time alerts to get notified in case there are issues.
- **Loki + Fluentd:** Collect and analyze logs.
- **Kubernetes dashboard:** UI-based cluster monitoring.
- **Jaeger/OpenTelemetry:** Distributed tracing.

How do you secure a Kubernetes cluster?

Security is a key aspect, and every administrator should follow best practices to secure a Kubernetes cluster:

- **Enable RBAC:** Restrict user access using Roles and RoleBindings.
- **Pod security admission:** Use admission controllers to enforce the Pod security standards.
- **Enforce NetworkPolicies:** Restrict Pod communication, as by default, all Pods can communicate with each other.
- **Rotate API tokens and certificates regularly.**
- **Use secrets management:** Use tools like Vault, AWS Secrets Manager, etc.

Learn how Kubernetes is implemented in the cloud with this guide on **container orchestration using AWS Elastic Kubernetes Service (EKS)**.

How do you set up Kubernetes logging?

Centralized logging is required for debugging and auditing. Two different logging stack options:

- **Loki + Fluentd + Grafana** (Lightweight and fast).
- **ELK Stack (Elastic, Logstash, Kibana)** (Scalable and enterprise-grade).

How do you implement high availability in Kubernetes?

High availability is essential to avoid downtime of applications running in your Kubernetes cluster. You can ensure high availability by:

- **Using multiple control plane nodes.** Multiple API servers prevent downtime if one fails.
- **Enabling the cluster autoscaler.** This automatically adds/removes nodes based on demand.

How do you implement Kubernetes cluster multi-tenancy?

Multi-tenancy is quite important if you are setting up Kubernetes for your company. It allows multiple teams or applications to share a Kubernetes cluster securely without interfering with each other.

There are two types of multi-tenancy:

1. **Soft multi-tenancy:** Uses Namespaces, RBAC, and NetworkPolicies to isolate on the namespace level.
2. **Hard multi-tenancy:** Uses virtual clusters or separate control planes to isolate a physical cluster (e.g., KCP).

How do you encrypt Kubernetes secrets in etcd?

Etcd stores the complete cluster state, meaning critical information is stored there.

By default, Kubernetes stores secrets unencrypted in etcd, making them vulnerable to compromise. Therefore, it can be crucial to enable **secret encryption at REST** so that secrets are stored and encrypted.

As a first step, you need to create an encryption configuration file and store an encryption/decryption key in that file:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: <BASE64_ENCODED_SECRET>
      - identity: {}
```


The configuration above specifies that Kubernetes will use the `aescbc` provider to encrypt Secret resources, with a fallback to `identity` for unencrypted data.

Next, you need to adapt the `kube-apiserver` configuration file, typically found at `/etc/kubernetes/manifests/kube-apiserver.yaml` on a control plane node, and include the `--encryption-provider-config` flag pointing to the encryption configuration file that you've created:

```
command:
  - kube-apiserver
  ...
  - --encryption-provider-config=/path/to/encryption-config.yaml
```


Save the changes and restart the `kube-apiserver` to apply the new configuration.

How do you manage Kubernetes resource quotas in multi-tenant environments?

Resource quotas prevent a single tenant (namespace) from over-consuming cluster resources, disturbing other tenants from working.

You can define ResourceQuotas for namespaces to give a certain amount of resources to that specific namespace. Users of that namespace can then create resources that consume as much resources as defined in the ResourceQuota of that namespace.

Example ResourceQuota YAML definition:

```
apiVersion: v1
kind: ResourceQuota
metadata:
```

```
  name: namespace-quota
  namespace: team-a
spec:
  hard:
    requests.cpu: "4"
    requests.memory: "8Gi"
    limits.cpu: "8"
    limits.memory: "16Gi"
    pods: "20"
```

POWERED BY datalab

You can check a ResourceQuota of a namespace using:

```
kubectl describe resourcequota namespace-quota -n team-a
```

POWERED BY datalab

What is CoreDNS? How do you configure and use it?

CoreDNS is the **default DNS provider** for Kubernetes clusters. It provides service discovery and allows Pods to communicate using internal DNS names instead of IP addresses.

Features of CoreDNS:

- Handles internal DNS resolution (`my-service.default.svc.cluster.local`).
- Supports custom DNS configuration.
- Load-balances DNS queries across multiple Pods.
- Allows caching for improved performance.

You can configure CoreDNS using the ConfigMap stored in the kube-system namespace. You can view the current settings using:

```
kubectl get configmap coredns -n kube-system -o yaml
```

POWERED BY datalab

Simply update that ConfigMap and apply the changes to adapt the CoreDNS configuration.

Scenario-Based and Problem-Solving Kubernetes Interview Questions

Engineers regularly face complex scalability, networking, security, performance, and troubleshooting challenges in real-world Kubernetes environments, and the interviewers know that. This section presents scenario-based interview questions that test your ability to solve practical Kubernetes problems.

Debugging a slow Kubernetes application

“Your team reports that an application running in Kubernetes has become slow, and users are experiencing **high response times**. How do you solve this problem?”

You can approach the problem using the following steps:

1. Check Pod resource utilization. Increase resources in case they are at their limits.

```
kubectl top pods --sort-by=cpu
kubectl top pods --sort-by=memory
```

POWERED BY datalab

2. Describe the slow Pod to get more information. Look for resource throttling, restart counts, or liveness probe failures.

```
kubectl describe pod <pod-name>
```


3. Check container logs for errors. Look for timeouts, errors, or connection failures.

```
kubectl logs <pod-name>
```


4. Test network latency, as they can slow down applications.

```
kubectl exec -it <pod-name> -- ping my-database
kubectl exec -it <pod-name> -- curl http://my-service
```


5. Verify Kubernetes node health and check for resource exhaustion on nodes.

```
kubectl get nodes
kubectl describe node <node-name>
```


An Nginx web server is running, but the exposed URL fails to connect

"You deployed an Nginx web server in Kubernetes, and the Pod is running fine, but the application is **not accessible via the exposed URL**. What can you do about it?"

Steps to approach the above problem:

1. Verify that the Nginx Pod is running and healthy.

```
kubectl get pods -o wide
kubectl describe pod nginx-web
```


2. Check the Service and port mapping. Ensure the correct port is exposed and matches the Pod's container port. Check that the Service finds the correct Pods.

```
kubectl describe service nginx-service
```


3. Check network policies. If a network policy blocks ingress traffic, the Service won't be accessible.

```
kubectl get networkpolicies
kubectl describe networkpolicy <policy-name>
```


4. Verify Ingress and external DNS configuration.

```
kubectl describe ingress nginx-ingress
```


Kubernetes Deployment fails after an upgrade

"You deployed a new version of your application, **but it fails to start,** causing downtime for your users. How can you fix the problem?"

Approach to tackle the problem:

1. Roll back to the previous working version.

```
kubectl rollout undo deployment my-app
```


2. Check the Deployment history and identify what has changed in the new version.

```
kubectl rollout history deployment my-app
```


3. Check the new Pod's logs for errors.

```
kubectl logs -l app=my-app
```


4. Check readiness and liveness probes.

5. Verify image-pulling issues. Sometimes, the new image is wrong or unavailable.

The application can't connect to an external database

"A microservice running in Kubernetes **fails to connect to an external database**, which is hosted outside the cluster. How can you fix the issue?"

Steps to approach the problem:

1. Verify external connectivity from inside a Pod. Check if the database is reachable from the Kubernetes network.

```
kubectl exec -it <pod-name> -- curl http://my-database.example.com:5432
```


2. Check DNS resolution inside the Pod. If it fails, CoreDNS may be configured wrong.

```
kubectl exec -it <pod-name> -- nslookup my-database.example.com
```


3. Check if network policies exist that block external access, as they can prevent outbound traffic.

```
kubectl get networkpolicies
```

```
kubectl describe networkpolicy <policy-name>
```

POWERED BY datalab

Cluster resources are exhausted, causing new Pods to remain in a pending state

“New pods stay in the state *Pending*. Looking deeper into the pods, we see that the message “0/3 nodes are available: insufficient CPU and memory”. How do you go about debugging and solving the problem?”

Steps to approach the problem:

1. Check cluster resource availability. Look for high CPU/memory usage that prevents scheduling.

```
kubectl describe node <node-name>
kubectl top nodes
```

POWERED BY datalab

2. Check which Pods are consuming the most resources. Set resources and limits for Pods to ensure they are not over-consuming. You can also enforce that for all namespaces in the cluster.

```
kubectl top pods --all-namespaces
```

POWERED BY datalab

3. Scale down non-essential workloads to free up resources.

```
kubectl scale deployment <deployment-name> --replicas=0
```

POWERED BY datalab

4. Increase available nodes to increase cluster resources. You can also add more nodes to the cluster autoscaler if one is used.

Tips for Preparing for a Kubernetes Interview

Through my own experience with Kubernetes interviews, I’ve learned that success requires more than just memorizing concepts. You need to apply your knowledge in real-world scenarios, troubleshoot efficiently, and clearly explain your solutions.

If you want to succeed in Kubernetes interviews, follow the tips below:

1. **Master core Kubernetes concepts**. Ensure you understand Pods, Deployments, Services, Persistent Volumes, ConfigMaps, Secrets, etc.
2. **Get hands-on experience with Kubernetes**. When I was preparing for my interviews, I found that setting up my own Minikube cluster and deploying microservices helped reinforce my understanding. You can also use a managed Kubernetes service from a cloud provider to practice at scale.
3. **Learn how to debug Kubernetes issues**, as troubleshooting is a significant part of working with Kubernetes in the real world. You will probably spend most of your time troubleshooting applications at work.
    - Typical issues that occur include stuck Pods, networking failures, persistent volumes not mounting correctly, and node failures.
4. **Understand Kubernetes networking and load balancing**. Focus on how networking is implemented, how Pods communicate, what Service types exist, and how to expose applications.
5. **Know how to scale and optimize Kubernetes workloads**. Interviewers often ask about scaling strategies and cost optimization. Be proficient with HPA, cluster autoscaler, resource requests, and limits.
6. **Understand Kubernetes security best practices**. Be familiar with RBAC, Pod security context, network policies, and secrets management.
7. **Study the Kubernetes architecture and internals**. Be familiar with the control plane components and how kubelet and container runtime interact.

By combining theoretical knowledge with hands-on practice, and learning from real-world troubleshooting, you will master any Kubernetes interview!

Conclusion

Kubernetes is a powerful but complex container orchestration system. Interviewing for Kubernetes roles requires a deep understanding of theory and real-world problem-solving. Whether you are a junior engineer looking for your first job or a senior engineer aiming for more advanced roles, preparation is always key.

I can't emphasize enough how important practice is. It helps you find issues in your applications faster and become more confident in your abilities.

If you're looking to strengthen your fundamentals, I highly recommend exploring courses like **Containerization and Virtualization Concepts** to build a solid foundation, followed by **Introduction to Kubernetes** to get practical experience with Kubernetes.

I wish you all the best in your interviews!

AUTHOR
Patrick Brus

I am a Cloud Engineer with a strong Electrical Engineering, machine learning, and programming foundation. My career began in computer vision, focusing on image

classification, before transitioning to MLOps and DataOps. I specialize in building MLOps platforms, supporting data scientists, and delivering Kubernetes-based solutions to streamline machine learning workflows.

TOPICS

Kubernetes MLOps

Learn more about Docker and Kubernetes with these courses!

TRACK

Containerization and Virtualization with Docker and Kubernetes

13 hr

Learn the power of Docker and Kubernetes, this interactive track will allow you to build and deploy applications in modern environments.

See Details → Start Course

See More →

Related

BLOG

Top 34 Cloud Engineer Interview Questions and Answers in 2025

BLOG

Top 26 Docker Interview Questions and Answers for 2025

BLOG

Top 30 MLOps Interview Questions and Answers for 2025

See More →