PRODUCTION SRE & DEVOPS INTERVIEW GUIDE 2026

Kubernetes Interview Questions 2026

100+ Production Questions and Answers: From Primitives to Control Plane Internals, Network Routing, Storage, RBAC & Incident Triage

---

Author: Naveed Ahmed (Senior Platform Engineer & SRE)

Ecosystem: blog.naveedkumbhar.com | interview.naveedkumbhar.com | k8s.naveedkumbhar.com

Edition: 2026 Complete Production Reference • Verified runnable CLI diagnostics

Operational Table of Contents

This handbook compiles 100+ production interview questions across eight architectural modules:

- **Module 1:** Architecture & Control Plane Internals (API Server, etcd, kube-scheduler, kubelet PLEG)
- **Module 2:** Pods, Containers & Workload Controllers (Sidecars, StatefulSets, DaemonSets, Jobs)
- **Module 3:** Networking, Service Discovery & Ingress (CNI, CoreDNS, kube-proxy, Ingress, Gateway API)
- **Module 4:** Storage, CSI & Volume Management (Persistent Volumes, CSI drivers, reclaim policies)
- **Module 5:** Security, RBAC & Admission Control (Validating Webhooks, Pod Security Standards, mTLS)
- **Module 6:** Scheduling, Autoscaling & Quotas (CFS throttling, HPA v2, VPA, Karpenter, Bin Packing)
- **Module 7:** Observability, Metrics & Health Probes (Liveness vs Readiness vs Startup probes)
- **Module 8:** SRE Incident Triage & Production Runbooks (CrashLoopBackOff, OOMKilled, NotReady nodes)

Module 1: Architecture & Control Plane Internals

> **Module 1 Capsule:** Control plane components coordinate state through the API Server backed by etcd. The API server validates requests via admission controllers, stores declarative intent as revisions in etcd, and notifies worker controllers via asynchronous watch channels. Controllers continuously reconcile observed cluster reality against declared state.

Q1: What happens under the hood when you run 'kubectl apply -f deployment.yaml'?

kubectl validates syntax and sends an HTTP POST/PATCH request with JWT credentials to kube-apiserver. The request passes through Authentication, Authorization (RBAC), and Admission Controllers (Mutating then Validating). The API server writes the Deployment manifest to etcd and increments the resource version. The Deployment Controller detects the new revision, generates a ReplicaSet, and the ReplicaSet Controller generates Pod definitions in Pending state. Kube-scheduler filters nodes via predicates, ranks candidate nodes via scoring functions, and writes a node binding. Finally, the target worker node kubelet detects the scheduled Pod via watch stream, invokes CRI containerd to pull images and initialize Linux namespaces/cgroups, and reports container Running state back to the API server.

Q2: How does etcd maintain distributed consistency, and what causes quorum loss?

etcd utilizes the Raft consensus algorithm where a single elected leader accepts all read/write proposals, replicates log entries to follower nodes, and commits once a majority quorum of `(N/2) + 1` nodes acknowledge replication. In a 3-node cluster, quorum is 2; losing 2 nodes causes quorum loss, transitioning etcd into read-only mode. Severe disk I/O latency (failing the 10ms fdatasync SLA) or network partitioning between Availability Zones can trigger repeated leader elections and API server timeouts.

Module 2: Workload Controllers & Packaging (Helm vs Operator)

Module 2 Capsule: Workloads balance stateless horizontal scaling (Deployments) against ordered ordinal identity and stable network indices (StatefulSets). DaemonSets guarantee per-node agent placement, while native sidecar containers manage initialization order and graceful background shutdown.

Dimension	Helm Chart Packaging	Kubernetes Custom Operator
Lifecycle Scope	Day-1 Installation & Upgrades	Day-2 Automated Continuous Operations
State Reconciliation	Static template rendering via CI/CD	Continuous control loop via Custom Controller
Automated Failover	Requires human or external CI trigger	Automated leader election, backups, and recovery
Stateful Management	Basic Helm hooks; cannot detect split-brain	Domain logic handles etcd/Postgres replication

Module 8: SRE Incident Triage & Battle Plans

Module 8 Capsule: SRE incident triage requires systematic fault-tree elimination. Isolate pod-level errors (OOMKilled exit code 137, CrashLoopBackOff exit code 1/139), node-level failures (DiskPressure, Kubelet PLEG not healthy), or control plane bottlenecks (API Server rate limiting, etcd latency).

Incident Triage Heuristic: CrashLoopBackOff vs OOMKilled

```
# Check exit code and previous termination reason
kubectl get pod payment-api-7b89-xjk2 -o jsonpath='{.status.containerStatuses[0].lastState.t

# Exit Code 137: Linux kernel OOM Killer terminated process due to memory limit
# Exit Code 1 / 139: Unhandled application exception, missing env var, or segfault
kubectl logs payment-api-7b89-xjk2 --previous --tail=100
```