Frequently Asked Questions: Getting Started with Kubernetes

Author: Kamakshaiah Musunuru | **Published:** 11 September 2026 | **Category:** FAQ | **Read time:** 2 min read

Abstract

A practical FAQ covering the essentials of Kubernetes — pods, deployments, services, ingress, namespaces and scaling — for teams moving from basic containers to orchestration.

Keywords: Kubernetes, pods, deployments, services, ingress, namespaces, scaling, orchestration

Tags: Kubernetes, containers, orchestration, DevOps, FAQ

Q1: What is Kubernetes and why use it?

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling and management of containerised applications. It solves problems that emerge when running containers at scale: scheduling, health checks, rolling updates, service discovery and self-healing.

Q2: What are the core objects I need to know?

- Pod: the smallest deployable unit, one or more containers that share a network namespace - Deployment: declaratively manages the desired state of stateless pods, including replicas and rolling updates - Service: a stable network abstraction that load-balances traffic to a set of pods - Ingress: exposes HTTP and HTTPS routes from outside the cluster to services - Namespace: partitions cluster resources for isolation and organisation - ConfigMap and Secret: decouple configuration and sensitive data from images

Q3: How do deployments differ from pods?

You rarely create pods directly. A Deployment declares the desired number of replicas and the pod template. The controller reconciles the actual state to the desired state — replacing failed pods, performing rolling updates and rolling back on failure.

Q4: How do services route traffic to pods?

Pods are ephemeral and get new IPs, so Services provide a stable virtual IP and DNS name. A Service selects pods via label selectors and load-balances among them. For external access, an Ingress routes by host and path to the appropriate Service.

Q5: How do I manage stateful workloads?

Databases and caches need persistent storage and stable identities. Use StatefulSets with PersistentVolumes, ordering guarantees and stable network identifiers. Back up data and manage storage classes carefully.

Q6: How do I handle configuration and secrets?

Store non-sensitive configuration in ConfigMaps and sensitive values in Secrets. Mount them as environment variables or volumes. Rotate secrets regularly and never commit them to source control.

Q7: How do I keep clusters healthy in production?

- Set resource requests and limits on every container - Configure liveness, readiness and startup probes - Use HorizontalPodAutoscaler for demand-based scaling - Apply resource quotas and network policies per namespace - Monitor with metrics, logs and tracing, and run regular upgrades - Use GitOps tools to manage cluster state declaratively

Kubernetes is powerful but complex. Start small, standardise on templates and GitOps, and invest in observability before scaling aggressively.

References

Kubernetes Documentation; Kubernetes The Hard Way; CNCF Cloud Native Landscape; 12-Factor App.