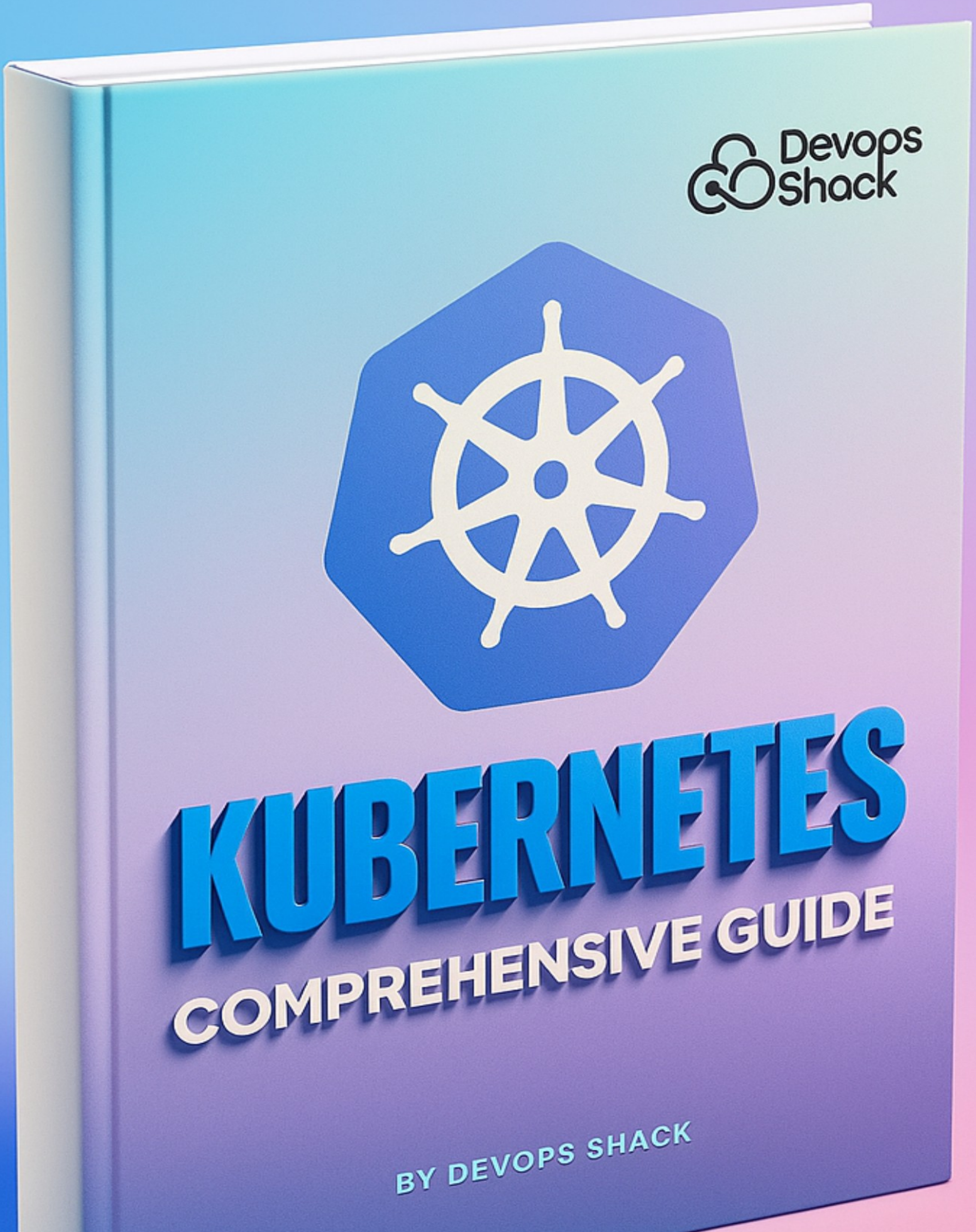

Devops Shack

KUBERNETES

COMPREHENSIVE GUIDE

BY DEVOPS SHACK

DevOps Shack

Table of Contents

Introduction

Kubernetes, originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF), is a powerful open-source container orchestration platform. It automates the deployment, scaling, and management of containerized applications. As enterprises increasingly adopt microservices and containers, Kubernetes has become the de facto standard for container orchestration in both cloud-native and hybrid infrastructures.

This guide is your all-in-one resource for mastering Kubernetes—from fundamental concepts to advanced deployment patterns, security, observability, and production-grade practices. Whether you are a beginner
looking to get started or a professional aiming to deepen your understanding, this document is structured to provide value at every level.

1. What is Kubernetes?

Kubernetes (K8s) is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. It enables declarative configuration and automation and supports a wide range of container tools including Docker and containerd. Kubernetes handles service discovery, load balancing, rolling updates, health checks, and much more, making it ideal for managing microservices architectures.

2. Kubernetes Architecture Overview

Kubernetes has a client-server architecture that includes the Control Plane (master components) and Worker Nodes. The Control Plane manages the cluster's overall state, while nodes run the actual workloads. Core architectural components include the API server, controller manager, scheduler, etcd, kubelet, and kube-proxy. It's a modular and extensible system designed for high availability and fault tolerance.

3. Key Kubernetes Components (Master and Node)

- Control Plane (Master):
  - API Server: Frontend for Kubernetes; all operations go through it.
  - Controller Manager: Watches the cluster state and performs automated reconciliation.
  - Scheduler: Assigns pods to nodes based on constraints and resource availability.
  - etcd: Distributed key-value store used as the source of truth.

- Node Components:
  - kubelet: Ensures containers are running in pods.
  - kube-proxy: Handles networking, routes traffic to appropriate pods.
  - Container Runtime: Executes containers (e.g., containerd, Docker).

4. Understanding Kubernetes Clusters

A Kubernetes cluster consists of one or more master nodes (control plane) and worker nodes. It provides a unified system to manage all the containerized applications across multiple machines. It uses a declarative model, where you define the desired state, and Kubernetes reconciles it continuously.

5. Kubernetes API Server

The API Server is the central management entity. All Kubernetes components (kubectl, controllers, etc.) communicate with it. It exposes a RESTful interface, validates requests, and updates the etcd store. It's stateless and can be horizontally scaled.

6. kubelet and kube-proxy

- kubelet: An agent running on each node, it reads the pod specs and ensures containers are running as defined.
- kube-proxy: Handles networking on the node. It manages IPtables or IPVS rules to route traffic to the appropriate pod.

7. etcd – The Brain of Kubernetes

etcd is a distributed, consistent key-value store used for storing all cluster data — configurations, state, metadata. It's a critical component of the control plane. It uses the Raft consensus algorithm to ensure consistency across distributed systems.

8. Control Plane vs Node Components

The Control Plane orchestrates the cluster (API Server, etcd, Scheduler, Controller Manager). The Node Components run actual workloads (kubelet, kube-proxy, container runtime). This separation allows clear responsibility and scaling models.

9. Kubernetes Installation Methods (Minikube, kubeadm, k3s)

- Minikube: Local development, single-node cluster.
- kubeadm: Production-grade cluster installer using CLI.
- k3s: Lightweight Kubernetes distribution for edge/IoT.

10. Pods – The Smallest Deployable Unit

A pod is a group of one or more containers with shared storage/network. Containers in a pod share the same network namespace. Pods are ephemeral — they can die and be replaced automatically by controllers like Deployments.

11. ReplicaSets and Scaling Pods

ReplicaSet ensures a specified number of pod replicas are running at all times. If a pod crashes, it spins up a new one. It supports declarative scaling — just change the replica count, and Kubernetes reconciles it.

12. Deployments – Declarative Updates

A Deployment provides declarative updates to ReplicaSets and Pods. You can perform rolling updates, rollbacks, and pause/resume deployments. It's the most commonly used workload controller.

13. StatefulSets – Managing Stateful Applications

StatefulSets manage stateful applications like databases. Each pod gets a unique identity and stable storage. Ideal for use cases requiring persistent identity, hostname, and stable storage like MongoDB, Kafka, etc.

14. DaemonSets – Running a Pod on All Nodes

DaemonSet ensures that a pod runs on every (or selected) node. Common for log collection, monitoring agents (e.g., Fluentd, Prometheus Node Exporter).

15. Jobs and CronJobs

- Job: One-time task. Runs until successful completion.
- CronJob: Scheduled task. Uses cron syntax to run jobs periodically.

16. Services – ClusterIP, NodePort, LoadBalancer

Services provide stable IPs and DNS for pods. Types:

- ClusterIP: Internal access only.
- NodePort: Exposes a service on a static port on each node.
- LoadBalancer: Integrates with cloud load balancers (AWS ELB, GCP LB).

17. Ingress Controllers and Ingress Resources

Ingress exposes HTTP/HTTPS services externally. It manages routing rules, TLS termination, and path-based routing. Requires an Ingress Controller (e.g., NGINX, Traefik).

18. ConfigMaps – Managing Configuration

ConfigMaps store non-sensitive key-value configuration for pods. They decouple configuration from application code and can be mounted as files or injected as environment variables.

19. Secrets – Secure Storage of Sensitive Data

Secrets store confidential data like passwords, tokens, keys. They are base64-encoded and can be used like ConfigMaps, but with RBAC restrictions and stricter access.

20. Namespaces – Logical Separation

Namespaces logically isolate resources within the same cluster. Useful for multi-tenancy, environment separation (dev, staging, prod), and access control.

21. Labels and Selectors

Labels are key-value pairs used to identify objects. Selectors help group objects using labels (e.g., Deployments selecting pods).

22. Annotations

Annotations store non-identifying metadata about objects (e.g., build timestamp, release ID). Unlike labels, they are not used for selection.

23. Kubernetes Volumes and Persistent Volumes

Volumes provide data persistence to pods. Kubernetes supports various volume types (hostPath, NFS, EBS, CSI drivers). Volumes can outlive container restarts.

24. PersistentVolumeClaim (PVC) & Dynamic Provisioning

PVCs are user requests for storage. PVs are actual storage. Dynamic provisioning automatically creates a PV when a PVC is requested, using a StorageClass.

25. Storage Classes

StorageClasses define types of storage (e.g., fast SSD, slow HDD). PVCs reference a StorageClass to request dynamically provisioned volumes.

26. Networking in Kubernetes (CNI, Pod-to-Pod, Service Discovery)

K8s requires all pods to communicate freely. CNI plugins (Calico, Flannel, Cilium) provide this networking. DNS is automatically provided via CoreDNS for service discovery.

27. Network Policies

Network Policies control pod-level traffic. They use label selectors and rules to restrict ingress/egress, adding a security layer in the cluster.

28. RBAC (Role-Based Access Control)

RBAC controls access to the Kubernetes API using Roles, ClusterRoles, RoleBindings, and ClusterRoleBindings. Essential for multi-tenant environments.

29. Service Accounts

Service accounts provide identity for pods to access the Kubernetes API securely. Useful in CI/CD and automation.

30. Taints and Tolerations

Taints repel pods from nodes, and tolerations allow exceptions. This controls which pods can run on which nodes, ensuring workload isolation.

31. Node Affinity and Pod Affinity/Anti-Affinity

- Node Affinity: Schedule pods to specific nodes.
- Pod Affinity/Anti-Affinity: Schedule pods based on proximity to or separation from other pods.

32. Init Containers

Init containers run before app containers in a pod and are ideal for setup tasks (e.g., wait-for-service, init DB). They run sequentially.

33. Sidecar Containers

Sidecars extend the primary container in a pod — e.g., logging agents, proxy (Envoy in Istio), or data sync agents. They share storage and network with the main container.

34. Liveness and Readiness Probes

- Liveness Probe: Checks if a pod is alive; restarts if failed.

- Readiness Probe: Checks if a pod is ready to serve traffic.

35. Horizontal Pod Autoscaler (HPA)

HPA scales pods based on CPU/memory usage or custom metrics. It's configured to maintain performance under load automatically.

36. Vertical Pod Autoscaler (VPA)

VPA automatically adjusts CPU and memory resource requests and limits for containers based on usage patterns.

37. Cluster Autoscaler

Cluster Autoscaler adds/removes nodes from the cluster based on pending pod requirements. Supported on AWS, GCP, Azure, etc.

38. Helm – Kubernetes Package Manager

Helm simplifies deployment of applications using "charts" (pre-configured Kubernetes resources). It supports templating, versioning, and lifecycle hooks.

39. Custom Resource Definitions (CRDs)

CRDs extend Kubernetes APIs by allowing users to define custom resources. Used with controllers/operators to build custom Kubernetes-native APIs.

40. Operators in Kubernetes

Operators automate complex applications in Kubernetes by encoding domain knowledge. Built on CRDs and controllers, they manage app lifecycle (e.g., Prometheus Operator, etcd Operator).

41. Kubernetes Dashboard

A web UI to monitor and manage clusters — view resources, deploy apps, inspect logs, and manage workloads.

42. Logging and Monitoring with Prometheus & Grafana

- Prometheus: Metrics collection and alerting.
- Grafana: Visualizes metrics from Prometheus.
- Together, they form a robust observability stack for Kubernetes.

43. Troubleshooting Kubernetes (kubectl tips)

Use `kubectl describe`, `kubectl logs`, `kubectl get events`, and `kubectl exec` to inspect, debug, and interact with pods, services, and resources.

44. Kubernetes Events and Logs

Events (`kubectl get events`) provide cluster activity. Pod/container logs (`kubectl logs`) help debug runtime issues.

45. Backup and Restore Strategies (Velero, etcd backup)

- Velero: Backup/restore Kubernetes resources and volumes.
- etcd: Snapshot-based backups for control plane data.
- Best practice: automate regular backups.

46. Security Best Practices (Pod Security, Network Policies)

Enforce:

- RBAC and least privilege.
- PodSecurityAdmission (PSPs are deprecated).
- Restrict hostPath, runAsRoot.
- Enable audit logs, image scanning.

47. Kubernetes Upgrades and Versioning

Kubernetes follows a quarterly release cycle. Upgrades must be planned carefully, ensuring etcd and control plane compatibility. Use `kubeadm upgrade` for managed upgrades.

48. GitOps with ArgoCD or Flux

GitOps uses Git as the source of truth for deployment. ArgoCD/Flux sync Git changes to the cluster automatically. It improves visibility, traceability, and automation.

49. CI/CD in Kubernetes (GitHub Actions, Jenkins Pipelines)

Use tools like:

- GitHub Actions to build, test, and deploy containers to Kubernetes.
- Jenkins with Kubernetes plugin for dynamic agent creation and Kubernetes-native workflows.

50. Real-World Kubernetes Architecture Design

Production-ready architecture includes:

- Multi-zone clusters
- Node pools (separate compute types)
- Monitoring/alerting
- CI/CD integration
- Secrets management
- Security policies

- **Auto-scaling**
- **Logging pipeline (ELK/EFK stack)**
- **GitOps workflows**