PEER-REVIEWED JOURNAL OF COMPUTER SCIENCE (PRJCS)

ISSN Online : Open Access Scholarly Journal

Getting Started With Kubernetes: A Practical Developer Guide

Kochumol Abraham

Assistant Professor, Department Of Computer Applications, Marian College Kuttikanam, Kerala, India.

Article information

Received: 20th November 2025
Received in revised form: 28th December 2025
Accepted: 4th January 2026
Available online: 9th January 2026

Volume: 1
Issue: 1
DOI: https://doi.org/10.5281/zenodo.18873848

Abstract

Kubernetes has become the standard platform for container orchestration, fundamentally changing how applications are deployed, scaled, and managed in production environments. This paper provides a practical introduction to Kubernetes aimed at developers transitioning from traditional deployment models to container-based architectures. The paper covers core concepts including pods, services, deployments, and namespaces, followed by hands-on guidance for cluster setup, application deployment, and operational management.
A comparative analysis of managed Kubernetes services from major cloud providers is included alongside a discussion of common pitfalls and best practices drawn from production experience. The paper demonstrates that while Kubernetes introduces significant operational complexity, its benefits in scalability, portability, and resource efficiency justify adoption for teams managing distributed applications at scale.

Keywords: - Kubernetes, container orchestration, Docker, microservices, cloud-native, DevOps

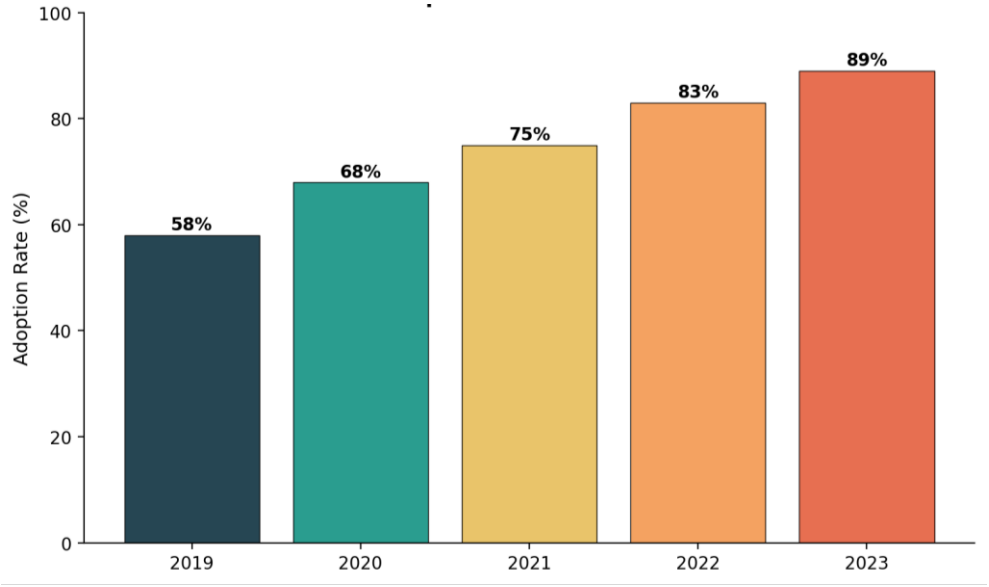
I. INTRODUCTION

Container technology has transformed software delivery by packaging applications with their dependencies into lightweight, portable units. Docker, introduced in 2013, made containerization accessible to mainstream developers, but managing containers at scale quickly revealed the need for automated orchestration [1]. Kubernetes, originally developed by Google based on its internal Borg system and released as open-source in 2014, emerged as the definitive solution to this challenge [2].

The Cloud Native Computing Foundation (CNCF) 2023 survey reports that a large majority of organizations using containers in production run Kubernetes, with adoption growing steadily since 2019 [3]. This widespread adoption reflects Kubernetes' ability to automate deployment, scaling, and operations of application containers across clusters of hosts.

However, the platform's complexity presents a steep learning curve for developers accustomed to traditional deployment workflows.This paper serves as a structured introduction for developers approaching Kubernetes for the first time. It assumes familiarity with containerization concepts and focuses on translating that knowledge into practical Kubernetes proficiency.

Figure 1: Kubernetes adoption rates in production environments, 2019-2023 [3]. Trend data is illustrative based on aggregated CNCF survey reports.



II. CORE ARCHITECTURE

A Kubernetes cluster consists of two primary components: the control plane and worker nodes. The control plane manages the overall cluster state and makes scheduling decisions, while worker nodes run the actual application workloads [4]. Understanding this architecture is fundamental to effective Kubernetes operation.

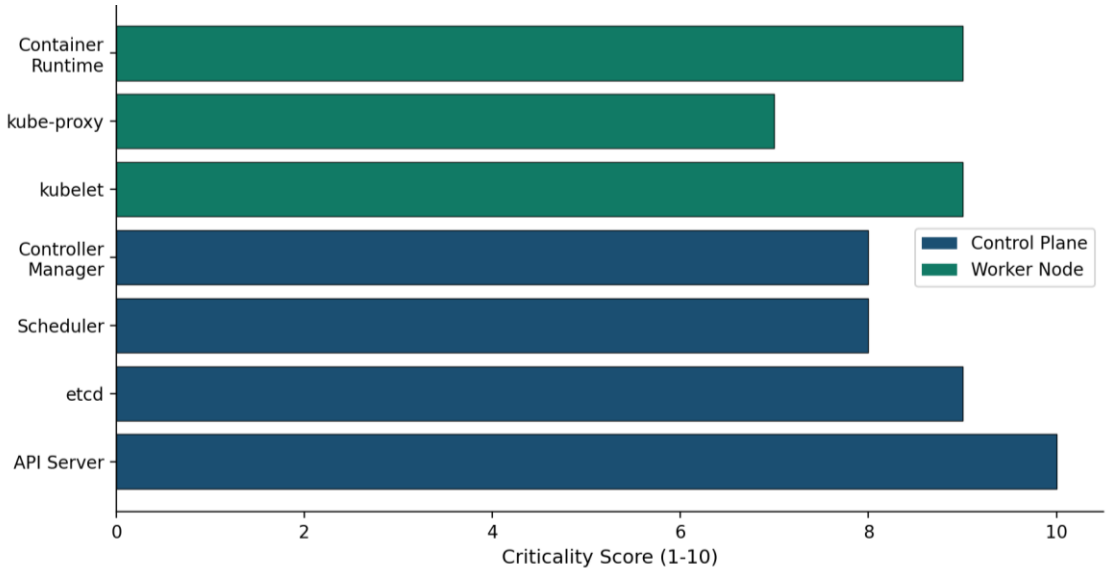
A. Control Plane Components

The API server acts as the central interface for all cluster operations, processing RESTful requests and updating the cluster state stored in etcd, a distributed key-value store that serves as the single source of truth for cluster configuration [5]. The scheduler assigns newly created pods to worker nodes based on resource availability, affinity rules, and constraint policies. The controller manager runs control loops that continuously reconcile the desired state (defined in resource manifests) with the actual state of the cluster.

B. Worker Node Components

Each worker node runs a kubelet agent that communicates with the control plane, ensuring that containers described in pod specifications are running and healthy. The kube-proxy manages network rules that enable communication between pods across nodes. The container runtime (containerd or CRI-O) handles the actual execution of containers [6].

Figure 2: Kubernetes core components and their criticality scores [4]. Criticality scores are illustrative assessments.



III. FUNDAMENTAL RESOURCES

A. Pods

The pod is the smallest deployable unit in Kubernetes, representing one or more containers that share network namespace and storage volumes. While pods can contain multiple containers, the most common pattern is a single container per pod with optional sidecar containers for logging, monitoring, or proxy functions [7]. Pods are ephemeral by design; they can be terminated and replaced at any time, which requires applications to be stateless or use external storage for persistent data.

B. Deployments and ReplicaSets

Deployments provide declarative updates for pods, managing the creation and scaling of ReplicaSets that maintain a specified number of pod replicas. When a deployment is updated, Kubernetes performs a rolling update by default, gradually replacing old pods with new ones to maintain availability [8]. Rollback capabilities allow reverting to a previous deployment version if issues are detected.

C. Services and Networking

Services provide stable network endpoints for groups of pods, abstracting the dynamic nature of pod IP addresses. ClusterIP services expose pods within the cluster, NodePort services expose pods on each node's IP, and LoadBalancer services integrate with cloud provider load balancers for external access [9]. Ingress resources manage HTTP and HTTPS routing from outside the cluster to internal services.

Table I. Kubernetes Service Types and Use Cases

Service Type	Accessibility	Use Case	Cloud Integration
ClusterIP	Internal only	Inter-service communication	None required
NodePort	External via node IP	Development, testing	None required
LoadBalancer	External via cloud LB	Production traffic	Cloud provider required
ExternalName	DNS alias	External service mapping	None required

IV. MANAGED KUBERNETES SERVICES

Major cloud providers offer managed Kubernetes services that abstract control plane management, reducing operational overhead. Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS) handle control plane provisioning, upgrades, and availability, allowing teams to focus on workload deployment [10].

Table II. Managed Kubernetes Service Comparison

Feature	AWS EKS	Google GKE	Azure AKS
Control Plane Cost	$0.10/hr	Free tier (one zonal cluster)	Free
Node Auto-Scaling	Cluster Autoscaler	Node Auto-Provisioning	Cluster Autoscaler
Max Nodes/Cluster	5,000	15,000	5,000
Serverless Option	Fargate	Autopilot	Virtual Nodes
Default CNI	VPC CNI	Calico/Cilium	Azure CNI

V. DEPLOYMENT BEST PRACTICES

Production Kubernetes deployments demand attention to several operational concerns. Resource requests and limits should be defined for every container, ensuring the scheduler can make informed placement decisions and preventing individual workloads from consuming excessive resources [11]. Liveness and readiness probes enable Kubernetes to detect and restart unhealthy containers automatically, maintaining application availability.

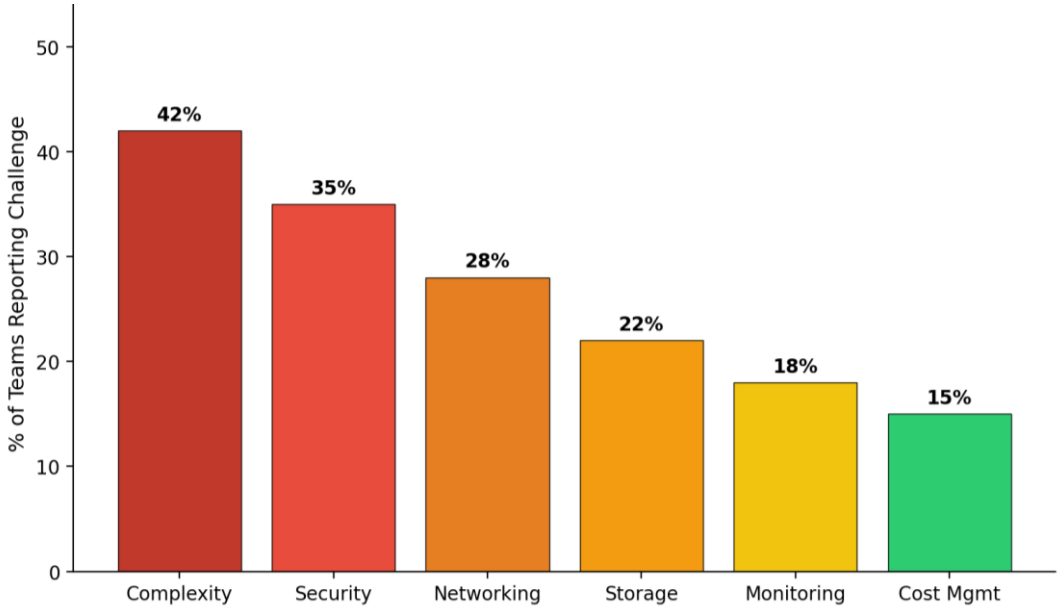
Namespace isolation separates workloads by team, environment, or application, providing logical boundaries for resource quotas and access control. Network policies restrict pod-to-pod communication to only the paths required by the application, reducing the attack surface in the event of a container compromise [12].

Configuration should be externalized from container images using ConfigMaps for non-sensitive data and Secrets for credentials. This separation allows the same image to be deployed across development, staging, and production environments with different configurations [13].

VI. COMMON CHALLENGES

Despite its capabilities, Kubernetes presents significant challenges that teams must anticipate. The CNCF survey identifies complexity as the top barrier to adoption, with a plurality of respondents citing it as their primary concern [3]. Security configuration, particularly around RBAC policies, pod security standards, and image vulnerability scanning, requires dedicated expertise. Figure 3. summarizes the most frequently reported challenges.

Figure 3: Most frequently reported challenges in Kubernetes adoption [3]. Challenge percentages are illustrative based on survey trends.



Persistent storage management presents another common difficulty. While Kubernetes supports various storage backends through the Container Storage Interface (CSI), configuring stateful workloads such as databases requires careful attention to storage classes, persistent volume claims, and backup procedures [14]. Many teams initially avoid running stateful workloads on Kubernetes, using managed database services instead until their operational maturity increases.

VII. CONCLUSION

Kubernetes has established itself as the standard platform for container orchestration, offering powerful automation for deployment, scaling, and management of containerized applications. For developers beginning their Kubernetes journey, the learning curve is substantial but manageable when approached systematically. Starting with core concepts, progressing through managed services, and gradually adopting advanced features such as custom operators and service mesh integration allows teams to build competence incrementally. The investment in Kubernetes proficiency pays dividends in application portability, operational efficiency, and the ability to manage distributed systems at scale.

REFERENCES

[1] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, vol. 2014, no. 239, Mar. 2014.
[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *ACM Queue*, vol. 14, no. 1, pp. 70–93, Jan. 2016.
[3] Cloud Native Computing Foundation, “CNCF Annual Survey 2023,” The Linux Foundation, San Francisco, CA, USA, 2023.
[4] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: Up and Running*, 3rd ed. Sebastopol, CA, USA: O’Reilly Media, 2022.
[5] etcd Authors, “etcd: A Distributed, Reliable Key-Value Store,” The Linux Foundation, San Francisco, CA, USA, 2023.
[6] Kubernetes Authors, “Kubernetes Components,” *Kubernetes Documentation*, The Linux Foundation, 2023.
[7] B. Burns and D. Oppenheimer, “Design Patterns for Container-Based Distributed Systems,” in *Proc. 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2016.
[8] Kubernetes Authors, “Deployments,” *Kubernetes Documentation*, The Linux Foundation, 2023.
[9] Kubernetes Authors, “Service,” *Kubernetes Documentation*, The Linux Foundation, 2023.
[10] Gartner, “Magic Quadrant for Container Management,” Gartner Research, Stamford, CT, USA, 2023.
[11] Kubernetes Authors, “Managing Resources for Containers,” *Kubernetes Documentation*, The Linux Foundation, 2023.
[12] Martin and M. Hausenblas, *Hacking Kubernetes: Threat-Driven Analysis and Defense*. Sebastopol, CA, USA: O’Reilly Media, 2022.
[13] Kubernetes Authors, “ConfigMaps,” *Kubernetes Documentation*, The Linux Foundation, 2023.
[14] Container Storage Interface Authors, “CSI Specification v1.8,” The Linux Foundation, San Francisco, CA, USA, 2023.

.