Kubernetes Research

1 Introduction

Kubernetes is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. It was originally created at Google, based on years of experience operating large-scale systems such as Borg and Omega, and was later donated to the Cloud Native Computing Foundation (CNCF). As modern applications increasingly rely on distributed architectures and microservices, Kubernetes provides a standardized way to run containers reliably across clusters of machines.

At its core, Kubernetes uses a declarative model: developers describe the desired state of an application, and the system continuously works to ensure that the actual state matches it. This allows for automated restarts, self-healing, rolling updates, and horizontal scaling. Because of its flexibility, strong ecosystem, and cloud-agnostic design, Kubernetes has become one of the most widely adopted platforms for running cloud-native workloads.

This research provides the background knowledge required for the workshop, including why Kubernetes exists, the challenges it solves, how it compares to alternative solutions, and when it is appropriate to use. It also explains how Kubernetes can be used.

2 Why Kubernetes Exists: Background, History, and the Problems It Solves

As software systems became more distributed and complex, organizations began shifting from monolithic applications to microservices deployed across clusters of machines. Containers made it easier to package and run these services consistently, but they did not address the operational challenges that emerged at scale. Managing hundreds or thousands of containers manually on different servers, with different resource requirements and different lifecycles, quickly became unmanageable. This growing complexity created the need for an automated orchestration system capable of running containers reliably in large, dynamic environments.

2.1 Historical background

Before Kubernetes existed, Google operated some of the largest distributed systems in the world. For them to manage internal workloads for services such as Search, Gmail, and YouTube, Google developed cluster management systems including Borg, and later Omega. These systems introduced ideas such as declarative state, automated scheduling, container restarts, and horizontal scaling. Google publicly documented this work in research papers, which later became the foundation for Kubernetes.

Kubernetes was announced in June, 2014 and version 1.0 was released on July 21, 2015. With contributions from both Google and the wider community, it evolved into a general-purpose orchestration platform that could run anywhere, this being on premises, in the cloud, or in hybrid environments. Its open governance model and extensibility helped it quickly become the industry standard for container orchestration.

2.2 What problems does it solve?

Modern enterprise systems often consist of hundreds of services running across multiple teams, environments, and data centers, and all must remain reliable, secure, scalable, and cost-efficient while supporting rapid development cycles. Kubernetes was designed to address these challenges directly. The following subsections detail the core problems enterprises face and how Kubernetes provides solutions.

Managing Containers at Scale Across Large Organizations

Enterprise applications commonly run dozens of microservices, each with multiple replicas. Different teams deploy services independently, resulting in thousands of containers in production. Because of this, manually managing where these containers run, how resources are allocated, and how failures are handled becomes impossible at enterprise scale. Kubernetes includes an advanced scheduler that automatically assigns containers to the most suitable nodes based on

CPU, memory, constraints, and other possible set requirements. This automation allows for efficient resource usage across environments, reduces operational load, and eliminates inconsistent manual deployments. Basically, Kubernetes allows large companies to operate huge, complex systems with fewer human errors, while maintaining a significantly lower operational overhead.

Ensuring Reliability and Self Healing for Services

Large organizations face strict uptime requirements. Outages can cause financial loss and break customer trust. When containers crash or nodes fail, manual intervention may take minutes or hours, which is not something acceptable at an enterprise scale. Thus, Kubernetes continuously monitors system health and maintains self-healing behavior, by automatically restarting failed containers, rescheduling workloads when nodes fail and replacing unhealthy pods. Thus, it reduces downtime and provides predictable reliability for important systems.

Standardizing Deployment Processes Across Teams

Different teams often use different deployment scripts and infrastructure setups, creating inconsistencies that cause failures during releases. Manual deployments increase the risk of configuration drift. Kubernetes uses a declarative model where teams describe the desired state of their applications: replicas, versions, resources, networking, and policies. Kubernetes ensures all environments align with the same definitions, reducing deployment errors.

Safe and Predictable Software Releases

Enterprises need to release new software frequently without impacting customers. Manual updates can often cause downtime or rollback difficulties, which can affect the experience of customers using their products. Kubernetes solves it by providing rolling updates that gradually replace old versions, zero-downtime deployments and instant rollbacks if errors occur during the new deployments. Because of this, teams can deploy multiple updates per day, while maintaining a stable system.

Automatically Scaling Services Based on Demand

Within the context of software, traffic patterns can vary dramatically. Some applications experience seasonal spikes, while others see some unpredictable bursts in traffic. Over-allocating resources leads to high costs, while under-allocating hurts performance and can even affect revenue. Kubernetes provides Horizontal Pod Autoscaling (HPA), Vertical Pod Autoscaling (VPA), and Cluster Autoscaling, enabling automatic scaling based on CPU/memory usage, custom metrics or even external metrics such as request rate. With Kubernetes, companies save costs while ensuring that applications scale smoothly during peak load.

Reliable service discovery

In large distributed systems, containers frequently restart and receive new IP addresses. Without a stable mechanism for service discovery, internal communication breaks. Kubernetes solves it by providing services stable virtual IPs, independent of pod lifecycles. Due to this, teams can build microservices without worrying about dynamic infrastructure changes.

3 Why Kubernetes Is Still Relevant Today

Kubernetes still remains relevant to this day because it continues to address the core operational challenges of modern enterprise systems while evolving alongside new technological demands. As organizations continue to rely more on distributed and cloud-native architectures, Kubernetes provides a standardized platform for deploying and managing applications.

Its open-source nature is another reason for its continued relevance. Kubernetes is developed and maintained by a global community, backed by organizations such as Google, Red Hat, Microsoft, and many more. This collaborative model ensures rapid innovation, transparent development, frequent security improvements, and a large ecosystem of compatible tools. Because it is open source, enterprises can adopt Kubernetes without licensing costs and can extend or customize it to meet their specific needs.

Combined with its ecosystem, widespread industry adoption, and large community knowledge, Kubernetes remains a stable, extensible, and future-proof solution for running modern cloud-native applications.

4 Where to Start and How Kubernetes Works

Kubernetes introduces a new model for running applications, one that differs significantly from traditional server-based deployments. Instead of running programs directly on virtual machines or physical servers, Kubernetes manages applications through a set of abstractions that automate placement, scaling, recovery, and updates.

4.1 Kubernetes as a System of Desired State

At the core of Kubernetes is the idea of desired state. Instead of manually starting and stopping applications, developers describe what the system should look like. This includes how many replicas of an application must run, which container image to use, how the application should be accessed, and what resources it requires. Kubernetes continuously monitors the system and works to ensure that the actual state matches the desired state.

4.2 The Kubernetes Architecture

Kubernetes is built around two major layers: the control plane and the worker nodes.

4.2.1 Control Plane

The control plane acts as the "brain" of the cluster and is responsible for global decision-making. It consists of:

- **API Server**: The entry point for all communication with the cluster. All operations, whether from users or internal components, pass through the API Server.
- **etcd**: A distributed key-value store that holds all cluster state, including both desired and actual configurations.
- **Scheduler**: Determines on which worker node a new Pod should run based on resource availability, constraints, and policies.
- **Controller Manager**: Runs a collection of controllers that continuously compare desired state with actual state and correct any differences.

4.2.2 Worker Nodes

Worker nodes are the machines where applications actually run. Each node includes:

- **kubelet**: An agent that communicates with the control plane and ensures containers are running as expected.
- **Container Runtime**: The software responsible for running containers, such as containerd.
- **kube-proxy**: Manages networking and routing rules to ensure communication between services and Pods.

Together, the worker nodes execute workloads and report status back to the control plane.

4.3 Core Kubernetes Concepts

To understand how to use Kubernetes, it is necessary to understand its fundamental abstractions.

4.3.1 Pods

The smallest deployable unit in Kubernetes. A Pod typically contains a single container, though it can host multiple tightly coupled containers when needed. Pods are ephemeral and may be created, destroyed, or rescheduled automatically.

4.3.2 Deployments

A Deployment manages Pods over time. It allows developers to run multiple replicas, perform rolling updates, roll back to previous versions, and ensure that the correct number of Pods is always available.

4.3.3 ReplicaSets

A lower-level controller used by Deployments to maintain the desired number of identical Pods.

4.3.4 Services

Since Pods receive new IP addresses when restarted, Services provide stable virtual IPs and load balancing between Pod replicas.

4.3.5 Ingress and Gateway API

These components expose applications to external networks. Ingress provides traditional HTTP routing, while the Gateway API offers a more modern and extensible approach to traffic management.

4.3.6 ConfigMaps and Secrets

ConfigMaps store non-sensitive configuration, while Secrets manage sensitive information such as passwords and certificates. Both can be injected into Pods at runtime.

4.3.7 PersistentVolumes and PersistentVolumeClaims

Containers are temporary, but applications often require persistent storage. PersistentVolumes represent storage on the cluster, and PersistentVolumeClaims allow Pods to request that storage in a portable and standardized way.

4.4 How Kubernetes Executes Work: The Reconciliation Loop

The reconciliation loop is the core mechanism that allows Kubernetes to maintain its desired state model. The process works as follows:

1. The user defines a desired state in a YAML manifest.
2. The API Server stores this state in etcd.
3. Controllers detect differences between the desired and actual state.
4. The Scheduler assigns new Pods to appropriate worker nodes.
5. The kubelet starts and monitors containers on those nodes.

6. If failures occur, the control loop repeats until the actual state matches the desired state.

Through this process, Kubernetes is able to recover from failures, reschedule workloads, scale applications, and apply updates reliably.

4.5 How to start working with Kubernetes

Kubernetes has a learning curve. A structured approach is helpful for newcomers:

- Learn the core concepts: Pods, Deployments, Services, ConfigMaps, Secrets, and Ingress.
- Begin with a local cluster using tools such as Minikube, Kind, or Rancher Desktop.
- Deploy a simple application, expose it with a Service, and practice scaling and updating it.
- Explore observability through logs, metrics, and dashboards using tools like Prometheus and Grafana.
- Learn deployment strategies and GitOps tools
- Transition to a managed Kubernetes service such as GKE, EKS, or AKS for production environments.

References

[1] "Kubernetes" Wikipedia, accessed November 2025. `https://en.wikipedia.org/wiki/Kubernetes`

[2] "Kubernetes" The Cloud Native Computing Foundation (CNCF) / Kubernetes.io, accessed November 2025. `https://kubernetes.io/`

[3] "Kubernetes overview" Red Hat Documentation, accessed November 2025. `https://docs.redhat.com/en/documentation/openshift_container_platform/4.17/html/getting_started/kubernetes-overview`