

Kubernetes Complete Documentation

Table of Contents

- **&** What is Kubernetes?
- Kubernetes Architecture
- S Control Plane Components
- Morker Node Components
- Subernetes Workflow
- Architecture Diagrams
- **Q** Key Benefits

& What is Kubernetes?

Kubernetes (also known as **K8s**) is an open-source container orchestration system that automates the deployment, scaling, and management of containerized applications. The name comes from the Greek word for "helmsman" - the person who steers a ship.[1][2]

abo Why K8s?

• K (first letter) + 8 (eight letters in between) + s (last letter) = K8s

& Core Purpose

Kubernetes solves the **container orchestration** problem by automating:

- **Deployment** of containers
- **Scaling** applications up and down
- S Management of container lifecycle
- **Self-healing** capabilities

History and Evolution

★ Traditional Deployment Era

```
graph LR
   A[Physical Server] --> B[Static IP]
   B --> C[Manual Setup]
   C --> D[High Maintenance]
```

Challenges:

- 🐧 **Expensive** hardware procurement
- Manual environment setup
- **Poor** scalability

- **O Vendor lock-in**
- Cloud Revolution (AWS Era)

```
graph LR
   A[AWS Launch] --> B[Cloud Native]
   B --> C[Easy Scaling]
   C --> D[Managed Services]
```

Benefits:

- Quick resource provisioning
- **Q** Auto-scaling capabilities
- **Managed services** (RDS, ELB, etc.)
- **E** Pay-as-you-use model
- Containerization Revolution

```
graph LR
   A[Heavy VMs] --> B[Lightweight Containers]
   B --> C[Docker Engine]
   C --> D[Container Orchestration Need]
```

Evolution:

- 🕱 Heavy VMs → 🍠 Lightweight containers
- **d Docker** made containerization accessible
- **@ Need** for container orchestration emerged
- **♂** Google's Solution: Borg → Kubernetes

```
graph LR
   A[Google Borg] --> B[Internal Tool]
   B --> C[Kubernetes Project]
   C --> D[CNCF Donation]
   D --> E[Open Source]
```

Timeline:

- **# Google** created **Borg** for internal use
- **2014**: Kubernetes project started (ground-up rewrite)
- ## 2014: Donated to CNCF (Cloud Native Computing Foundation)[2]

Kubernetes Architecture

Kubernetes follows a master-worker architecture with two main components:[2]

***** High-Level Architecture

```
graph TB
   subgraph "Control Plane"
      API[ API Server]
      ETCD[  etcd]
      SCHED[ Scheduler]
      CTRL[☼ Controller Manager]
      CCM[ 	■ Cloud Controller Manager]
   end
   subgraph "Worker Node 1"
      KUBELET1[  Kubelet]
      PROXY1[  Kube-proxy]
      POD1[∰ Pods]
   end
   subgraph "Worker Node 2"
      KUBELET2[  Kubelet]
      PROXY2[  Kube-proxy]
      POD2[∰ Pods]
   end
   API KUBELET1
   API KUBELET2
   API ETCD
   API SCHED
   API CTRL
   API CCM
```

Control Plane Components

The **Control Plane** manages the overall state of the cluster. It consists of:[2][3]

& API Server

```
graph LR

USER[ ♣ User] --> API[ ⑤ API Server]

API --> AUTH[ ᠬ Authentication]

AUTH --> VALID[ ✓ Validation]

VALID --> ETCD[ ☐ etcd]
```

Functions:

- **Entry point** for all administrative tasks
- di Authentication and authorization

- **Validates** API requests
- **& Communication hub** between components[3]

etcd

```
graph TB

ETCD[  etcd Key-Value Store]

ETCD --> STATE[  Cluster State]

ETCD --> CONFIG[  Configuration Data]

ETCD --> META[  Metadata]

ETCD --> OBJECTS[  Kubernetes Objects]
```

Purpose:

- 📱 Distributed key-value store
- Stores all **cluster state** information
- **Only accessible** via API Server[3]

Scheduler

```
graph LR

SCHED[⊞ Scheduler] --> FILTER[♠ Filter Nodes]

FILTER --> SCORE[ℍ Score Nodes]

SCORE --> SELECT[☑ Select Best Node]

SELECT --> ASSIGN[ஂ Assign Pod]
```

Responsibilities:

- • Assigns pods to appropriate worker nodes
- **Q Evaluates** resource requirements
- **A Load balancing** across nodes
- **Optimizes** resource utilization[2]

Controller Manager

```
graph TB

CTRL[♠ Controller Manager]

CTRL --> NODE[➡ Node Controller]

CTRL --> REP[♠ Replication Controller]

CTRL --> ENDPOINT[� Endpoint Controller]

CTRL --> SERVICE[♚ Service Account Controller]
```

Controllers Include:

• Rode Controller: Monitors node health

- Replication Controller: Manages pod replicas
- **@ Endpoint Controller**: Updates endpoint objects
- **G** Service Account Controller: Creates default service accounts[2]
- Cloud Controller Manager (CCM)

```
graph LR

CCM[		Cloud Controller Manager] --> LB[		Load Balancer]

CCM --> ROUTE[		Route Controller]

CCM --> NODE[		Route Controller]

CCM --> SERVICE[		Service Controller]
```

Purpose:

- Cloud-specific control logic
- 🌺 Manages load balancers
- Sets up **network routes**

Morker Node Components

Worker Nodes run the actual containerized applications. Each node contains:[3]

Kubelet

```
graph TB

KUBELET[ Kubelet] --> REGISTER[☑ Register Node]

KUBELET --> CREATE[፫ Create Containers]

KUBELET --> MONITOR[⑥ Monitor Pods]

KUBELET --> REPORT[ᠬ Report Status]
```

Functions:

- Registers worker node with API server
- **E** Creates/manages containers for pods
- **(a) Monitors** pod health (liveness, readiness probes)
- Reports node and pod status[4]

Kube-proxy

Responsibilities:

- **Metwork proxy** on each node
- Maintains network rules
- **Load balances** traffic to pods
- Routes requests to appropriate pods[3]

Container Runtime Interface (CRI)

```
graph TB

CRI[♥ Container Runtime Interface]

CRI --> DOCKER[♥ Docker]

CRI --> CONTAINERD[♥ containerd]

CRI --> CRIO[९ CRI-O]
```

Options:

- 👸 Docker Engine
- 🕸 containerd
- 🔧 CRI-O
- **Ø** Other OCI-compliant runtimes[4]

Pods Pods

```
graph TB

POD[∰ Pod] --> CONTAINER[❤ Container(s)]

POD --> NETWORK[∰ Shared Network]

POD --> STORAGE[∰ Shared Storage]

POD --> IP[❤ Pod IP]
```

Characteristics:

- 👸 Smallest deployable unit
- Contains one or more containers
- Bhared network and storage

Kubernetes Workflow

Deployment Process

```
sequenceDiagram

participant User as L User

participant API as API Server

participant ETCD as etcd

participant Controller as Controller

participant Scheduler as Scheduler
```

```
participant Kubelet as  Container Runtime

User->>API: 1. Deploy 2 Nginx containers

API->>API: 2. Authenticate & Validate

API->>Controller: 3. Create 2 pods

Controller->>ETCD: 4. Store desired state (2 pods)

Controller->>Scheduler: 5. Schedule pods

Scheduler->>Kubelet: 6. Assign pod to node

Kubelet->>CRI: 7. Start containers

CRI->>Kubelet: 8. Container running

Kubelet->>API: 9. Report status

API->>ETCD: 10. Update current state
```

State Reconciliation

Key Concept: Declarative Management

- **&** You define **desired state**
- Q Kubernetes continuously monitors
- **4** Automatically reconciles differences
- Self-healing capabilities

Architecture Diagrams

E Complete Kubernetes Architecture

```
graph TB

subgraph "ಔ Control Plane (Master Node)"

API[ಔ kube-apiserver• Authentication• API Gateway• Communication Hub]

ETCD[ಔ etcd• Key-Value Store• Cluster State• Configuration Data]

SCHED[௵ kube-scheduler• Pod Scheduling• Resource Optimization• Node

Selection]

CTRL[戊 kube-controller-manager• Node Controller• Replication Controller•

Endpoint Controller]

CCM[♠ cloud-controller-manager• Load Balancer• Route Controller• Cloud

Integration]

end

subgraph "☒ Worker Node 1"

KUBELETI[※ kubelet• Pod Management• Node Registration• Health
```

```
Monitoring]
        PROXY1[  kube-proxy• Network Rules• Load Balancing• Traffic Routing]
        CRI1[♥ Container Runtime• Docker/containerd• Container Lifecycle• Image
Management]
        subgraph "👸 Pods"
            POD1A[ Pod 1nginx]
            POD1B[ Pod 2nginx]
        end
    end
    subgraph " A Worker Node 2"
        KUBELET2[ www kubelet• Pod Management• Node Registration• Health
Monitoring]
        PROXY2[  kube-proxy• Network Rules• Load Balancing• Traffic Routing]
        CRI2[ ♥ Container Runtime• Docker/containerd• Container Lifecycle• Image
Management]
        subgraph "ຕື່ Pods"
            POD2A[ Pod 3nginx]
        end
    end
    API ETCD
    API SCHED
    API CTRL
    API CCM
    API KUBELET1
    API KUBELET2
    KUBELET1 CRI1
    KUBELET2 CRI2
    CRI1 --> POD1A
    CRI1 --> POD1B
    CRI2 --> POD2A
```

Pod Lifecycle Flow

```
graph LR

subgraph " User Request"

USER[  User: Deploy 5 pods]

end

subgraph " Control Plane Processing"

API[ API Server]

ETCD[ etcd Store]

CTRL[ Controller]

SCHED[ Scheduler]

end

subgraph " Worker Nodes"

NODE1[ Node 12 pods]
```

```
NODE2[ Node 22 pods]
NODE3[ Node 31 pod]
end

USER --> API
API --> CTRL
CTRL --> ETCD
CTRL --> SCHED
SCHED --> NODE1
SCHED --> NODE2
SCHED --> NODE3
```

Metworking Architecture

```
graph TB
   subgraph "  External Traffic"
       LB[∰ Load Balancer]
       end
   subgraph " Services Layer"
       SVC1[ Service 1nginx-service]
       SVC2[  Service 2api-service]
   end
   subgraph "∰ Pod Network"
       POD1[  nginx-pod-110.244.1.2]
       POD2[  nginx-pod-210.244.2.3]
       POD3[  api-pod-110.244.1.4]
   end
   LB --> INGRESS
   INGRESS --> SVC1
   INGRESS --> SVC2
   SVC1 --> POD1
   SVC1 --> POD2
   SVC2 --> POD3
```

Key Benefits

| Feature | Description | ♂ Benefit |
|----------------|--|--------------------------------|
| Cloud Agnostic | Works on any infrastructure | No vendor lock-in |
| | Scales based on demand | S Cost optimization |
| Self-healing | Automatically replaces failed containers | lacktriangle High availability |

| Feature | Description | & Benefit |
|------------------|---------------------------------|---------------------------|
| ♣ Load Balancing | Distributes traffic efficiently | ⊗ Better performance |
| ි Security | Built-in security features | ● Enterprise-ready |
| Rolling Updates | Zero-downtime deployments | © Continuous delivery |

& Use Cases

```
mindmap
 root((  KubernetesUse Cases))
   Web Applications
     Frontend Apps
     🖺 Database Clusters
   Microservices
     ♦ Service Mesh
     Inter-service Communication
     Service Discovery
   ■ Data Processing

← ML Workloads

    □ Big Data Analytics

     Batch Processing
   Enterprise
     Multi-cloud Strategy
     Compliance Requirements
```

% Ecosystem Integration

```
graph TB
   K8S[  Kubernetes Core]
   subgraph "♦ Container Tools"
       DOCKER[∰ Docker]
       CONTAINERD[♥ containerd]
       PODMAN[ % Podman]
   end
   subgraph "  Networking"
       CALICO[ ⊕ Calico]
       FLANNEL[  Flannel]
       WEAVE[  Weave Net]
   end
   subgraph "🖺 Storage"
      CEPH[ Ceph]
       NFS[ NFS]
```

```
end
subgraph "@ Monitoring"
    PROMETHEUS[ Prometheus]
    GRAFANA[₩ Grafana]
    JAEGER[ ♥ Jaeger]
end
K8S --> DOCKER
K8S --> CONTAINERD
K8S --> PODMAN
K8S --> CALICO
K8S --> FLANNEL
K8S --> WEAVE
K8S --> CEPH
K8S --> NFS
K8S --> CLOUD
K8S --> PROMETHEUS
K8S --> GRAFANA
K8S --> JAEGER
```

Summary

Kubernetes has revolutionized container orchestration by providing:

Key Takeaways

- Centralized control through the Control Plane
- Distributed execution via Worker Nodes
- **Declarative management** with desired state reconciliation
- Cloud-agnostic architecture preventing vendor lock-in
- Enterprise-grade features for production workloads

Getting Started

- 1. **Learn** container fundamentals (Docker)
- 2. **<u>E</u> Understand** Kubernetes architecture
- 3. **Practice** with local clusters (minikube, kind)
- 4. **Deploy** to cloud providers (EKS, GKE, AKS)
- 5. **Master** kubectl and YAML manifests

Kubernetes continues to be the **de facto standard** for container orchestration, enabling organizations to build scalable, resilient, and portable applications in the cloud-native era.[2][3]

[1] https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/62888276/18255543-3118-4d7e-a119-a31107f18490/paste.txt [2] https://kubernetes.io/docs/concepts/architecture/ [3] https://www.simform.com/blog/kubernetes-architecture/ [4] https://devopscube.com/kubernetes-architecture-explained/ [5] https://app.eraser.io/workspace/2GyNfNQs6bEl7WFlNzel [6] https://app.eraser.io/workspace/2G [7] https://k21academy.com/docker-kubernetes/kubernetes-architecture-components-overview-for-beginners/ [8] https://creately.com/guides/kubernetes-architecture-diagram/ [9]

https://spot.io/resources/kubernetes-architecture/11-core-components-explained/ [10]

https://dev.to/vaibhav_ca0da2b8bef9b07c2/kubernetes-architecture-workernode-2ohk [11]

https://www.clickittech.com/devops/kubernetes-architecture-diagram/ [12]

https://www.armosec.io/glossary/kubernetes-control-plane/ [13]

https://layer5.io/resources/kubernetes/kubernetes-architecture-101 [14]

https://cloudairy.com/blog/understanding-kubernetes-architecture-diagrams-and-components/ [15]

https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture [16]

https://cloudairy.com/template/kubernetes-architecture-diagram/ [17]

https://kubernetes.io/docs/contribute/style/diagram-guide/ [18] https://spacelift.io/blog/kubernetes-

architecture [19] https://kubernetes.io/docs/concepts/architecture/nodes/ [20]

https://www.eraser.io/ai/kubernetes-diagram-generator [21] https://dev.to/monarene/inside-the-kubernetes-

control-plane-28ie [22] https://geekflare.com/devops/kubernetes-architecture/ [23]

https://kubernetes.io/docs/concepts/overview/components/