**Kubernetes Architecture - Study Notes (Day 31)**

**Course: Free DevOps Course**  
**Instructor: Abhishek**

**1. Introduction**

**Why "K8s"?**

* **Kubernetes** → **"K8s"** (Count the letters between 'K' and 's': **8 letters**).
* Example:
  + "K" + **8 letters** + "s" = **K8s**.

**Recap: Kubernetes vs. Docker**

| **Feature** | **Docker** | **Kubernetes (K8s)** |
| --- | --- | --- |
| **Scope** | Single Host | Multi-Host Cluster |
| **Auto-Healing** | ❌ No | ✅ Yes (ReplicaSets) |
| **Auto-Scaling** | ❌ Manual | ✅ Yes (HPA) |
| **Enterprise Support** | ❌ Limited | ✅ Extensive (Load Balancers, etc.) |

**2. Kubernetes Architecture Overview**

Kubernetes is divided into **two planes**:

1. **Control Plane (Master Node)** – Manages the cluster.
2. **Data Plane (Worker Nodes)** – Runs applications.

**3. Control Plane Components**

**1. API Server**

* **Role:** Heart of Kubernetes, handles all requests (REST API).
* **Example:** When you run kubectl create pod, the request goes to the API Server.

**2. Scheduler**

* **Role:** Decides **which worker node** runs a pod.
* **Example:** Assigns a pod to a node with **available CPU/memory**.

**3. etcd**

* **Role:** Key-value store for **cluster state** (backup & recovery).
* **Example:** Stores pod deployment history, node status.

**4. Controller Manager**

* **Role:** Ensures desired state (e.g., ReplicaSet maintains pod count).
* **Example:** If a pod crashes, it triggers auto-healing.

**5. Cloud Controller Manager (CCM)**

* **Role:** Links Kubernetes to **cloud providers** (AWS, GCP, Azure).
* **Example:** Creates a **LoadBalancer** service in AWS.

**4. Data Plane (Worker Node) Components**

**1. Kubelet**

* **Role:** Ensures pods are running.
* **Example:** Reports pod status to the API Server.

**2. Kube-Proxy**

* **Role:** Manages **networking rules** (IP tables/IPVS).
* **Example:** Load balances traffic between pods.

**3. Container Runtime**

* **Role:** Runs containers (Docker, containerd, CRI-O).
* **Example:** Executes nginx container inside a pod.

**5. How Kubernetes Works (Step-by-Step)**

**Step 1: User Requests a Pod**

* Command: kubectl create -f pod.yaml → Goes to **API Server**.

**Step 2: Scheduler Assigns Pod to Node**

* Checks **etcd** for node resources → Assigns pod to **Worker Node 1**.

**Step 3: Kubelet Creates the Pod**

* **Kubelet** on Worker Node 1 pulls image via **Container Runtime**.

**Step 4: Kube-Proxy Sets Up Networking**

* Assigns **IP** and load balancing rules.

**Step 5: Controller Manager Monitors State**

* If pod crashes → **ReplicaSet** creates a new one.

**6. Key Comparisons**

| **Component** | **Docker Equivalent** | **Kubernetes Equivalent** |
| --- | --- | --- |
| **Orchestration** | ❌ None | ✅ Control Plane |
| **Networking** | Docker0 Bridge | Kube-Proxy |
| **Runtime** | Docker Shim | Container Runtime (CRI) |
| **Scaling** | ❌ Manual | ✅ HPA |

**7. Practical Assignment**

✅ **Task:** Draw Kubernetes architecture & post on LinkedIn/GitHub.

* **Include:**
  + Control Plane & Data Plane components.
  + How kubectl create pod flows through components.

📌 **Example Post:**

\_"Today I learned Kubernetes architecture! 🚀

* Control Plane: API Server, Scheduler, etcd
* Worker Nodes: Kubelet, Kube-Proxy, Container Runtime  
  Check my GitHub for a detailed diagram: [Link]"\_

**8. Summary**

* **Control Plane** = Brain (API Server, Scheduler, etcd).
* **Data Plane** = Muscle (Kubelet, Kube-Proxy, Containers).
* **K8s > Docker** due to **auto-scaling, healing & multi-node support**.

🚀 **Next Lesson:** Kubernetes Pods Deep Dive!

📢 **Feedback?** Comment below! 👍 **Like & Share** if this helped!

**End of Notes** 🎉

# \*\*Kubernetes Architecture - Study Notes (Day 31)\*\*

\*\*Course: Free DevOps Course\*\*

\*\*Instructor: Abhishek\*\*

---

## \*\*1. Introduction\*\*

### \*\*Why "K8s"?\*\*

- \*\*Kubernetes\*\* → \*\*"K8s"\*\* (Count the letters between 'K' and 's': \*\*8 letters\*\*).

- Example:

- "K" + \*\*8 letters\*\* + "s" = \*\*K8s\*\*.

### \*\*Recap: Kubernetes vs. Docker\*\*

| \*\*Feature\*\* | \*\*Docker\*\* | \*\*Kubernetes (K8s)\*\* |

|---------------------------|-------------------------------------|-------------------------------------|

| \*\*Scope\*\* | Single Host | Multi-Host Cluster |

| \*\*Auto-Healing\*\* | ❌ No | ✅ Yes (ReplicaSets) |

| \*\*Auto-Scaling\*\* | ❌ Manual | ✅ Yes (HPA) |

| \*\*Enterprise Support\*\* | ❌ Limited | ✅ Extensive (Load Balancers, etc.) |

---

## \*\*2. Kubernetes Architecture Overview\*\*

Kubernetes is divided into \*\*two planes\*\*:

1. \*\*Control Plane (Master Node)\*\* – Manages the cluster.

2. \*\*Data Plane (Worker Nodes)\*\* – Runs applications.

![Kubernetes Architecture](https://d33wubrfki0l68.cloudfront.net/2475489eaf20163ec0f54ddc1d92aa8d4c87c96b/e7c81/images/docs/components-of-kubernetes.svg)

---

## \*\*3. Control Plane Components\*\*

### \*\*1. API Server\*\*

- \*\*Role:\*\* Heart of Kubernetes, handles all requests (REST API).

- \*\*Example:\*\* When you run `kubectl create pod`, the request goes to the API Server.

### \*\*2. Scheduler\*\*

- \*\*Role:\*\* Decides \*\*which worker node\*\* runs a pod.

- \*\*Example:\*\* Assigns a pod to a node with \*\*available CPU/memory\*\*.

### \*\*3. etcd\*\*

- \*\*Role:\*\* Key-value store for \*\*cluster state\*\* (backup & recovery).

- \*\*Example:\*\* Stores pod deployment history, node status.

### \*\*4. Controller Manager\*\*

- \*\*Role:\*\* Ensures desired state (e.g., ReplicaSet maintains pod count).

- \*\*Example:\*\* If a pod crashes, it triggers auto-healing.

### \*\*5. Cloud Controller Manager (CCM)\*\*

- \*\*Role:\*\* Links Kubernetes to \*\*cloud providers\*\* (AWS, GCP, Azure).

- \*\*Example:\*\* Creates a \*\*LoadBalancer\*\* service in AWS.

---

## \*\*4. Data Plane (Worker Node) Components\*\*

### \*\*1. Kubelet\*\*

- \*\*Role:\*\* Ensures pods are running.

- \*\*Example:\*\* Reports pod status to the API Server.

### \*\*2. Kube-Proxy\*\*

- \*\*Role:\*\* Manages \*\*networking rules\*\* (IP tables/IPVS).

- \*\*Example:\*\* Load balances traffic between pods.

### \*\*3. Container Runtime\*\*

- \*\*Role:\*\* Runs containers (Docker, containerd, CRI-O).

- \*\*Example:\*\* Executes `nginx` container inside a pod.

---

## \*\*5. How Kubernetes Works (Step-by-Step)\*\*

### \*\*Step 1: User Requests a Pod\*\*

- Command: `kubectl create -f pod.yaml` → Goes to \*\*API Server\*\*.

### \*\*Step 2: Scheduler Assigns Pod to Node\*\*

- Checks \*\*etcd\*\* for node resources → Assigns pod to \*\*Worker Node 1\*\*.

### \*\*Step 3: Kubelet Creates the Pod\*\*

- \*\*Kubelet\*\* on Worker Node 1 pulls image via \*\*Container Runtime\*\*.

### \*\*Step 4: Kube-Proxy Sets Up Networking\*\*

- Assigns \*\*IP\*\* and load balancing rules.

### \*\*Step 5: Controller Manager Monitors State\*\*

- If pod crashes → \*\*ReplicaSet\*\* creates a new one.

---

## \*\*6. Key Comparisons\*\*

| \*\*Component\*\* | \*\*Docker Equivalent\*\* | \*\*Kubernetes Equivalent\*\* |

|-----------------------|-----------------------|---------------------------|

| \*\*Orchestration\*\* | ❌ None | ✅ Control Plane |

| \*\*Networking\*\* | Docker0 Bridge | Kube-Proxy |

| \*\*Runtime\*\* | Docker Shim | Container Runtime (CRI) |

| \*\*Scaling\*\* | ❌ Manual | ✅ HPA |

---

## \*\*7. Practical Assignment\*\*

✅ \*\*Task:\*\* Draw Kubernetes architecture & post on LinkedIn/GitHub.

- \*\*Include:\*\*

- Control Plane & Data Plane components.

- How `kubectl create pod` flows through components.

📌 \*\*Example Post:\*\*

> \_"Today I learned Kubernetes architecture! 🚀

> - Control Plane: API Server, Scheduler, etcd

> - Worker Nodes: Kubelet, Kube-Proxy, Container Runtime

> Check my GitHub for a detailed diagram: [Link]"\_

---

## \*\*8. Summary\*\*

- \*\*Control Plane\*\* = Brain (API Server, Scheduler, etcd).

- \*\*Data Plane\*\* = Muscle (Kubelet, Kube-Proxy, Containers).

- \*\*K8s > Docker\*\* due to \*\*auto-scaling, healing & multi-node support\*\*.

🚀 \*\*Next Lesson:\*\* Kubernetes Pods Deep Dive!

📢 \*\*Feedback?\*\* Comment below! 👍 \*\*Like & Share\*\* if this helped!

---

\*\*End of Notes\*\* 🎉