





# Introduction to Kubernetes

**What is Kubernetes?** Kubernetes is an open-source container **orchestration** platform that automates deploying, scaling, and managing applications in containers.

**History:** Kubernetes was initially developed by Google to solve the challenges of managing containerized applications at scale. Google had an internal system called **Borg** that handled container orchestration. Project 7  When a group of engineers started to work on open-sourcing a more streamlined, scalable orchestration platform based on their learnings from Borg, they code-named it "Project 7. Project 7 evolved into a robust and open-source-ready platform; it was officially named "Kubernetes.

## Why is it Essential in Modern DevOps?

Containers are a good and easy way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. In docker we used docker swarm for this. but any how docker has drawbacks

- 1.) **Efficient Resource Usage** : Kubernetes optimizes resource use, helping teams save on hardware and cloud costs.
- 2.) **Environment Consistency:** It provides a consistent platform for development, testing, and production, ensuring smooth deployments.
- 3.) **Scalability** : Automatically scales applications up or down to handle load changes efficiently.
- 4.) **Self-Healing:** Detects and replaces failed containers, maintaining uptime.
- 5.) **Security & Compliance** : Kubernetes offers built-in security features, like Role-Based Access Control (RBAC) and Secrets management.

**Cluster:** A Kubernetes cluster is a collection of nodes (machines) where Kubernetes manages and orchestrates containerized applications. A cluster represents a Kubernetes deployment as a whole, combining both worker nodes (where applications run) and a control plane (which manages and monitors the cluster).

## **Problems with docker**

### **1. Single Host Limitation**

Docker runs containers on a single host unless orchestrated otherwise. If you run 100 containers on one host and one of them starts consuming too much memory or CPU, it can affect other containers — for example, the 50th or 99th — because all containers share the same underlying host resources. There's no isolation or prioritization unless you manually configure resource limits. If a container doesn't get the necessary resources, it may crash and restart repeatedly, impacting overall stability.

### **2. Lack of Native Auto-Healing**

If someone manually stops or kills a container, Docker itself doesn't have built-in auto-healing mechanisms. This means the application running inside that container becomes unavailable unless someone manually restarts it or a custom script is in place. Docker by itself does not ensure service continuity in the event of container failure.

### **3. No Auto-Scaling**

Docker does not provide automatic scaling. Suppose your application is deployed on a host with 4 CPUs and 4 GB RAM and can handle 10,000 users. During peak times, like a festival season, if traffic surges to 100,000 users, the host will be overwhelmed. Without the ability to scale out dynamically across more resources or machines, the application will crash due to overload.

### **4. Not Enterprise-Ready Out-of-the-Box**

Docker is lightweight and ideal for simple workloads. However, it lacks several features expected in enterprise environments, such as:

- Auto-scaling and self-healing
- Native support for load balancing
- Security features like firewall rules
- Service discovery and API gateways

To address these gaps, you need a more comprehensive platform — and that's where Kubernetes comes in.

---

## How Kubernetes Solves These Problems

1. **Clustering by Default** Kubernetes organizes infrastructure into a cluster, which is a group of nodes (physical or virtual machines). These nodes share resources and workloads can be distributed across them, solving the single-host limitation.

2. **Resource Management** Kubernetes allows you to define resource limits and requests per container (CPU/memory), ensuring fair allocation and preventing any single container from starving others.

3. **Auto-Healing** Kubernetes constantly monitors the health of pods (which run containers). If a container crashes or is killed, Kubernetes automatically restarts it. If a node fails, the workloads are rescheduled on healthy nodes — no manual intervention needed.

4. **Auto-Scaling** Kubernetes supports horizontal pod autoscaling, which automatically increases or decreases the number of pod replicas based on CPU usage or custom metrics. It can also support cluster autoscaling, which adds or removes nodes based on demand.

### 5. **Enterprise Features Kubernetes supports:**

- ❑ Built-in load balancing across pods
- ❑ Network policies (firewalls)
- ❑ Ingress controllers and API gateways
- ❑ Secrets management
- ❑ Rolling updates and rollbacks

---

## Conclusion

While Docker is excellent for building and running containerized applications, it's not enough for running them reliably at scale, especially in production environments. Kubernetes complements Docker by offering powerful features like clustering, self-healing, auto-scaling, and enterprise-grade networking and security.

In short, Docker helps you run containers; Kubernetes helps you run containers at scale.