KUBERNETES VS DOCKER SWARM: COMPARING CONTAINER ORCHESTRATION TOOLS

When deploying applications at scale, you need to plan and coordinate all your architecture components with current and future strategies in mind. Container orchestration tools help achieve this by automating the management of application microservices across multiple clusters. Two of the most popular container orchestration tools are Kubernetes and Docker Swarm.

Let's explore the major features and differences between Kubernetes and Docker Swarm in this article, so you can choose the right one for your tech stack.

(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

Kubernetes overview

Kubernetes is an open-source, cloud-native infrastructure tool that automates scaling, deployment, and management of containerized applications—apps that are in containers.

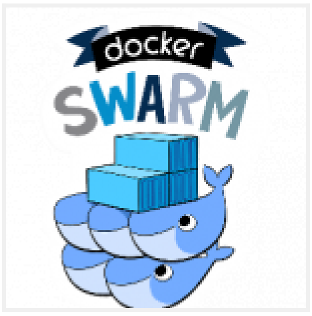
Google originally developed Kubernetes, eventually handing it over to the Cloud Native Computing Foundation (CNCF) for enhancement and maintenance. Among the top choices for developers, Kubernetes is a feature-rich container orchestration platform that benefits from:

- Regular updates by CNCF
- Daily contributions from the global community

Docker Swarm overview

Docker Swarm is native to the Docker platform. Docker was developed to maintain application efficiency and availability in different runtime environments by deploying containerized application microservices across multiple clusters.

Docker Swarm, what we're looking at in this article, is a container orchestration tool native to Docker that enables applications to run seamlessly across multiple nodes that share the same containers. In essence, you use the Docker Swarm model to efficiently manage, deploy, and scale a cluster of nodes on Docker.



Differences between Kubernetes and Docker Swarm

Kubernetes and Docker Swarm are both effective solutions for:

- Massive scale application deployment
- Implementation
- Management

Both models break applications into containers, allowing for efficient automation of application management and scaling. Here is a general summary of their differences:

- Kubernetes focuses on open-source and modular orchestration, offering an efficient container orchestration solution for high-demand applications with complex configuration.
- Docker Swarm emphasizes ease of use, making it most suitable for simple applications that are quick to deploy and easy to manage.

Now, let's look at the fundamental differences in how these cloud orchestration technologies operate. In each section, we'll look at K8s first, then Docker Swarm.

Installation

With multiple installation options, Kubernetes can easily be deployed on any platform, though it is recommended to have a basic understanding of the platform and cloud computing prior to the installation.

Installing Kubernetes requires downloading and installing **kubectl,** the Kubernetes Command Line Interface (CLI):

- On Linux, you can install **kubectl** using **curl**, native or other package management procedure as a snap application.
- On MacOS, **kubectl** can be installed using **curl**, Homebrew, or MacPorts.
- On Windows, you can install **kubectl** using multiple options, including **curl** , Powershell Gallery package manager, Chocolatey package manager, or Scoop command-line installer.

Detailed steps on **kubectl** installation can be found here.

Compared to Kubernetes, installing Docker Swarm is relatively simple. Once the Docker Engine is installed in a machine, deploying a Docker Swarm is as easy as:

- Assigning IP addresses to hosts
- Opening the protocols and ports between them

Before initializing Swarm, first assign a **manager node** and one or multiple **worker nodes** between the hosts.

Graphical user interface (GUI)

Kubernetes features an easy Web User Interface (dashboard) that helps you:

- Deploy containerized applications on a cluster
- Manage cluster resources
- View an error log and information on the state of cluster resources (including Deployments, Jobs, and DaemonSets) for efficient troubleshooting

Unlike Kubernetes, Docker Swarm does not come with a Web UI out-of-the-box to deploy applications and orchestrate containers. However, with its growing popularity, there are now several third-party tools that offer simple to feature-rich GUIs for Docker Swarm. Some prominent Docker Swarm UI tools are:

- Portainer
- Dockstation
- Swarmpit
- Shipyard

Application definition & deployment

A Kubernetes deployment involves describing declarative updates to application states while updating Kubernetes **Pods** and **ReplicaSets**. By describing a Pod's desired state, a controller changes the *current* state to the *desired* one at a regulated rate. With Kubernetes deployments, you can define all aspects of an application's lifecycle. These aspects include:

- The number of pods
- Images to use
- How pods should be updated

In Docker Swarm, you deploy and define applications using predefined Swarm files to declare the *desired state* for the application. To deploy the app, you just need to copy the YAML file at the root level. This file, also known as the **Docker Compose File**, allows you to leverage its multiple node machine capabilities, thereby allowing organizations to run containers and services on:

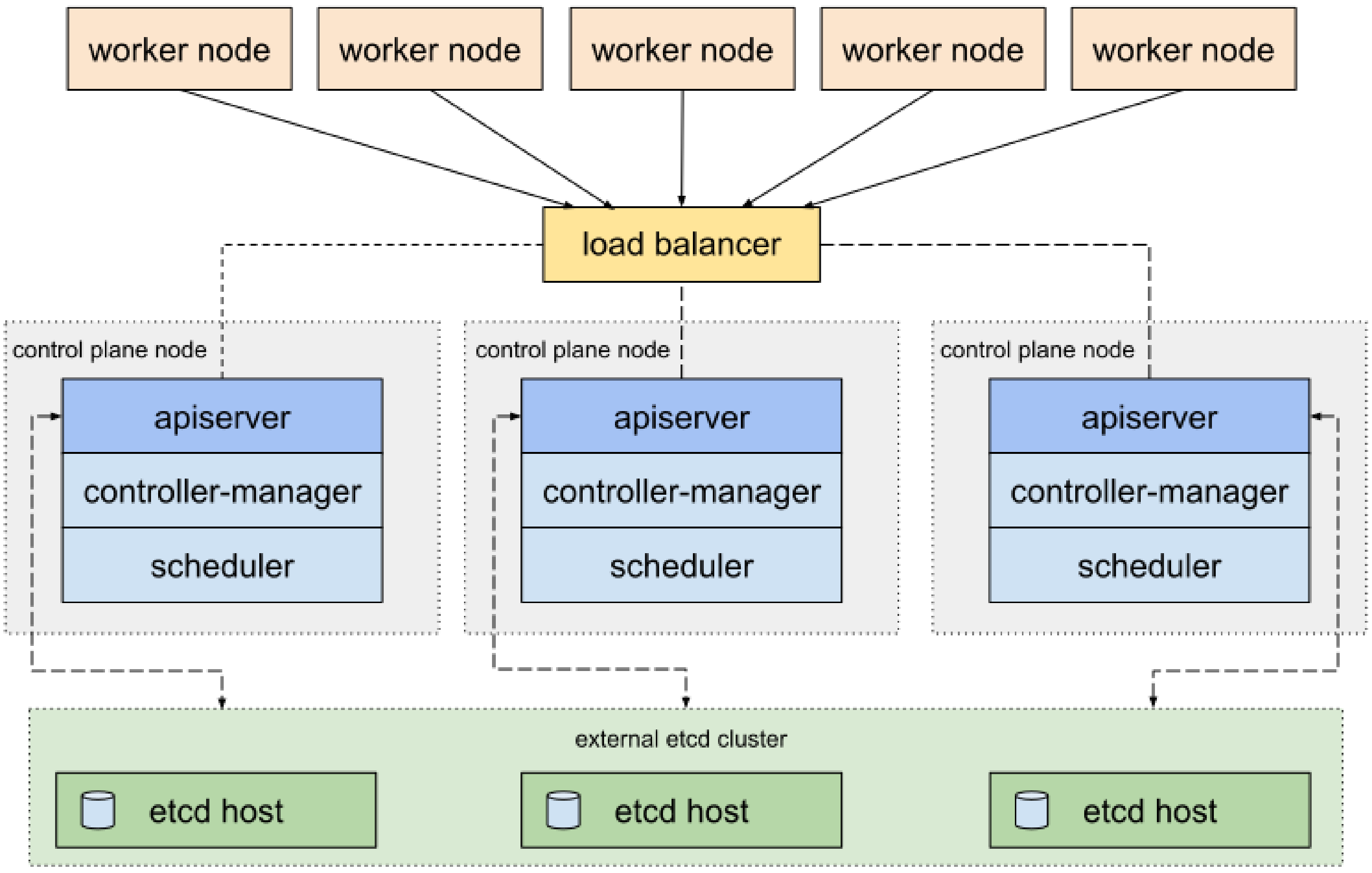
- Multiple machines
- Any number of networks

Availability

Kubernetes allows two topologies by default. These ensure high availability by creating clusters to eliminate single point of failures.

- You can use **Stacked Control Plane** nodes that ensure availability by co-locating **etcd** objects with all available nodes of a cluster during a failover.
- Or, you can use external **etcd** objects for load balancing, while controlling the control plane nodes separately**.**

Notably, both methods leverage using **kubeadm** and use a **Multi-Master** approach to maintain high availability, by maintaining **etcd** cluster nodes either externally or internally within a control plane.



External etcd topology (Image source)

To maintain high-availability, Docker uses service replication at the Swarm Nodes level. By doing so, a Swarm Manager deploys multiple instances of the same container, with replicas of services in each. By default, an **Internal Distributed State Store:**

- Controls the Swarm Manager nodes to manage an entire cluster
- Administers worker node resources to form highly available, load-balanced container instances

Scalability

Kubernetes supports autoscaling on both:

- The cluster level, through **Cluster Autoscaling**
- The pod level, with **Horizontal Pod Autoscaler**

At its core, Kubernetes acts as an *all-inclusive* network for distributed nodes and provides strong guarantees in terms of unified API sets and cluster states. Scaling in Kubernetes fundamentally involves creating new pods and scheduling it to nodes with available resources.

Docker Swarm deploys containers quicker. This gives the orchestration tool faster reaction times that allow for on-demand scaling. Scaling a Docker application to handle high traffic loads involves replicating the number of connections to the application. You can, therefore, easily scale your application up and down for even higher availability.

Networking

Kubernetes creates a flat, peer-to-peer connection between pods and node agents for efficient inter-cluster networking. This connection includes network policies that regulate communication between pods while assigning distinct IP addresses to each of them. To define subnet, the Kubernetes networking model requires two **Classless Inter-Domain Routers (CIDRs)**:

- One for Node IP Addressing
- The other for services

Docker Swarm creates two types of networks for every node that joins a Swarm:

- One network type outlines an overlay of all services within the network.
- The other creates a host-only bridge for all containers.

With a multi-layered overlay network, a peer-to-peer distribution among all hosts is achieved that enables secure and encrypted communications.

Monitoring

Kubernetes offers multiple native logging and monitoring solutions for deployed services within a cluster. These solutions monitor application performance by:

- Inspecting services, pods, and containers
- Observing the behavior of an entire cluster

Additionally, Kubernetes also supports third-party integration to help with event-based monitoring including:

- ElasticSearch/Kibana
- InfluxDB
- Grafana
- Sysdig

Unlike Kubernetes, Docker Swarm does not offer a monitoring solution out-of-the-box. As a result, you have to rely on third-party applications to support monitoring of Docker Swarm. Typically, monitoring a Docker Swarm is considered to be more complex due to its sheer volume of cross-node objects and services, relative to a K8s cluster.

These are a few open-source monitoring tools that collectively help achieve a scalable monitoring solution for Docker Swarm:

- InfluxDB
- Grafana
- cAdvisor

Closing thoughts

The greater purpose of Kubernetes and Docker Swarm do overlap each other. But, as we've outlined, there are fundamental differences between how these two operate. At the end of the day, both options solve advanced challenges to make your digital transformation realistic and efficient.

Additional resources

For related reading, explore these resources:

- BMC DevOps Blog
- Container Management Platforms: Which Are Most Popular?
- Kubernetes Guide, a series of tutorials and articles
- How To Introduce Docker Containers in Enterprise
- Managing Containers & Code for DevOps
- Containers Aren't Always the Solution