KUBERNETES VS DOCKER: A QUICK COMPARISON

Kubernetes and Docker are different technologies that may work separately—but they're best paired to facilitate high scalability and availability in containerized applications.

While Docker specifically manages containers on individual nodes, Kubernetes helps you automate tasks like load balancing, scaling, container provisioning, and networking across several hosts within a cluster.

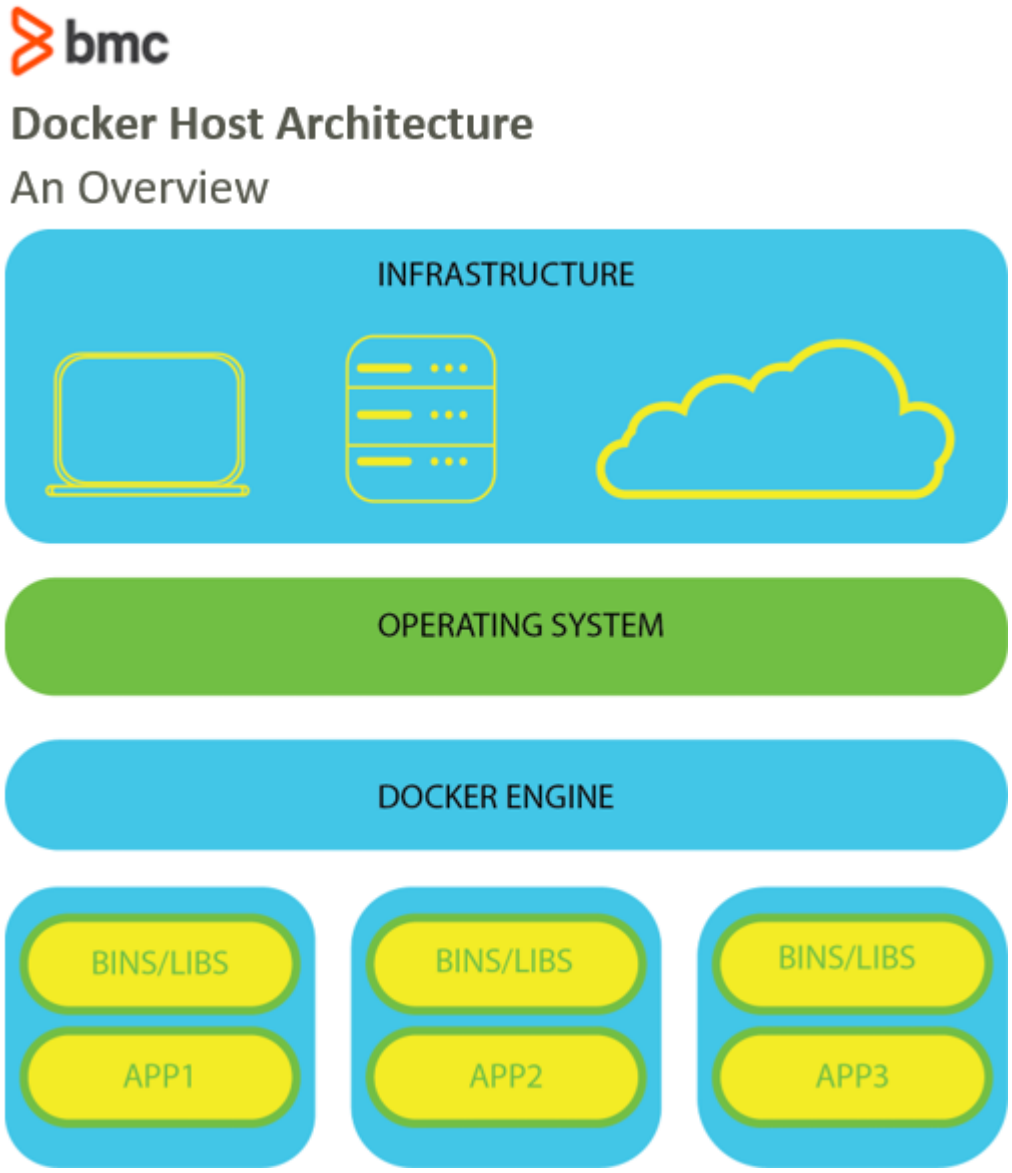
Increasing organizational best practices patterns also suggests integrating Kubernetes and Docker to create an isolation mechanism that lets you augment container resources more efficiently. With these constructs, developers can collaborate on complex projects without having to replicate the entire application in their respective IDEs.

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

Docker overview

Docker is an open-source containerization platform that simplifies the deployment of applications on any computing infrastructure. While there is a host of other containerized technologies worldwide, Docker continues to be the most popular Platform as a Service for application build and deployment.

Through a text file format (dockerfile), Docker lets you package applications as self-sufficient, portable components that you can easily deploy on-premises or on the cloud. Docker's runtime environment, the **Docker Engine**, allows developers to build applications on any machine and share images through a registry for faster deployments.

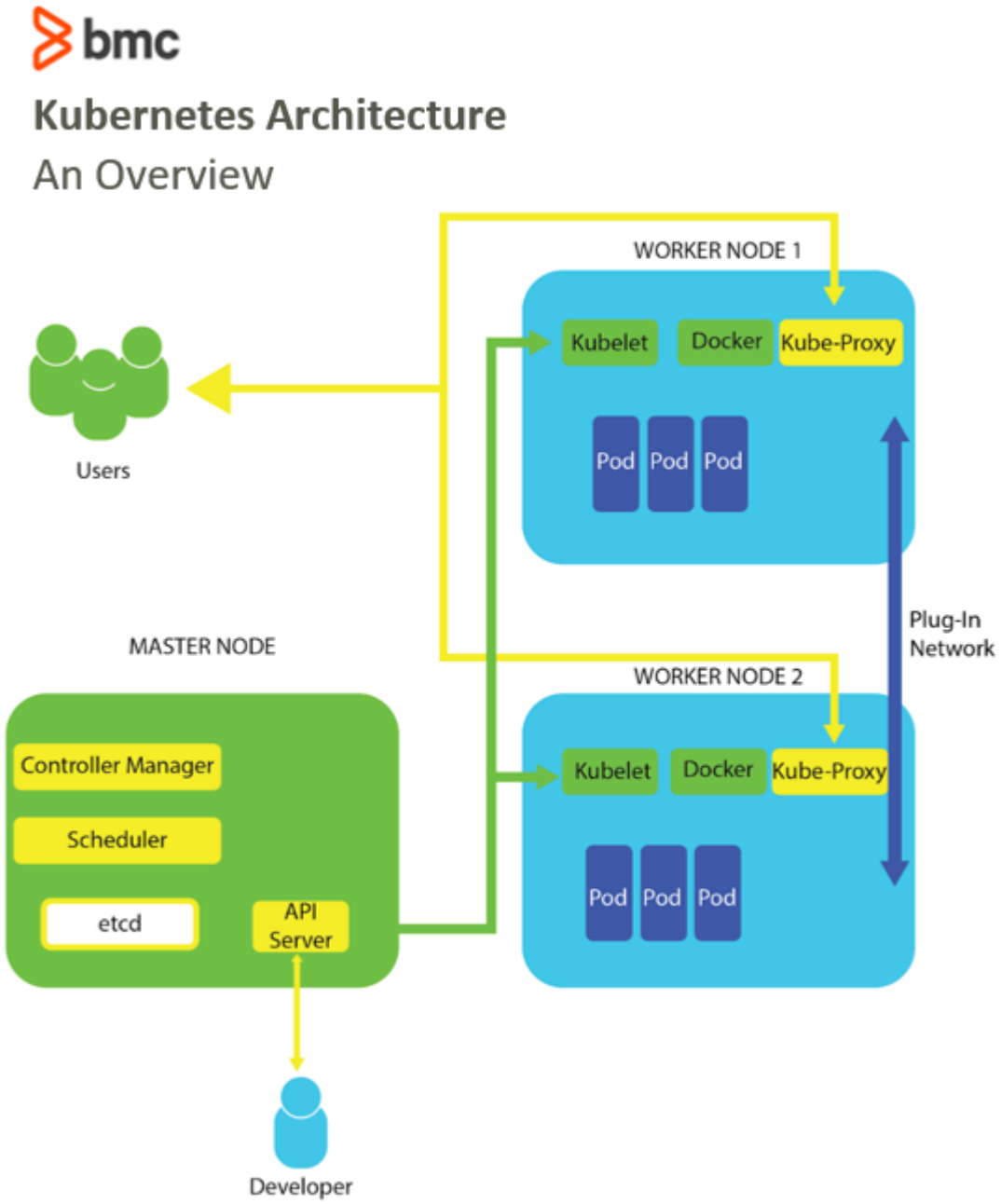


Kubernetes overview

As applications scale up in size, they require multiple containers hosted on distributed servers. When many distributed containers make operating the application tedious and complicated, Kubernetes forms a framework that efficiently controls how containers run.

Besides allowing your containers to run, Kubernetes also solves issues that arise when scaling several distributed containers by orchestrating a cluster of virtual machines (VMs) and creates a schedule for running containers on each VM. To manage container schedules and computing resources, Kubernetes:

- Provides an application programming interface (API)
- Maintains high availability through advanced autoscaling and automatic load balancing



Comparing Kubernetes & Docker

With today's advanced technologies, Docker and Kubernetes solve different problems arising from diverse backgrounds of application development.

An organization may use Kubernetes with other container runtimes interfaces, such as CRI-O and RunC, and at the same time use Docker with other orchestration tools for communication between multiple nodes by forming a network of containerized applications.

Let's take a look at how these two technologies compare.

Similarities

Both Docker and Kubernetes work toward creating high application availability—but they use different approaches to do so:

- Kubernetes replicates pods across all nodes within a cluster and avoids downtime by

detecting and acting on failed nodes.
- Docker distributes nodes across the Swarm, managing and replicating resources at scale to ensure high availability.

Kubernetes and Docker are both open-source frameworks, with large, globally distributed communities for support and consultation:

- Kubernetes gets unlimited support from the three prominent cloud service providers, Azure, GCP, and AWS, and also from open-source communities.
- Docker benefits from a growing base of active community users who regularly contribute with updates and plugins that improve the framework's functionality.

Differences

Docker provides a framework for packaging and distributing applications as microservices in containers, while Kubernetes handles these containers' management, control, and coordination. Since this makes the two technologies fundamentally incomparable, it makes more sense to differentiate between Kubernetes and Docker Swarm, Docker's native cloud orchestration tool.

Docker Swarm comes with its own API and perfectly fits into the Docker ecosystem. Any organization that uses Docker for containerization also uses Docker Swarm to make transitioning and provisioning relatively easy. This tight integration and simplicity also make Docker Swarm the go-to orchestration tool for light development needs.

But, this changes when you're dealing with complex-scale application provisioning. In this case, Kubernetes replaces Swarm as the trusted orchestration tool. That's because Kubernetes:

- Comes with a rich collection of tools and an interactive user interface, making it perfect for complex workflows and large production environments.
- Offers the flexibility to extend and customize it to handle any workload, enhancing auto-scaling and system monitoring.

A Docker Swarm is used to run on a single node, while Kubernetes runs across a distributed cluster. This essentially means that Docker Swarm can only share data volumes between different containers sharing a pod, while Kubernetes shares storage volumes between all containers within a cluster.

The Kubernetes & Docker synergy

Instead of thinking about K8s versus Docker, you'll get much better scaling and stability when you use them together.

While Kubernetes integrates with most other container runtimes, it integrates seamlessly with Docker. There are plenty of Docker-centric tools that convert Docker settings to be used in Kubernetes.

On the same lines, Docker embraces the Kubernetes integration through an Enterprise Kubernetes Distribution. An integration between the two improves the architecture of microservices, facilitates quicker production times, and coordinates containerized applications.

Using containers

Containers help organizations create portable, scalable, and responsive applications that can be deployed on any computing infrastructure. While Docker's use-cases are inclined towards creating individual containers, Kubernetes is utilized to manage multiple containers during runtime. Running Docker Containers with Kubernetes allows organizations to leverage DevOps best practices by coordinating and orchestrating containerized applications efficiently.

Related reading

- BMC DevOps Blog
- Kubernetes Guide, a series of tutorials
- State of Containers in 2020
- Containers Aren't Always the Solution