HOW CONTAINERS & KUBERNETES WORK TOGETHER



Application containerization has become the norm in the IT industry. It allows organizations or developers to encapsulate the application code and all its dependencies in a portable environment so that the app can be deployed virtually anywhere.

Kubernetes is a powerful, open-source orchestration service that can be used to manage containers in a containerized application.

In this article, we will dig deeper into both technologies and see how they complement each other.

How containers work

Before learning about containers, we first need to know why containers were designed.

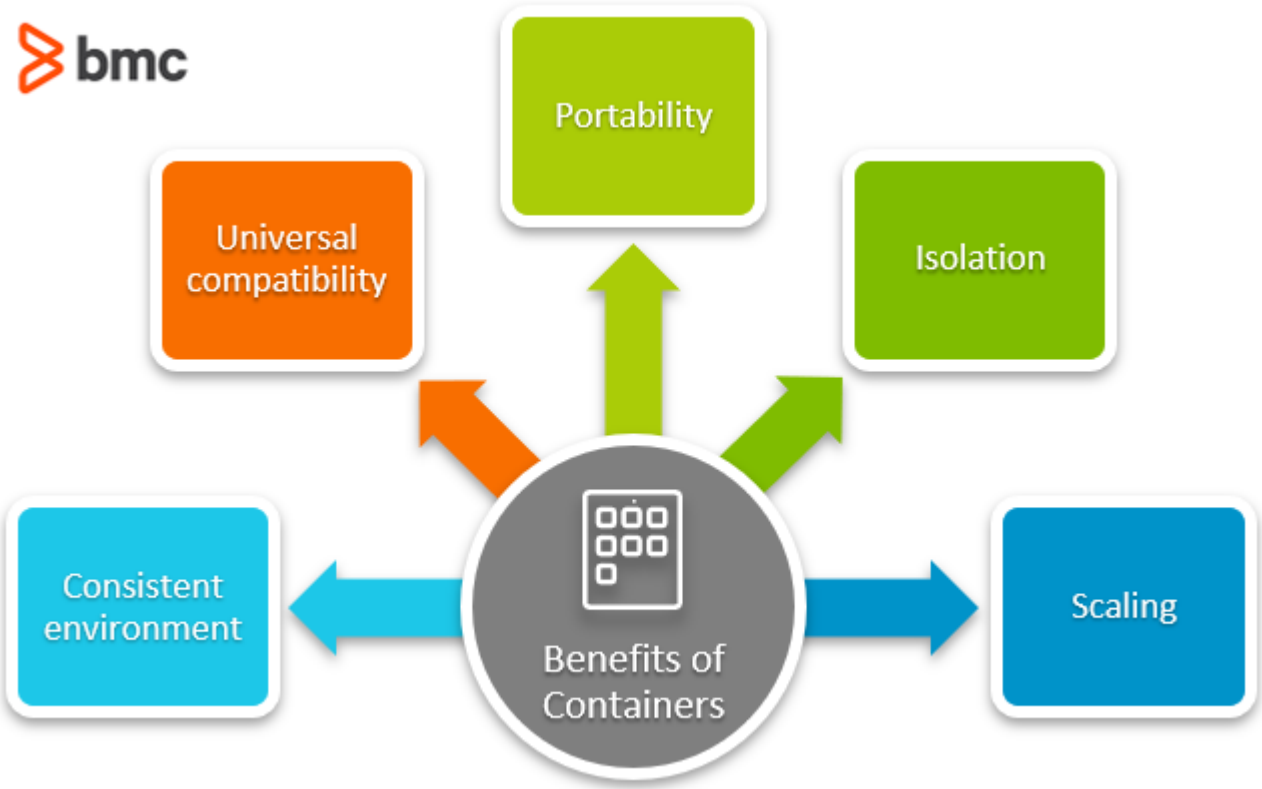
In traditional app development, once an application or a service is developed, the next step is to deploy it, which can be a complex and time-consuming process. That's because a successful app deployment requires a myriad of careful configurations, from resource allocation and dependency management to setting up environmental variables.

Another consideration is the portability of the application. If the developer needs to change the deployment environment or move to a new cloud provider, they need to go through the whole deployment process again from the ground up. Again, this is a tedious and time-consuming process that may lead to:

- Config drifts
- Production bugs

Containers successfully address this issue by bundling the application code base with all the dependencies—such as runtime, system tools, configuration files, and libraries—into a package called the container. This containerization creates a lightweight, secure, and immutable package known as the container image.

This image can then be used to deploy a containerized application in any environment, offering developers the freedom to develop an application without worrying about environmental constraints.



Advantages of containers

Developers like containers for a variety of reasons:

- **Consistent environment.** Containerization provides a consistent environment by bundling the application code and dependencies into a single package. This ensures consistency across the development, test, and production environments.
- **Universal compatibility.** Containers can be deployed virtually anywhere from Windows, Linux, and macOS to cloud providers and data centers. Even serverless deployments support containers.
- **Portability.** The container can be easily moved to a new environment with little to no modifications as all the required software and dependencies are packaged in a standardized container format. (Easily facilitate lift and shift scenarios.)
- **Isolation.** Containers feature strong isolation from external factors due to their containerized nature and sandboxed view of the underlying operating system resources (OS-level virtualization) such as CPU, memory, storage, and network resources.
- **Scaling.** The lightweight nature of containers contributes to rapid scaling capabilities: containers can be easily created and destroyed with limited resources to meet user demand.

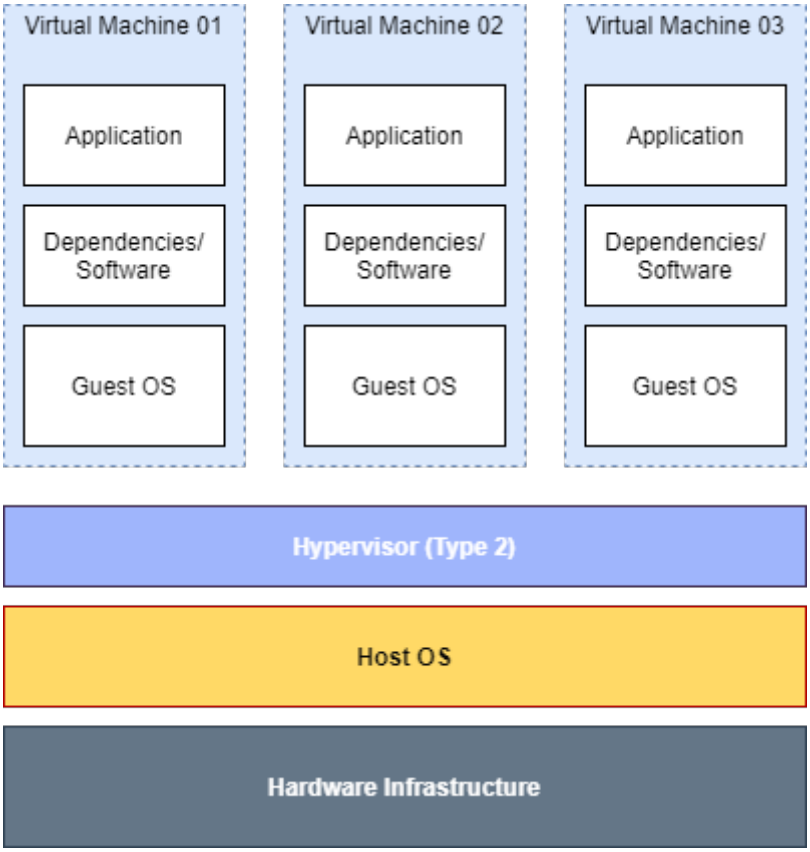
Container vs virtual machines (VMs)

Containers sound a lot like virtual machines. Both can be used to create isolated environments where applications are deployed with all the dependencies preconfigured.

However, there are fundamental differences between the two technologies.

Virtual machines

In their simplest form, VMs are designed to virtualize underlying hardware and run multiple operating systems on a single machine. Due to that, VMs can run different types of operating systems in a virtualized environment (single machine/server). This shared approach to utilize hardware enables organizations to manage physical hardware resources efficiently and cost-effectively.



Structure of virtual machines

A software called a hypervisor is required to create and manage VMs. Some popular types of hypervisors are:

- Microsoft Hyper-V
- VMWare ESXi
- Linux KVM

The main downside when it comes to using VMs is their size. Since each virtual machine contains the operating system image, libraries, software, etc., they can quickly grow into large sizes, resulting in:

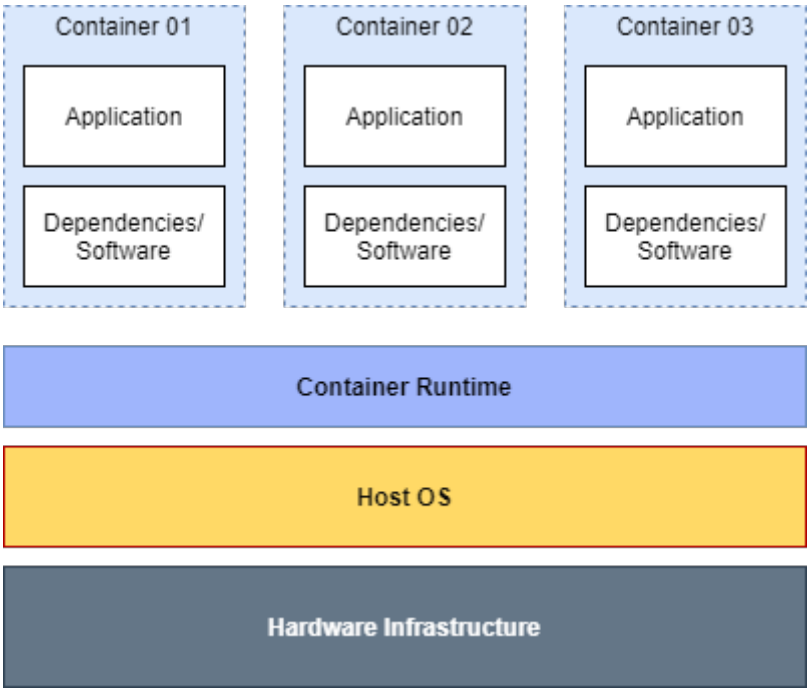
- Additional maintenance
- Scalability issues

Containers

Unlike VMs, containers virtualize the underlying operating system. As a result, a container can consider that all the resources in the underlying operating system are available to it.

Using this shared approach eliminates the need to host a separate OS image in each container and boot into the OS to load the necessary libraries. This offers two benefits over VMs:

- Drastically reduces the overall size of a container compared to a VM
- Allows deploying a larger number of containers in a single server



Structure Of Containers

Additionally, this shared approach of containers will reduce the maintenance needs—developers only need to update and patch the underlying operating system, not each individual container.

Containers utilize software called "container runtime" to execute and manage containers. Some widely used container runtimes are:

- Docker
- containerd
- CRI-O

The main downside of containers is that they are limited to the operating system they are defined for. They can only be deployed in the defined operating system, unlike VMs where users can run Windows virtual machines on a hypervisor-based Linux server and vice versa. If a container is defined for Linux, it can only be deployed in Linux.

(Read our in-depth explainer on containers vs VMs.)

How Kubernetes manages containers

Now we know what containers are. The next logical step is to manage the containers.

For starters: we can always manually manage all our containers. However, this is not a scalable or

efficient solution.

Instead, we can use Kubernetes! Kubernetes, aka K8s, is an open-source container orchestration platform to easily deploy, manage, and scale containerized applications.

(Explore our comprehensive Kubernetes tutorial series.)

Structure of Kubernetes

Kubernetes is the ideal solution to scale containerized applications from a single server to a multi-server deployment. This kind of multi-server distributed approach is known as a Kubernetes cluster.

Kubernetes gets containers and distributes the application based on the resource availability between different servers (Nodes). The application will be shown as a single entity to the end-users; however, in reality, it can be a group of loosely coupled containers running on multiple nodes.

Kubernetes also provides built-in support for containerized apps, including:

- Service discovery
- Load balancing
- Scaling
- Health monitoring

Core Kubernetes components

Kubernetes consists of the following core components:

- Control plane
- Nodes
- Pods

The **control plane** is the brain of the Kubernetes cluster, managing everything from scheduling to cluster events. The control plane consists of different components, like:

- kube-apiserver which manages the API service
- kube-scheduler which manages the deployment of pods to corresponding nodes
- And more

The control plane components can be executed in any node within the cluster. The standard practice is to associate a single instance for the control plane tasks.

The next component are nodes. Nodes are essentially endpoints or machines of a Kubernetes cluster. They are the locations where containers will get deployed in Kubernetes. Nodes are responsible for providing the runtime environment for containers.

Nodes consist of different components such as:

- kube-proxy to maintain the network routes
- kubelet to manage and monitor the health of containers

The final essential component is the pod. Pods are a logical wrapper for a container. (Pods are also the smallest deployable component in Kubernetes.) A K8s Pod refers to either:

- A single container
- Multiple containers grouped together with shared resources (networking, storage, etc.)

The containers within a pod are tightly coupled together and essential for the core functionality of the containerized application.

To sum up how these three components work together: The control plane distributes these pods to the nodes in a Kubernetes cluster depending on:

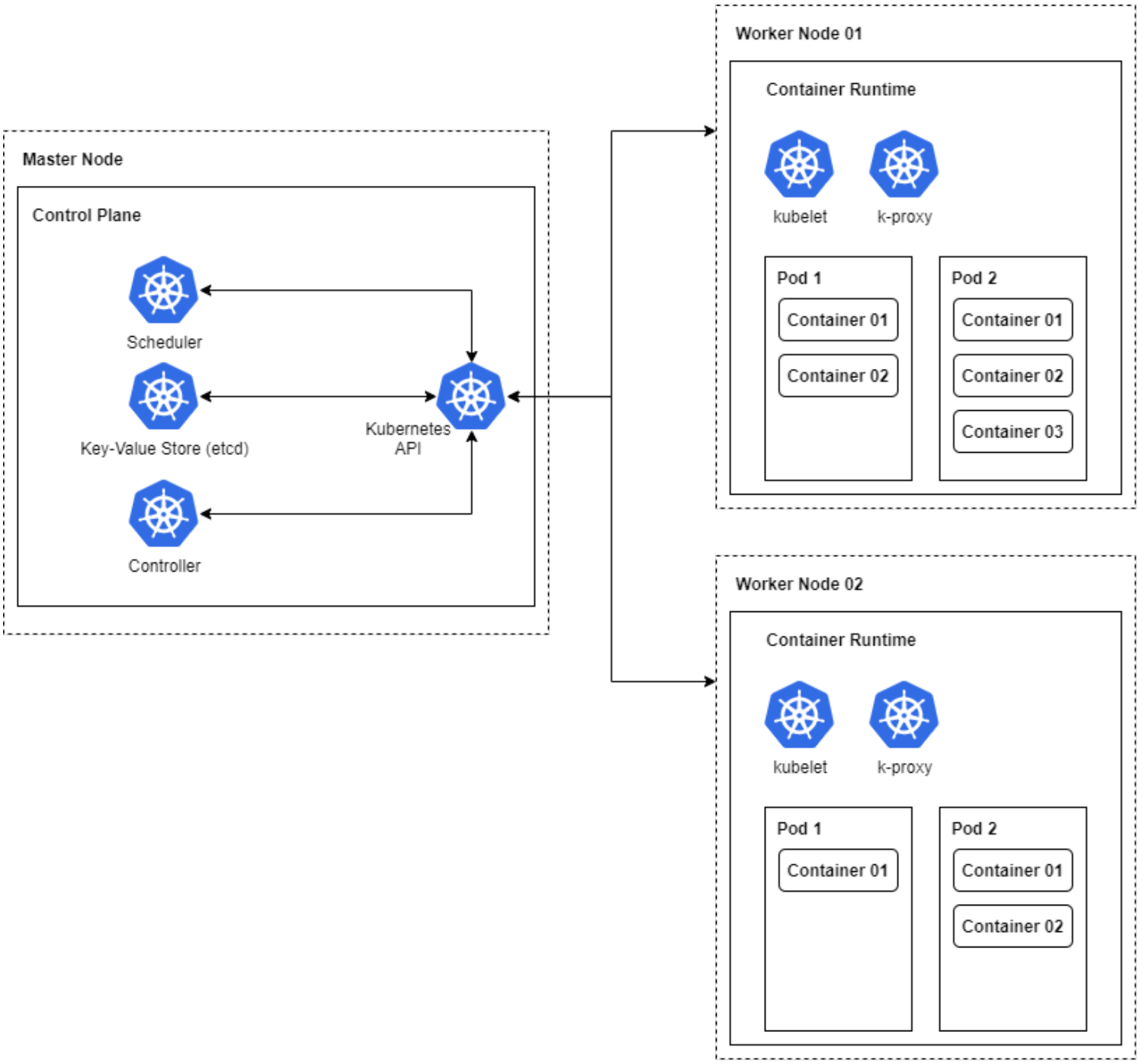
- Resource availability
- Application load

Basic Kubernetes deployment

In a simple Kubernetes deployment, there will be a dedicated server for the control plane (*Master Node*) to manage the complete cluster. The controllers within the control plane will manage all the nodes and services within the cluster. Additionally, they will also take care of scheduling, API service, and state management using etcd.

Other servers or nodes act as the workers (*Worker Nodes*), providing the runtime environment and running the containers (*Kubernetes Pods*). The Kubernetes Pod will wrap a container or multiple containers to create a single entity for easier resource allocation, management, and scaling.

(Understand how Kubernetes deployments work.)



Kubernetes vs normal container runtimes

What truly differentiates Kubernetes from normal container runtimes are the features it offers on top of containers. Some of those features are:

- **Load balancing.** Create load balancers and distribute traffic between the available pods.
- **Automatic scheduling.** Automatically distribute pods in the most appropriate node for optimal performance.
- **Horizontal scaling.** Quickly scale pods up or down to meet the application demands.
- **Self-healing.** All pods are constantly monitored and automatically restarted or replaced in an event of failure.
- **Rollouts.** Kubernetes facilitates managed rollouts and rollbacks to easily deploy containerized applications using different deployment strategies.

Using containers & Kubernetes

Containers and Kubernetes are two technologies that go hand in hand.

As a developer, the simplest way to adapt to them is to:

1. Create containerized applications with multi-platform compatibility as isolated, portable packages.
2. Use the robust and feature-rich Kubernetes platform to orchestrate those containerized applications.

This essentially means that developers can easily maintain the application as a container in a Kubernetes cluster throughout the application lifecycle from development to deployment and maintenance.

As each technology complements the other, combining the strengths of both these technologies will lead to more convenient, shorter, and less complex development and deployment processes.

Kubernetes clusters are supported by all the major cloud providers or on-premise deployments. So, you can simply spin up a cluster and deploy the application with just a few commands while Kubernetes handles the bulk of tedious, time-consuming management tasks (load balancing, scaling, etc.) The built-in features of Kubernetes enable it to manage the whole application without depending on any other external tools or services.

Increasing developer agility

Containers and Kubernetes are modern technologies designed to reduce the workload of a developer or the delivery team. Containers offer a standardized way to package applications and Kubernetes helps you easily manage them.

Now that you understand the basic concepts of containers and Kubernetes, the next step is to explore these concepts to create your perfect containerized application. Start with our Kubernetes tutorial series.

Related reading

- BMC DevOps Blog
- The State of Containers Today: A Report Summary
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools
- Docker Security: 14 Best Practices for Securing Docker Containers
- The Role of Virtualization in DevOps
- Containers Aren't Always the Solution