**What Is Container Orchestration?**

by [BoxBoat](https://boxboat.com/authors/boxboat) | Friday, Jan 25, 2019 | [Docker](https://boxboat.com/categories/docker) [Kubernetes](https://boxboat.com/categories/kubernetes)



Containerization has changed the way software is built, deployed, and maintained. But when it comes to enterprise-grade applications, managing containers can become one big headache. That’s why you need container orchestration.

In this article, we explain what exactly is container orchestration, how it works, and take a brief look at some of the tools available.

**What is Container Orchestration?**

Container orchestration should be seen as a way to handle and manage a large number of containers.

If you have an application that is running on 5 containers, then you can run, deploy, and manage these containers on Docker alone without much difficulty. But for enterprise applications that comprises of a thousand or more containers, then management becomes extremely complicated. This is where container orchestration comes into play.

Many development teams utilize container orchestration to manage containers running in a large and dynamic environment. Container orchestration is used to control and automate a multitude of tasks including provisioning and deploying containers, allocating resources between containers, scaling containers, shifting containers from one host to another if the host becomes unavailable or there is a lack of resources, load balancing, and monitoring the health of both the containers and hosts.

**How Does Container Orchestration Work?**

To orchestrate your containers, you need to use a container orchestration tool such as Kubernetes or Docker Swarm (we’ll talk about these a little later).

Depending on what tool you use, the user would describe the configuration of the application in either a YAML or JSON file. This configuration file will communicate with the container orchestration tool to instruct where to retrieve the container images (this could be from a private registry or a public registry like Docker Hub), how to establish the network between the containers, where to store logs, and how to mount storage volumes.

Usually, development teams will branch out and version control these config files to deploy the same applications on different development and testing environments before deploying them to a production environment.

As for container deployment, the orchestration tool will deploy these onto the host, typically in replicated groups. Whenever a new container is deployed into a cluster, the container orchestration tool will automatically schedule the deployment and look for an appropriate host based on predefined constraints such as memory and CPU requirements. Containers can even be placed by their relations to other hosts or by their labels or metadata.

On placing the container on the host, the orchestration tool will then follow the specifications that have been laid out in the container’s compose file (Dockerfile) to manage the container’s lifecycle.

The great thing about container orchestration tools is that you can utilize them in any environment. Containers can run on on-premise servers, local machines, and cloud servers such as Amazon Web Services (AWS), Google Cloud Platform, and Microsoft Azure.

**What Container Orchestration Tools Are Available?**

There are a number of container orchestration tools, with the majority of the well-known frameworks being open source. Here, we take a look at the three main container orchestration tools that are extremely popular in the application container market.

**Kubernetes**

Considered as the gold standard for container orchestration, this open source project was developed as a by-product of Google’s Borg project. It is backed by many giants of the cloud computing world including AWS, Microsoft, IBM, Intel, and Cisco.

[Kubernetes](https://boxboat.com/2018/12/13/kubernetes-training-module-1/) continues to be popular with practitioners of [DevOps](https://boxboat.com/2018/12/26/what-is-devops/) since it enables them to deliver a Platform-as-a-Service, which creates an abstract hardware layer for development teams.

Known for its portability, Kubernetes’ starting pointing is the cluster itself, which allows you to move workloads around without having to worry about redesigning your application or redefine your infrastructure.

**Key Components of Kubernetes**

**Cluster:** A set of nodes that adjoined to at least one master node and several other worker nodes.

**Master:** Also known as the Kubernetes master. This component manages the deployment and scheduling of the application across the nodes. The master sends instructions to the nodes via the Kubernetes API server. It also assigns nodes to pods based on the resources and predefined constraints.

**Kubelet:** This component sits within the node. The Kubelet is responsible for starting, stopping, and maintaining the application container based on the instructions from the Kubernetes API server

**Pod:** A pod is a group of containers that share the same IP address. This allows containers to always be guaranteed to be group together on the host and share the same resources. A YAML or JSON is used to define a pod.

**Docker Swarm**

While the Docker community has embraced Kubernetes as the number one container orchestration tool, the open source project has its own fully integrated offering, Docker Swarm. The tool provides a much easier path to container deployment although it is not extensible as Kubernetes. On saying that, Docker Enterprise Edition bundles both Kubernetes and Swarm together in an effort to making both tools complementary to one another.

**Key components of Docker Swarm**

**Swarm:** Similar to clusters, a swarm is a set of nodes that is adjoined to at least one master node and several other worker nodes.

**Service:** A set of tasks that have been defined by the swarm administrator and performed by the manager node. It defines which container images should be used and what instructions will run in each container.

**Manager node:** On deployment of an application, the manager node delivers the tasks to the worker nodes and is also responsible for managing the state of the swarm. The manager node can also run the same services as the worker nodes but you can configure it to run only manager-related services.

**Worker nodes:** These nodes perform the task assigned by the manager node. Worker nodes report back to the manager node to inform the current state of the task assigned, helping the manager node to keep track of the services running in the swarm.

**Apache Mesos and Marathon**

Apache Mesos is an open source software that was originally developed at the University of California at Berkeley and is now used by organizations like Uber, Twitter, and PayPal.

Mesos’ interface is extremely lightweight and lets you easily scale up to 10,000+ nodes and it also lets you use your own framework to evolve it as you please. Mesos’ API supports a number of languages including Java, C++, and Python.

The downside though, is that Mesos only provides the management of clusters, not orchestration, and has a higher learning curve. Fortunately, a number of frameworks have been developed on top of Mesos to facilitate more features. One of these features is Marathon, a “production-grade” container orchestration tool.

**Key Components of Mesos and Marathon**

**Master Daemon:** A part of the master node which manages agent daemons.

**Agent Daemon:** Another part of the master node which performs tasks set by the framework, Marathon.

**Framework:** As mentioned, Mesos is not an orchestration tool. Instead, Marathon receives the resources from the Mesos master daemon, in the form of offers. Marathon then executes the tasks based on the resources defined by the offer.

**Offer:** The Mesos master daemon collects all the information about the agent node’s memory and CPU availability and then sends this information to Marathon. This helps Marathon know what resources are available.

**Which Container Orchestration Tool Should You Go For?**

All the container orchestration tools listed in this article have their pros and cons. If you’re running smaller deployment and you don’t have much need to scale, then Docker Swarm is probably suitable. But if you want to scale to tens of thousands of containers, then Mesos is your best shout, with Kubernetes not too far behind. In terms of ease of use, Docker Swarm has a lower learning curve, Mesos would likely require a level of specialization and technical know-how, and Kubernetes sits right in the middle.

For help and support with your container orchestration, CI/CD pipeline, or container development, get in contact with BoxBoat today.

## ****What is container orchestration?****

**Container orchestration** is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container’s lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.

## vSphere with Kubernetes 101 - An Introduction for vSphere Administrators

[Download Now](https://www.vmware.com/learn/484948_REG.html?src=WWW_us_ELQ_TM0HWr93Qjf8B1EgkaY7&int_cid=7012H000001KPUp)

## ****What are containers and their benefits?****

[Containers](https://www.vmware.com/topics/glossary/content/containers) are a method of building, packaging and deploying software. They are similar to but not the same thing as [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine) (VMs). One of the primary differences is that containers are isolated or abstracted away from the underlying operating system and infrastructure that they run on. In the simplest terms, a container includes both an application’s code and everything that code needs to run properly.

Because of this, containers offer many benefits, including:

* **Portability:** One of the biggest benefits of containers is that they’re built to run in any environment. This makes containerized workloads easier to move between different cloud platforms, for example, without having to rewrite large amounts of code to ensure it will execute properly, regardless of the underlying operating system or other factors. This also boosts developer productivity, since they can write code in a consistent manner without worrying about its execution when deployed to different environments—from a local machine to an on-premises server to a [public cloud](https://www.vmware.com/topics/glossary/content/public-cloud).
* **Application development:**Containers can speed up application development and deployments, including changes or updates over time. This is particularly true with containerized microservices. This is an approach to software architecture that entails breaking up a larger solution into smaller parts. Those discrete components (or microservices) can then be deployed, updated or retired independently, without having to update and redeploy the entire application.
* **Resource utilization and optimization:**Containers are lightweight and ephemeral, so they consume fewer resources. You can run many containers on a single machine, for example.

## ****Why do we need container orchestration?****

Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort. Particularly when paired with [microservices](https://www.vmware.com/topics/glossary/content/microservices)—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.

This can introduce significant complexity if managed manually. Container orchestration is what makes that operational complexity manageable for development and operations—or [DevOps](https://www.vmware.com/topics/glossary/content/devops)—because it provides a declarative way of automating much of the work. This makes it a good fit for DevOps teams and culture, which typically strive to operate with much greater speed and agility than traditional software teams.

## ****What are the benefits of container orchestration?****

Container orchestration is key to working with containers, and it allows organizations to unlock their full benefits. It also offers its own benefits for a containerized environment, including:

* **Simplified operations:**This is the most important benefit of container orchestration and the main reason for its adoption. Containers introduce a large amount of complexity that can quickly get out of control without container orchestration to manage it.
* **Resilience:**Container orchestration tools can automatically restart or scale a container or cluster, boosting resilience.
* **Added security:**Container orchestration’s automated approach helps keep containerized applications secure by reducing or eliminating the chance of human error.

## ****What is Kubernetes container orchestration?****

Kubernetes is a popular open source platform for container orchestration. It enables developers to easily build containerized applications and services, as well as scale, schedule and monitor those containers. While there are other options for container orchestration, such as Apache Mesos or Docker Swarm, Kubernetes has become the industry standard. Kubernetes provides extensive container capabilities, a dynamic contributor community, the growth of cloud-native application development and the widespread availability of commercial and hosted Kubernetes tools. Kubernetes is also highly extensible and portable, meaning it can run in a wide range of environments and be used in conjunction with other technologies, such as service meshes.

In addition to enabling the automation fundamental to container orchestration, Kubernetes is considered highly declarative. This means that developers and administrators use it to essentially describe how they want a system to behave, and then Kubernetes executes that desired state in dynamic fashion.

## ****What is multi-cloud container orchestration?****

In the most basic sense, the term “[multi-cloud](https://www.vmware.com/topics/glossary/content/multi-cloud)” refers to an IT strategy of using two or more cloud services from two or more providers. In the context of containers and orchestration, multi-cloud usually means the use of two or more cloud infrastructure platforms, including public and private clouds, for running  applications. Multi-cloud container orchestration, then, refers to the use of an orchestration tool to operate containers across multi-cloud infrastructure environments—instead of running containers in a single cloud environment.  
  
Software teams pursue multi-cloud strategies for different reasons, but the benefits can include infrastructure cost optimization, flexibility and portability (including reducing vendor lock-in), and scalability (such as dynamically scaling out a cloud from an on-premises environment when necessary.) Multi-cloud environments and containers go hand-in-hand because of the latter’s portable, “run anywhere” nature.

## ****Container orchestration versus Docker****

Docker is a specific platform for building containers, including the Docker Engine container runtime, whereas container orchestration is a broader term referring to automation of any container’s lifecycle. Docker also includes Docker Swarm, which is the platform’s own container orchestration tool that can automatically start Docker containers.