How Containers and Kubernetes Accelerate Microservices

In Part 1 of this series, I discussed how a shift from monolithic applications to microservices enables organizations to be more agile while still building more reliable applications. This one-two punch of enterprise IT efficiency is increasingly built upon two related, yet distinct, building blocks: containers and Kubernetes.

Good Things Come in Small Packages

Who doesn't love opening a birthday present? That nice pretty package might hold exactly what you've been waiting for. Similarly, software applications also come in packages—whether it's a game like Roblox, an online banking app, or a human-resources system for benefits.

Traditionally, software packages have included all the code needed to run the application on a particular operating system, like Windows or Linux. Surprisingly, you need more than just application code to run an application. You also need other applications. For instance, an application for looking up stock prices might use a library to convert company names to ticker symbols and vice versa. This functionality is generic and

not value-added, but it's still important to allow a user to type "Apple" and get the stock "AAPL." The library is an example of a dependency. Without IT knowing it, any application might have hundreds of these types of dependencies.

One of the main reasons that containers became so popular is that they provided a mechanism and format to package application code—with its dependencies—in a way that made it easy to run an application in different environments. This might sound fairly underwhelming given all the hype around containers, but it solved a big problem for developers who were constantly fighting environment-compatibility issues between their development laptops, testing environments, and production. By using containers to package their applications, they could "code once and run anywhere," dramatically speeding up application delivery.

In Part 1, we looked at how speed and agility are a primary benefit of microservices. By making it easier to build and deploy applications, containers accelerate this speed. While containers weren't new in 2014 when Docker burst onto the scene, Docker was the first company to see how containers could accomplish more as an application-packaging mechanism than as simply a slightly more efficient form of virtualization.

The Complexity of Managing Multiple Containers

Because modern applications comprise multiple microservices, they're also often made up of multiple containers. This makes entire applications easy to run in multiple environments, but it also opens a question: "How do I keep track of and manage all these containers?"

Almost as soon as Docker became popular, software engineers and the companies they worked for were trying to figure out how to "monetize" containers. After all, developers usually don't have big budgets to work with. However, managing—or orchestrating—all these containers was seen as a necessary and lucrative extension of the container revolution early on. In June 2014, Google released Kubernetes as an open-source project and one of the first to take on the management challenges of containers.

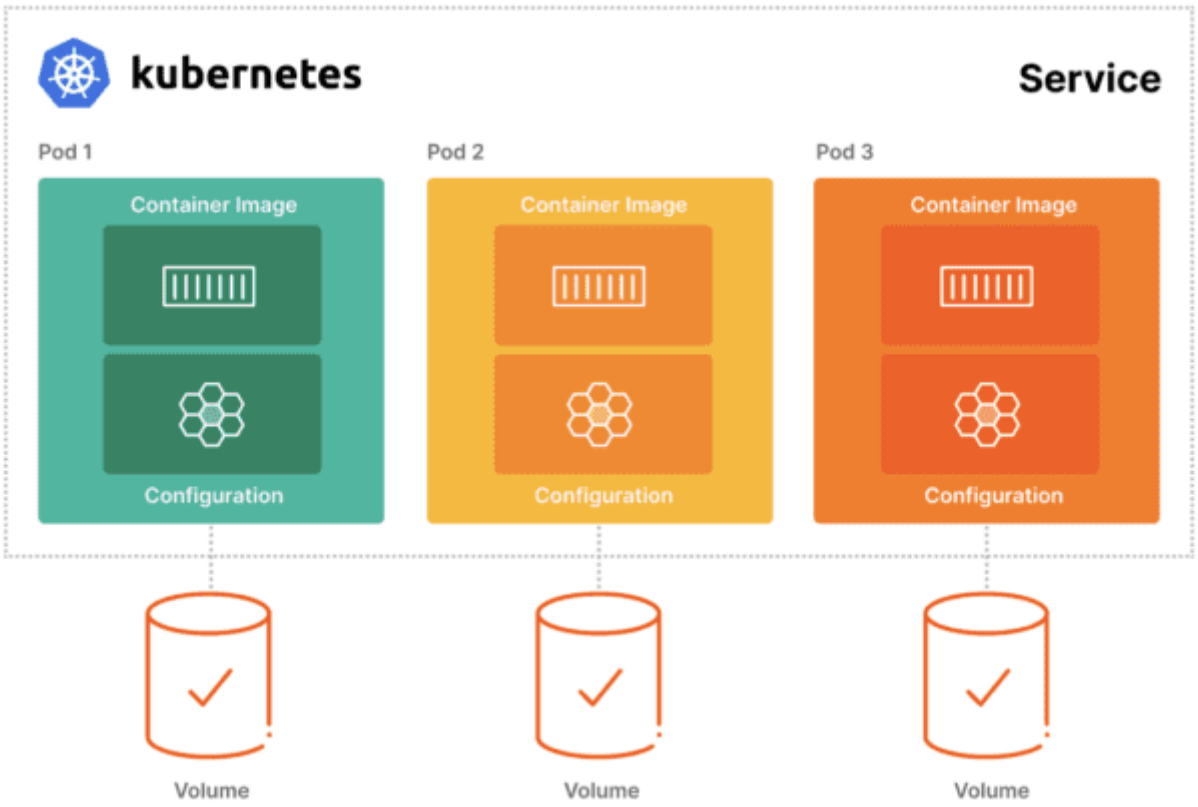
For instance, to run a highly available application in a container, you always need to have a certain number of containers running at any given time, especially if the servers fail. Because of the sheer number of containers running at a time, this needs to be fully automated, with no human intervention. A DevOps engineer should be able to declare the desired state of their application (e.g., how much CPU or RAM and how many replicas), and if the actual state of the application differs from the desired state, Kubernetes will correct it. This correction could involve restarting stopped containers or deleting them from one host and starting them on another that has more available resources. This declarative approach to application operations is easy to understand, (relatively) easy to implement, and massively scalable. As a result, it's not surprising that Kubernetes adoption is through the roof.

The Anatomy of a Kubernetes Application

Like the microservices that they comprise, Kubernetes applications are composed of several components. Here are some of the most important components:

- **Container image:** This is the application code and dependencies needed to run a container packaged in a standard format, like Docker or OCI. The container image is immutable, which means that it's always the same, regardless of what environment it's running in. Changes in state, like a database write, aren't stored in the container but rather in a data volume attached to the container (more on volumes below).

- **Pod:** A Pod is one or more containers that share storage and network resources that should always run together on the same host. The Pod is the basic unit of analysis of a Kubernetes application. While most Pods only have a single container, some apps require more than one container even for basic functionality—for example, "one container serving data stored in a shared volume to the public while a separate *sidecar* container refreshes or updates those files."
- **Service:** A service is a logical grouping of Pods that are exposed as a network service. After all, an application is nothing unless someone can access it.
- **Configuration:** The workhorse of Kubernetes, configuration (sometimes referred to as metadata) defines things like deployment templates and environment variables needed to run a specific application composed of immutable containers in different locations. For example, I can easily run a containerized app in my data center or AWS only if I can expose the web front end via a specific IP address, which will be different from environment to environment. Rather than hard-coding this variable in the app, breaking the immutable nature of containers, you can include it as a configuration. Secrets, like passwords and API keys, are another common example of configuration.
- **Volumes:** A good container never changes, but a good app always does! It takes e-commerce orders or displays the latest game or movie. Applications that collect and display data are called stateful applications. However, to remain consistent with the immutability of containers, data for an application isn't stored in the container itself but rather a volume (or an object storage system like S3 or Pure Storage® FlashBlade®). When people colloquially refer to stateful containers, they're referring to the combination of a container and a volume.

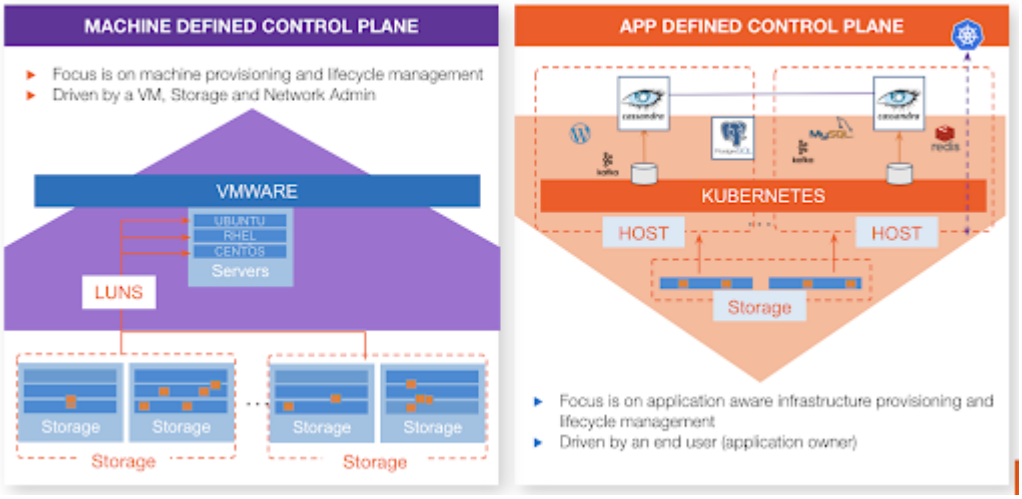


Apps, Not Infrastructure, Are Royalty

The virtualization revolution was about making infrastructure more efficient. It was wasteful to run only one

application per physical server. So VMware came along and made it possible to run multiple apps per server, leading to data-center consolidation and cost savings. The cloud took this further and enabled enterprises to rent infrastructure capacity instead of requiring them to buy it. Containers extend this efficiency further, but that isn't the value. Containers and Kubernetes are really about *reinventing* how we build and run applications.

No matter how efficient your infrastructure is, you're only halfway through the marathon. Bringing applications to production is how you generate revenue and stay ahead of the competition. Teams with an infrastructure-focus on operations ask, "What's the best way to manage this machine?" On the other hand, teams with an application-focus on operations ask, "What's the best way to manage this value-generating activity?" Increasingly, this management is driven by a self-service experience that differs greatly from traditional infrastructure managed via tickets.



What Lies Beyond Containers?

So there you have it. We started with an understanding of why microservices are the modern architectural paradigm of choice, how containers are the perfect building block for microservices, and why we need Kubernetes once the number of containers explodes. But the story might not end there. Increasingly, Kubernetes is being used to manage applications that are not containerized, only *componentized*. So, you can build modern, efficient applications without containers. As long as the application is automatable and separates configuration and storage from compute, Kubernetes can help. While containerization is likely to continue, we may increasingly see enterprises adopt Kubernetes without containers for some applications. When they do, we'll be here to explain how you can get the most out of them.

In the meantime, keep an eye out for Part 3, in which we'll look at storage and data requirements for Kubernetes apps.

- Part 1: Modern Applications: From Monolith to Microservices
- Part 2: How Containers and Kubernetes Accelerate Microservices (this post)
- Part 3: Storage and Data Requirements for Kubernetes Apps
- Part 4: Portworx: The Data-services Platform for Kubernetes