Kubernetes: The Future of Infrastructure

How Kubernetes Is Modernizing the Microservices Architecture

Marco Palladino

Kubernetes: The Future of Infrastructure

How Kubernetes Is Modernizing the Microservices Architecture

Marco Palladino

In this book we'll discuss how Kubernetes enhances a container-based microservices architecture. We'll examine the rise of containers and Kubernetes to understand the organizational and technical advantages of each. This will include a deep dive into the ways Kubernetes can improve processes for deploying, scaling, and managing containerized applications.

Content

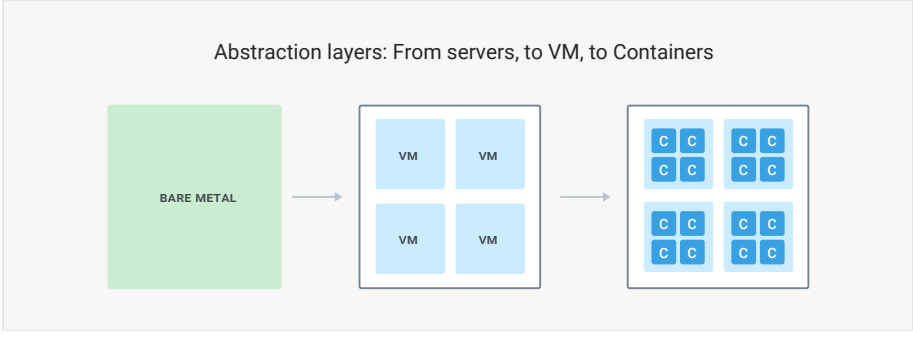
Next Generation Application Development

The world of IT infrastructure has evolved dramatically since the days of relying solely on bare metal hardware to support application development. In the early 2000s, a company called VMWare broke the mold, giving rise to virtualization — using an abstraction that makes software (virtual machines) look and behave like hardware. The primary difference between virtualization and bare metal is the inherent benefit of using software to "virtualize" infrastructure. Virtualization offers increased flexibility, scalability, reliability, and often overall capability and performance, all while lowering capital and operational expenses.

In recent years, the confluence of cloud computing and big data has fueled another explosion in technology designed to further improve flexibility and convenience at scale — the adoption of containers to facilitate software development and the proliferation of microservices.

Containers are a more efficient way of packaging software — including all the elements needed to run software, such as code, runtime, system tools, system libraries, and settings. 451 Research projects that the overall market for containers will hit roughly $2.7 billion in 2020, a 3.5-fold increase from the $762 million spent on container-related technology in 2016.

With containers, all the moving parts and dependencies across various infrastructure, become less complex and obvious, enabling developers to work with identical development environments and stacks.



Like Virtual Machines, but Better

There are many parallels between the rise of virtual machines and the rise of containers. Containers are transforming the way in which software is developed in the same way that virtualization and virtual machines did in the early and mid-2000s. Before virtual machines, if we wanted to scale our infrastructure to support software development needs, we had to buy more servers and physically scale it. This was a very resource intensive and costly endeavor to not only build out, but also maintain for performance and availability. But virtual machines changed the game, allowing us to scale more easily without needing to run to our hardware vendor to purchase servers. We could increase the efficiency of the hardware we currently had and scale to thousands of virtual machines on top of that existing infrastructure. In their simplest form, containers help us accomplish the same thing by enabling us to squeeze the most out of every single inch of hardware we are running underneath it, but at a much larger scale.

There are many benefits of using containers. First, DevOps teams benefit greatly due to the more efficient approach to software development. Containers allow engineering teams to be more agile by reducing wasted resources and empowering teams to build and share code more rapidly in the form of microservices. On top of this, containerization improves scalability through a more lightweight and resource-efficient approach. This improved scalability, coupled with the improvements in development efficiency and velocity, results in greater time and cost efficiencies.

Containers are highly flexible and scalable. With containers, we can leverage more processes on each virtual machine and increase its efficiency. We can also run any sort of workload within a container because it is isolated, which ensures that each workload is protected from the others. As a result, a container cannot destroy or impact another container.

Ease of use is another core benefit of containers. Similar to how virtual machines were easier to create, scale, and manage compared to physical hardware, containers make it even easier to build software than virtual machines because they can start up in a few seconds. Gone are the days of running the heavy load of processes required by a full virtual machine. Instead, a container affords us the luxury of running a lightweight, isolated process on top of our existing virtual machine. This allows us to quickly and easily scale without getting bogged down with DevOps busy work.

The Next Frontier: Container Orchestration

As container adoption has exploded, so has the adoption of container orchestration. This again parallels the virtualization world where companies like VMWare offer the necessary tools to launch, monitor, create, and destroy virtual machines. Much like virtual machines, containers need to be monitored and orchestrated to ensure the underlying virtual machines are working properly.

Container orchestration allows developers to better track, schedule and operationalize various containers at scale. If we want to run multiple containers across multiple servers and virtual machines — which we'll need to do if we're using microservices — this would require otherwise require substantial DevOps resources to make a reality.

The many moving parts require us to answer several questions, such as when to start the right containers, how to ensure the containers can talk to each other, what the storage considerations are, and how to ensure high availability across our infrastructure? There's no wonder the term used to do all of this is "container orchestration". Fortunately, there are tools designed to do just this — removing the complexity of orchestrating our container executions on the underlying virtual machines, much like how AWS makes it simple to provision EC2 instances. We just create an instance on-demand when needed. We don't have to worry about the infrastructure issues such as what physical hardware it's going to be executed on.

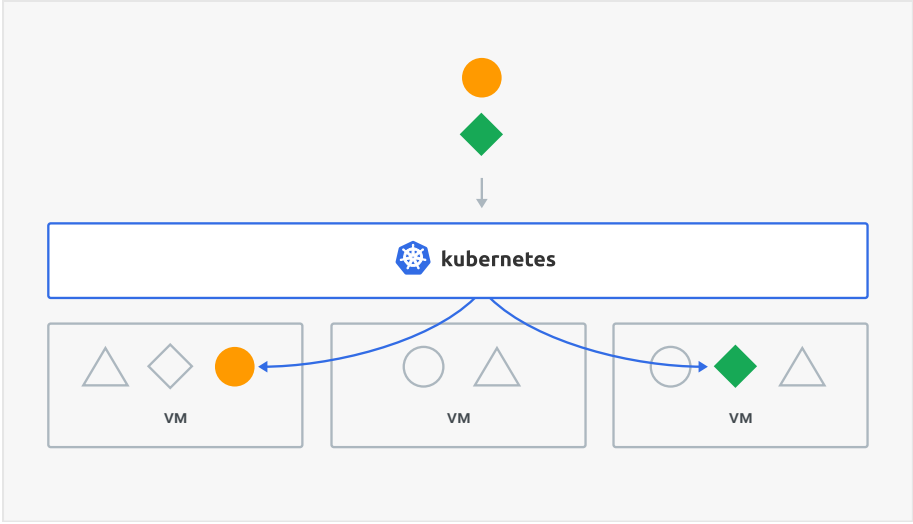
Likewise, containers and orchestration tools allow us to start new containers without having to worry what underlying virtual machine is going to take that workload. For example, a container orchestration tools such as Kubernetes will do things like decide if a virtual machine is underutilized and then decide to run a container on that virtual machine over another.

Kubernetes: Modern Infrastructure for Modern Applications

As recent survey data from The New Stack suggests that container adoption is the most significant catalyst of orchestration adoption. In fact, 60 percent of respondents who've deployed containers in production report that they also rely on Kubernetes in production to assist with orchestration. Another 19 percent of respondents with broad container deployments in production were in the initial stages of broad Kubernetes adoption.

So, what exactly is Kubernetes? Started by Google in 2014, Kubernetes is an open source project that focuses on building a robust orchestration system for running thousands of containers in production. Through automation of these processes, Kubernetes can eliminate many of the painful manual tasks and infrastructure complexity that often fall on DevOps teams to help with the deployment, scaling, and management of containerized applications.

Kubernetes allows us to not only easily run containers, but also to easily run different kinds of workloads alongside our containers. The key to the rise of Kubernetes is in its ability to understand that, when you are running a system, it's not just about running a container. Furthermore, not every container is the same. A container can be a web app, or it can be something that needs to store data persistently.



Common questions that are addressed by Kubernetes include:

- How do we design applications that may consist of many moving parts, but can still be easily deployed and orchestrated?
- How can we design applications that can be easily moved from one cloud to another?
- How can we keep storage consistent with multiple instances of an application?
- How can we ensure load is evenly distributed across all containers?
- How can we reuse our existing technical skills and techniques on application development and design without deep diving into other areas that can slow us down?

Even with the rise of Kubernetes as the de facto container orchestration system, there are still some misconceptions on what it is. First off, many folks initially do not understand the difference between Docker and Kubernetes. Simply put, Docker is a tool used to create containers whereas Kubernetes provides the management of those Docker containers. Another misunderstanding is that Kubernetes is not a platform as a service (PaaS), however we can use it as the foundation to develop a PaaS if we want.

How Kubernetes Gets Work Done

Within Kubernetes, each node runs several services and a Docker engine either in a virtualized environment or a physical server. There are two types of nodes: a master node which serves as the controller of a given cluster and worker nodes which are used to run or host our services represented as containers.

Within a worker node, we have the concept of a “pod” which represents a unit of work or collection of containers that are deployed to the same host. Usually we will run a single container or service inside of a pod. However, there are instances where it makes sense to run several containers inside the same pod if they are tightly coupled together.

Kubernetes then connects our pod to your environment and manages our pod, including scaling, rolling deployments and monitoring. Pods have their own IP addresses which means that applications can easily find our service through Kubernetes service discovery.

Next up is the notion of a “service”. A service groups together logical collections of pods that perform the same function to present them as a single entity. When a service is created, all the nodes of a cluster are made aware of that new service. What is does is provide a single point of access which makes it easier to facilitate communications between a collection of pods. The deployment of a service simplifies our container design and makes it easier for a user to discover containers.

A “label” is an organizational concept that creates a metadata tag for Kubernetes resources for easy searchability. This allows developers to easily query based on how they are labeled. Since a host of Kubernetes functionalities rely on querying the clusters for certain resources, the ability to label resources is a critical to ensure developer efficiency. Labels are fundamental to how both services and replication controllers function.

A volume represents a location where containers can access and store information. For the application, the volume appears as part of the local filesystem. But volumes may be backed by local storage, Ceph, Gluster, Elastic Block Storage, and several other storage backends.

A namespace functions as a grouping mechanism inside of Kubernetes. Services, pods, replication controllers, and volumes can easily cooperate within a namespace, but the namespace provides a degree of isolation from the other parts of the cluster.

A replication controller is designed to maintain as many pods that have been requested by a user — allowing for easy scaling. If a container goes down, the replication controller will start up another container, ensuring that there are always the correct number of replica pods available.

The Container Orchestration Landscape

There are a couple of tools that also provide container orchestration capabilities. The two biggest players competing with Kubernetes are Docker Swarm and Mesosphere DC/OS. Docker Swarm is an easier-to-use option, which hits Kubernetes where they receive the most criticism around being very complex to deploy and manage.

Mesosphere DC/OS is a container orchestrator that was designed for big data. It was designed to run containers alongside other workloads such as machine learning and big data and offers integrations with related tools such as Apache Spark and Apache Cassandra.

Overall, Kubernetes is currently the most mature and popular out of the three as evidence by the number of community contributors and enterprise adoption. The keys to their success has been their ability to provide not only the building blocks for launching containers and monitoring those containers, but they have also focused their efforts on creating different sets of container use cases on top of their platform to address different types of advanced workloads.

In Kubernetes for example, we can find native objects, native entities within the system that allow us to start a daemon, or to start a container, or to start a database. For other solutions, there is no distinguishing between containers that are running something that could be destroyed at any time.

How Kubernetes is Changing Microservices Architectures

The impact of a microservices framework to the development of IT solutions in today's enterprises is clear. At a high level, microservices allow for applications to be engineered into smaller, independent services, that are not dependent upon a specific coding language. This means that when we are building a monolithic application, a microservices architecture allows an engineering organization to decouple the application into disparate components that are not dependent on each other. Instead, the components can be combined to offer the full breadth of functionality of that monolithic application.

These components or microservices communicate with each other through an API. And since it is not dependent on a programming language, we can have multiple teams building various components across languages without any compatibility issues. For example, we can have a team running a microservice in Ruby with a container running on Kubernetes communicate seamlessly with a microservice built in Python running in a container via Kubernetes.

As we adopt a microservices architecture, it's also important to figure out how to monitor across the different services. This is where Kubernetes and its growing ecosystem of tools comes into the picture. Kubernetes provides organizations with more options to transition from virtual machines to containers. Kubernetes has done a great job promoting an ecosystem around itself so that when we use Kubernetes, we know that out of the box, there are monitoring tools, CI/CD tools, and many platforms all that natively support Kubernetes.

As a result, the decision to use Kubernetes is not solely based on which container orchestration tool to use to further an organization's microservices strategy. What organizations are considering is also what the complementary ecosystem of tools looks like. The Kubernetes ecosystem offers all the building blocks for everything we need to leverage containers to build out a rock solid microservices architecture.

The Future is Bright

Containers are quickly enveloping the world of software development. And the momentum behind Kubernetes as the future of infrastructure is not slowing down any time soon. It has become the go to container orchestrator through its deep expertise, enterprise adoption, and robust ecosystem. With a growing number of contributors and IT service providers backing the framework, Kubernetes will continue to improve and expand upon its functionality, the types of applications it can support, and integrations with the overarching ecosystem

About the Author

Marco Palladino is an inventor, software developer, and internet entrepreneur based in San Francisco, California. He is the co-founder and CTO of Kong, the most widely adopted OSS API and Microservice gateway. Besides being a core maintainer, Marco is currently responsible for the design and delivery of the Kong products, while also providing the technical thought leadership around APIs and Microservices within Kong and the external community. Marco was also the co-founder of Mashape, which started in 2010 and is today the largest API marketplace in the world.

Konghq.com

Kong Inc.
contact@konghq.com

251 Post St, 2nd Floor
San Francisco, CA 94108
USA