Microsoft Open Source Blog (https://cloudblogs.microsoft.com/opensource/)

Demystifying containers, Docker, and Kubernetes

July 15, 2019 • 4 min read

Jason Haley (https://cloudblogs.microsoft.com/opensource/authhaley/)

Azure MVP

Cloud (https://cloudblogs.microsoft.com/opensource/product/cloud/), Containers (https://cloudblogs.microsoft.com/opensource/product/containe Tutorials and demos (https://cloudblogs.microsoft.com/opensource/content-type/tutorials-and-demos/), Community/partners

(https://cloudblogs.microsoft.com/opensource/scenario/commu
partners/)

Modern application infrastructure is being transformed by containers. The question is: How do you get started?

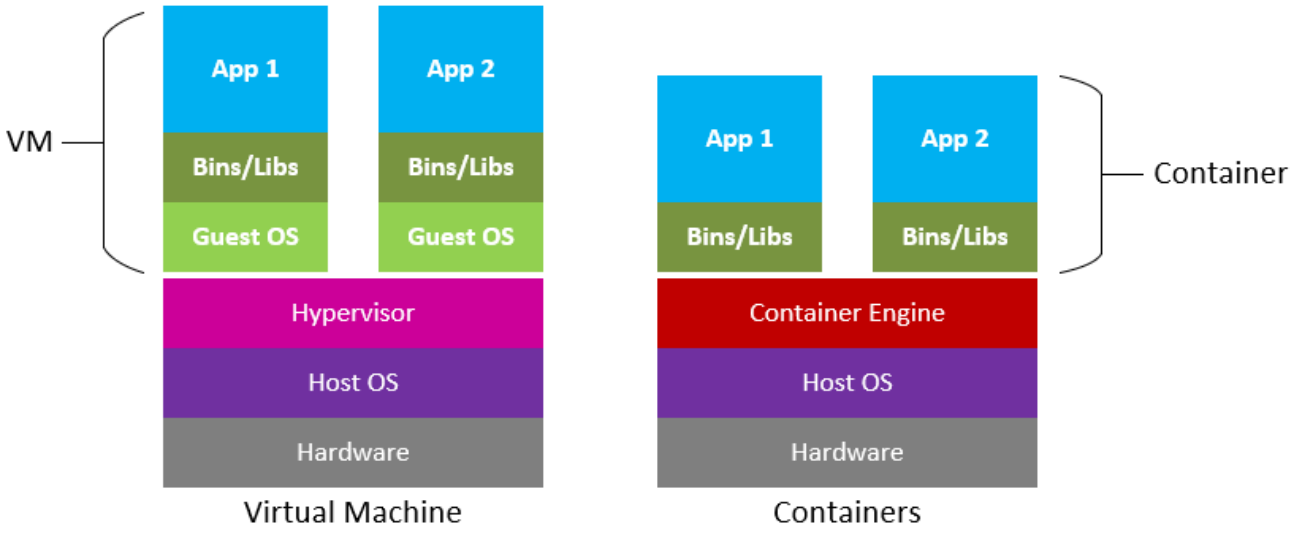
Understanding what problems containers, Docker, and Kubernetes solve is essential if you want to build modern cloud-native apps or if you want to modernize your existing legacy applications. In this post, we'll go through what they are and how you can learn more to advance to the next level.

What are containers?

Containers effectively virtualize the host operating system (or kernel) and isolate an application's dependencies from other containers running on the same machine. Before containers, if you had multiple applications deployed on the same virtual machine (VM), any changes to shared dependencies could cause strange things to happen—so the tendency was to have one application per virtual machine.

The solution of one application per VM solved the isolation problem for conflicting dependencies, but it wasted a lot of resources (CPU and memory). This is because a VM runs not only your application but also a full operating system that needs resources too, so less would be available for your application to use.

Containers solve this problem with two pieces: a container engine and a container image, which is a package of an application and its dependencies. The container engine runs applications in containers isolating it from other applications running on the host machine. This removes the need to run a separate operating system for each application, allowing for higher resource utilization and lower costs.



If you want to learn more about containers, watch this short video on why you should care about containers (https://www.youtube.com/watch?v=EUitQ8DaZW8&list=PLLasX02E8BPCrIhFrc_ZiINhbRkYMKdPT&index=2&t=3s).

What is Docker?

Docker (https://docker.com) was first released in 2013 and is responsible for revolutionizing container technology by providing a toolset to easily create container images of applications. The underlying concept has been around

longer than Docker's technology, but it was not easy to do until Docker came out with its cohesive set of tools to accomplish it. Docker consists of a few components: a container runtime (called dockerd), a container image builder (BuildKit), and a CLI that is used to work with the builder, containers, and the engine (called docker).

Docker images vs. Docker containers

A Docker image is a template; a Docker container is a running instance of that template.

To create an image with your application's source code, you specify a list of commands in a special text file named Dockerfile (https://docs.docker.com/engine/reference/builder/). The docker builder takes this file and packages it into an image. Once you have an image, you push it to a container registry—a central repository for versioning your images.

When you want to run a Docker image, you need to either build it or pull the image from a registry. DockerHub (https://hub.docker.com/) is a well-known public registry, but there are also private registries like Azure Container Registry (https://azure.microsoft.com/en-us/services/container-registry/) that allow you to keep your application images private.

If you want a hands-on example, this is a good great resource: Deploy Python using Docker containers (https://code.visualstudio.com/docs/python/tutorial-deploy-containers).

What is Kubernetes?

Kubernetes (https://azure.microsoft.com/en-us/topic/kubernetes/) is an open-source container management platform that unifies a cluster of machines into a single pool of compute resources. With Kubernetes, you organize your applications in groups of containers, which it runs using the Docker engine, taking care of keeping your application running as you request.

Kubernetes provides the following:

- **Compute scheduling**—It considers the resource needs of your containers, to find the right place to run them automatically
- **Self-healing**—If a container crashes, a new one will be created to replace it.
- **Horizontal scaling**—By observing CPU or custom metrics, Kubernetes can add and remove instances as needed.
- **Volume management**—It manages the persistent storage used by your applications
- **Service discovery & load balancing**—IP address, DNS, and multiple instances are load-balanced.

- **Automated rollouts & rollbacks**–During updates, the health of your new instances are monitored, and if a failure occurs, it can roll back to the previous version automatically.
- **Secret & configuration management**. It manages application configuration and secrets.

Kubernetes uses a master/slave communication model where there is at least one master and usually several worker nodes. The master (sometimes called the control plane) has three components and a data store:

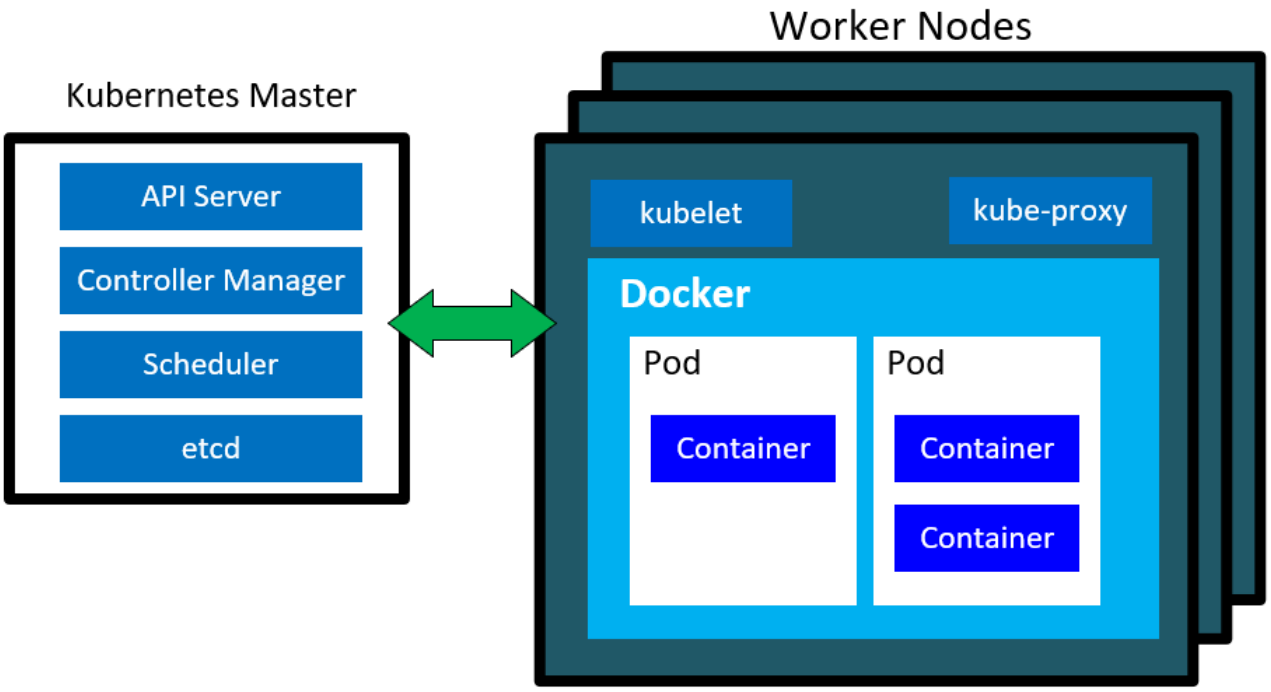
- **API server**—exposes the Kubernetes API for controlling the cluster
- **Controller manager**—responsible for watching the cluster's objects and resources and ensuring the desired state is consistent
- **Scheduler**—responsible for scheduling compute requests on the cluster
- **etcd**—an open-source distributed key value store used to hold the cluster data

The worker nodes provide the container runtime for your applications and have a few components responsible for communicating with the master and networking on every worker node:

- **Kubelet**—responsible for communicating to the master and ensuring the containers are running on the

node

- **Kube-proxy**—enables the cluster to forward traffic to executing containers
- **Docker (container runtime)**—provides the runtime environment for containers



The master and workers are the platform that run your applications. In order to get your applications running on the cluster, you need to interact with the API server and work with the Kubernetes object model.

Kubernetes objects

To run an application on Kubernetes, you need to communicate with the API server using the object model. The objects are usually expressed in .yaml or .json format; kubectl is the command-line interface used to interact with the API.

The most common objects are:

- **Pod**—a group of one or more containers and metadata
- **Service**—works with a set of pods and directs traffic to them
- **Deployment**—ensures the desired state and scale are maintained

To find out more about how your applications work on Kubernetes, watch this short video by Brendan Burns on How Kubernetes works (https://www.youtube.com/watch?v=daVUONZqn88&list=PLLasX02E8BPCrIhFrc_ZiINhbRkYMKdPT&index=3&t=2s).

Here is a great walkthough (https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough) that uses a Python voting application and a Redis cache to help you get started with the Kubernetes concepts.

Conclusion

Containers are the foundation of modern applications. Docker provides the toolset to easily create container images of your applications, and Kubernetes gives you the platform to run it all.

Now that you know the basic pieces of the puzzle and have a better idea of what containers, Docker, and Kubernetes are all about, you can learn more at Kubernetes Learning Path. (https://azure.microsoft.com/en-us/resources/kubernetes-learning-path/)

Tags

Docker (https://cloudblogs.microsoft.com/opensource/tag/docker/)

Kubernetes (https://cloudblogs.microsoft.com/opensource/tag/kubernete

Older post (https://cloudblogs.microsoft.com/opensource/2019/0

Related blog posts

Getting Linux based eBPF programs to run with eBPF for Windows › (https://cloudblogs.microsoft.com/opensource/2022/02/22/getting-linux-based-ebpf-programs-to-run-with-ebpf-for-windows/)

In our previous blog, we spoke about the progress we have made for the eBPF…Read more (https://cloudblogs.microsoft.com/opensource/2022/02/22/getting-linux-based-ebpf-programs-to-run-with-ebpf-for-windows/)

Join us at the Build in the Open Happy Hour to connect and collaborate with Open Source enthusiasts › (https://cloudblogs.microsoft.com/opensource/2022/02/07/join-us-at-the-build-in-the-open-happy-hour-to-connect-and-collaborate-with-open-source-enthusiasts/)

Open source has forever changed software development for the better. It has allowed developers from…Read more (https://cloudblogs.microsoft.com/opensource/2022/02/07/join-us-at-the-build-in-the-open-happy-hour-to-connect-and-collaborate-with-open-source-enthusiasts/)

Announcing Azure Active Directory (Azure AD) workload identity for Kubernetes › (https://cloudblogs.microsoft.com/opensource

/2022/01/18/announcing-azure-active-directory-azure-ad-workload-identity-for-kubernetes/)

Today, we are excited to announce an open-source project called Azure AD workload identity for Kubernetes....Read more (https://cloudblogs.microsoft.com/opensource/2022/01/18/announcing-azure-active-directory-azure-ad-workload-identity-for-kubernetes/)

Follow OpenAtMicrosoft

 (https://twitter.com/OpenAtMicrosoft)