



USE CASES

A Brief History of Containers

Learn how containers and microservices are changing the face of software architecture, development, and operations.



Jul 19, 2018

D2IQ
D 2 I Q

6 min read

Containers may seem like a technology that came out of nowhere to transform IT, but in truth they are anything but new. The original idea of a container has been around since the 1970s, when the concept was first employed on Unix systems to better isolate application code. While useful in certain application development and deployment scenarios, the biggest drawback to containers in those early days was the simple fact that they were anything but portable.

Nevertheless, containers evolved, bringing forth abstraction capabilities that are now being broadly applied to make enterprise IT more flexible. Thanks to the rise of Docker containers it's now possible to more easily move workloads between different versions of Linux as well as orchestrate containers to create microservices.

Much like containers, a microservice is not a new idea either. The concept harkens back to service-oriented architectures (SOA). What is different is that microservices based on containers are more granular and simpler to manage. Instead of updating applications by "patching" them, new functionality is added to a microservice by replacing existing containers with ones that incorporate new functionality. Each microservice then exposes an application programming interface (API) that enables developers to programmatically weave them together to construct an application much faster than ever before. That approach also serves to simplify ongoing maintenance of individual components and enhance overall application security.

Obviously, building and maintaining a series of microservices based on containers is going to be more challenging than maintaining a monolithic application. To address that challenge, various platforms for orchestrating containers running in distributed computing environments have emerged.

Only when IT organizations start to appreciate the capabilities those platforms provide does it become clear how containers are on the cusp of becoming the foundation component for a new layer of abstraction that promises to forever change how enterprise IT is managed.

How containers have evolved

Back in the 1970s, early containers created an isolated environment where services and applications could run without interfering with other processes – producing something akin to a sandbox to test applications, services, and other processes. The original idea was to isolate the container's workload from production systems in way that enabled developers to test their applications and processes on production hardware without risking disruption to other services. Over the years, containers have gained the ability to isolate users, files, networking, and more. A container can now also have its own IP address.

Around 2012, containers got a big usability boost, thanks to numerous Linux distributions bringing forth additional deployment options and management tools. Containers running on Linux platforms were transformed into an operating system-level virtualization technology, designed specifically to provide multiple isolated Linux environments on a single Linux host. Linux containers do not require dedicated guest operating systems; instead, they share the host operating system kernel. Elimination of the need for a dedicated operating system also allows containers to spin up much faster than any virtual machine.

Containers were able to use Linux kernel features such as Namespaces, Apparmor, SELinux profiles, chroot and CGroups for creating an isolated operational environment,

RELATED CONTENT



Charting a Course for
Kubernetes Success



How to Execute an End-
to-End as a Service
Solution | D2IQ



The Top 5 Challenges to
Getting Started with
Kubernetes

while Linux security modules added a layer of protection, which promised that access to the host machine and the kernel from containers prevented intrusions. The Linux flavor of containerization also added more flexibility by allowing containers to run different Linux distributions from their host operating system, so long as both operating systems ran on the same CPU architecture.

Linux containers provided the means to create container images based on various Linux distributions, while also incorporating an API for managing the lifecycle of the containers. Linux distributions further included the necessary client tools for interacting with the API, bundled features to take snapshots, and support for migrating container instances from one container host to another.

But while containers running on a Linux platform increased their applicability, there were still several significant challenges to overcome, including unified management, true portability, compatibility, and control of scale.

Clearing the hurdles

Usage of containers on Linux platforms began to significantly advance with the arrival of Apache Mesos, Google Borg, and Facebook Tupperware, all of which brought varying degrees of container orchestration and cluster management capabilities. These platforms provided the ability to spin up hundreds of containers on demand, as well as support for automated failover, and other mission-critical features required to manage containers at scale. But it wasn't until the variant of containers that we now call Docker that the shift to containers began in real earnest.

Docker (both the technology and the company) started off as a public PaaS offering called dotCloud. PaaS solutions benefited from container technology, which brought new capabilities, such as live migrations and updates without interruption. In 2013, dotCloud open-sourced its underlying container technology and called it the Docker Project. Growing support for the Docker Project spawned a large community of container adopters. Soon after, dotCloud became Docker, Inc., which in addition to contributing to the Docker container technology, began to build its own management platform.

The popularity of Docker gave rise to several management platforms, including Marathon, Kubernetes, Docker Swarm, and more broadly, the [DC/OS environment that Mesosphere built](#) on top of Mesos to manage not only on containers, but also broad swath of other legacy applications and data services written in, for example, Java. Although each platform has its own take on how to best deliver orchestration and management, they all have one thing in common – the drive to make containers more mainstream in the enterprise.

However, it takes much more than orchestration and management tools to further adoption of containers into the enterprise. It also takes purpose. In other words, no one should look to containerization unless they understand the problems the technology can solve.

The future of containers

Containers eschew much of that baggage by only creating an abstraction layer between the host OS and the application (or service). That approach allows many more containers to run per physical machine than is possible by deploying containers on top of a virtual machine.

From an architectural standpoint, virtualization is used to abstract hardware, meaning that each virtual machine must have an operating system and all its dependencies – and only then can an application or service be installed and made available to a cloud instance.

Containers are also much smaller than a virtual machine. Because of these issues in time it is expected more containers will be deployed on bare-metal servers. Going forward, it's already been shown that containers will be deployed on much lighter-weight hypervisors to provide greater isolation, without having to incur all the overhead associated with virtual machines deployed on existing hypervisors.

Today, however, most containers are deployed on top of virtual machines running on premises or in a public cloud; or within the context of a PaaS environment. The primary reason for that is most organizations are either uncomfortable with a lack of isolation that containers provide today compared to a virtual machine employing a hypervisor, or they don't yet have the skills or tooling in place to manage containers on bare-metal servers.

In the last year, container usage has dramatically accelerated in both emerging "cloud native" applications as well as in circumstances where IT organizations want to "containerize" an existing legacy application to, for example, more easily lift and shift it into the cloud. As adoption of cloud-native development methodologies mature, it's now really a case of when – not if – containers will become the de facto standard for application delivery.

The challenge facing IT organizations next will be figuring out how best to manage containerized applications alongside all those existing applications running on legacy platforms that are not likely to be retired any time soon.

[< Previous](#)

[Next >](#)



Ready to get started?

[Request a Demo](#)

PRODUCTS

[DKP D2IQ Kubernetes](#)
[Platform](#)
[DKP Essential](#)
[DKP Enterprise](#)
[DKP Edge/IoT](#)
[DKP Gov Essential](#)
[DKP Gov Advanced](#)

SOLUTIONS

[AI/ML](#)
[Air Gapped Deployment](#)
[AKS + D2IQ](#)
[Edge/IoT](#)
[EKS + D2IQ](#)
[Hybrid and Multi-Cloud](#)
[Kubernetes Adoption](#)
[Public Sector](#)
[Smart Cloud Native](#)

RESOURCES

[D2IQ Blog](#)
[D2IQ Engineering Blog](#)
[Events](#)
[Kubernetes Glossary](#)
[Kubernetes Cheatsheet](#)
[Resources](#)

SUPPORT

[Contact Support](#)
[CNCF Certification](#)
[Documentation](#)
[Knowledge Base](#)
[Kubernetes Training](#)
[Service Catalog](#)

COMPANY

[About D2IQ](#)
[Federal](#)
[Contact Us](#)
[News](#)
[Partners](#)

CONNECT

[Facebook](#)
[Instagram](#)
[LinkedIn](#)
[Twitter](#)
[YouTube](#)

© 2025 D2IQ, Inc. All Rights Reserved.

[Terms of Service](#) [Privacy Policy](#) [EULA](#)

[Cookies Settings](#)

