O'REILLY®

Compliments of IBM®

Kubernetes in the Enterprise

Deploying and Operating Production Applications on Kubernetes in Hybrid Cloud Environments

Michael Elder, Jake Kitchener
& Dr. Brad Topol

Build

Kubernetes makes it easy to bind your app to Watson, by relieving the pain around security, scale, and infrastructure management.

Get hands-on experience through tutorials and courses.

ibm.biz/oreillykubernetes

Smart

IBM

Kubernetes in the Enterprise

Deploying and Operating Production Applications on Kubernetes in Hybrid Cloud Environments

Michael Elder, Jake Kitchener, and Dr. Brad Topol

Beijing · Boston · Farnham · Sebastopol · Tokyo O'REILLY®

Kubernetes in the Enterprise
by Michael Elder, Jake Kitchener, and Dr. Brad Topol

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Nicole Tache and Michele Cronin
Production Editor: Melanie Yarbrough
Copyeditor: Octal Publishing, LLC
Proofreader: Sonia Saruba
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

October 2018: First Edition

Revision History for the First Edition
2018-09-27: First Release

978-1-492-04324-9

[LSI]

Table of Contents

Preface

Kubernetes is a cloud infrastructure that provides for the deployment and orchestration of containerized applications. The Kubernetes project is supported by a very active open source community that continues to experience explosive growth. With support from all the major vendors and the myriad contributors of all sizes, Kubernetes has established itself as the de facto standard for cloud-native computing applications.

Although Kubernetes has the potential to dramatically improve the creation and deployment of cloud-native applications in the enterprise, getting started with it in enterprise environments can be difficult. This book is targeted toward developers and operators who are looking to use Kubernetes as their primary approach for creating, managing, deploying, and operating their container-based cloud-native computing applications.

The book is structured so that developers and operators who are new to Kubernetes can use it to gain a solid understanding of Kubernetes fundamental concepts. In addition, for experienced practitioners who already have a significant understanding of Kubernetes, this book provides several chapters focused on the creation of enterprise-quality Kubernetes applications in private, public, and hybrid cloud environments. It also brings developers and operators up to speed on key aspects of production-level cloud-native enterprise applications such as continuous delivery, log collection and analysis, security, scheduling, autoscaling, networking, storage, audit, and compliance. Additionally, this book provides an overview of several helpful resources and approaches that enable you to quickly become a contributor to Kubernetes.

Chapter 1 provides an overview of both containers and Kubernetes. It then discusses the Cloud Native Computing Foundation (CNCF) and the ecosystem growth that has resulted from its open governance model and conformance certification efforts. In Chapter 2, we provide an overview of Kubernetes architecture, describe several ways to run Kubernetes, and introduce many of its fundamental constructs including Pods, ReplicaSets, and Deployments. Chapter 3 covers more advanced Kubernetes capabilities such as load balancing, volume support, and configuration primitives such as ConfigMaps and Secrets, StatefulSets, and DaemonSets. Chapter 4 provides a description of our production application that serves as our enterprise Kubernetes workload. In Chapter 5, we present an overview of continuous delivery approaches that are popular for enterprise applications. Chapter 6 focuses on the operation of enterprise applications, examining issues such as log collection and analysis and health management of your microservices. Chapter 7 provides in-depth coverage of operating Kubernetes environments and addresses topics such as access control, autoscaling, networking, storage, and their implications on hybrid cloud environments. We offer a discussion of the Kubernetes developer experience in Chapter 8. Finally, in Chapter 9, we conclude with a discussion of areas for future growth in Kubernetes.

CHAPTER 1

An Introduction to Containers and Kubernetes

In this first chapter, we begin with a historical background of the origin of both containers and Kubernetes. We then describe the creation of the Cloud Native Computing Foundation and the role it has played in the explosive growth of Kubernetes and its ecosystem. We conclude this chapter with an overview of Kubernetes Conformance Certification initiatives, which are critical to ensuring Kubernetes interoperability, supporting portable workloads, and maintaining a cohesive open source ecosystem.

The Rise of Containers

In 2012, the foundation of most cloud environments was a virtualization infrastructure that provided users with the ability to instantiate multiple virtual machines (VMs). The VMs could attach volume storage and execute on cloud infrastructures that supported a variety of network virtualization options. These types of cloud environments could provision distributed applications such as web service stacks much more quickly than was previously possible. Before the availability of these types of cloud infrastructures, if an application developer wanted to build a web application, they typically waited weeks for the infrastructure team to install and configure web servers and database and provide network routing between the new machines. In contrast, these same application developers could utilize the new cloud environments to self-provision the same application infrastructure in less than a day. Life was good.

Although the new VM-based cloud environments were a huge step in the right direction, they did have some notable inefficiencies. For example, VMs could take a long time to start, and taking a snapshot of the VM could take a significant amount of time as well. In addition, each VM typically required a large number of resources, and this limited the ability to fully exploit the utilization of the physical servers hosting the VMs.

At Pycon in March of 2013, Solomon Hykes presented an approach for deploying web applications to a cloud that did not rely on VMs. Instead, Solomon demonstrated how Linux containers could be used to create a self-contained unit of deployable software. This new unit of deployable software was aptly named a *container*. Instead of providing isolation at a VM level, isolation for the container unit of software was provided at the process level. The process running in the container was given its own isolated file system and was allocated network connectivity. Solomon announced that the software they created to run applications in containers was called *Docker*, and would be made available as an open source project.

For many cloud application developers that were accustomed to deploying VM-based applications, their initial experience with Docker containers was mind-blowing. When using VMs, deploying an application by instantiating a VM could easily take several minutes. In contrast, deploying a Docker container image took just a few seconds. This dramatic improvement in performance was because instantiating a Docker image is more akin to starting a new process on a Linux machine. This is a fairly lightweight operation, especially when compared to instantiating a whole new VM.

Container images also showed superior performance when a cloud application developer wanted to make changes to a VM image and snapshot a new version. This operation was typically a very time-consuming process because it required the entire VM disk file to be written out. With Docker containers, a multilayered filesystem is used instead. If changes are made in this situation, they are captured as changes to the filesystem and represented by a new filesystem layer. Because of this, a Docker container image could snapshot a new version by writing out only the changes to the filesystem as a new filesystem layer. In many cases, the amount of changes to the filesystem for a new container image are quite small and thus the snapshot operation is extremely efficient. For many cloud application developers who started experimenting with containers, it quickly became obvious that this new approach had tremendous potential to improve the current state of the art for deploying applications in clouds.

There was still one issue holding back the adoption of container images: the perception that it was not possible to run enterprise middleware as container images. Advanced prototyping initiatives took place to investigate the difficulty of running these images. It was proven quickly that developers could successfully run enterprise middleware such as WebSphere Liberty, and Db2 Express as Docker container images. Sometimes, a few changes were necessary or perhaps a Linux kernel upgrade was required, but in general the Docker container image approach was proven to be suitable for running enterprise middleware.

The container approach for deploying web applications experienced significant growth in a short period, and it was soon supported on a variety of cloud platforms. Here is a summary of the key advantages of using the container-image approach over VM images for deploying software to cloud-based environments:

Container image startup is much faster than VM image startup
: Starting a container image is essentially the equivalent of starting a new process. In contrast, starting a VM image involves first booting an operating system (OS) and related services and is much more time consuming,

Capturing a new container image snapshot is much faster than a VM snapshot operation
: Containers utilize a layered filesystem and any changes to the filesystem are written as a new layer. With container images, capturing a new snapshot of the container image requires writing out only the new updates to the filesystem that the process running in the container has created. When performing a snapshot of a VM image instance, the entire VM disk file must be written out, and this is typically an extremely time-consuming process.

Container images are much smaller than VM images
: A typical container image is portrayed in megabytes, whereas a VM image is most commonly portrayed in gigabytes.

Build once, run anywhere
: Docker enabled developers to build container images on their laptops, test them, and then deploy to the cloud knowing that not only the same code would be running in the cloud, but the entire runtime would be a bit-for-bit copy. Oftentimes with virtualization and traditional Platform as a Service (PaaS), developers test on one runtime configuration on their local system but don't have control over the cloud runtime. This leads to reduced confidence and more test requirements.

Better resource utilization
: Because container images are much smaller in size and are at the process level, they take up fewer resources than a VM. As a result, it is possible to put a larger number of containers on a physical server than is possible when placing VMs on a physical server.

In the next section, we provide a background on Kubernetes, which is a platform for the management and orchestration of container images.

Kubernetes Arrives to Provide an Orchestration and Management Infrastructure for Containers

As previously discussed, Docker was responsible for introducing developers to the concept of container-based applications. Docker provided very consumable tooling for container development and storage of containers in registries. However, Docker was not the only company with experience using container-based applications in cloud environments.

For more than a decade, Google had embraced the use of Linux containers as the foundation for applications deployed in its cloud.[1] Google had extensive experience orchestrating and managing containers at scale and had developed three generations of container management systems: Borg, Omega, and Kubernetes. Kubernetes was the latest generation of container management developed by

1 Brendan Burns et al., "Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade". *ACM Queue* 14 (2016): 70–93.

Google. It was a redesign based upon lessons learned from Borg and Omega, and was made available as an open source project. Kubernetes delivered several key features that dramatically improved the experience of developing and deploying a scalable container-based cloud application:

Declarative deployment model
: Most cloud infrastructures that existed before Kubernetes was released provided a procedural approach based on a scripting language such as Ansible, Chef, Puppet, and so on for automating deployment activities. In contrast, Kubernetes used a declarative approach of describing what the desired state of the system should be. Kubernetes infrastructure was then responsible for starting new containers when necessary (e.g., when a container failed) to achieve the desired declared state. The declarative model was much clearer at communicating what deployment actions were desired, and this approach was a huge step forward compared to trying to read and interpret a script to determine what the desired deployment state should be.

Built-in replica and autoscaling support
: In some cloud infrastructures that existed before Kubernetes, support for replicas of an application and providing autoscaling capabilities were not part of the core infrastructure and, in some cases, never successfully materialized. These capabilities were provided as core features in Kubernetes, which dramatically improved the robustness and consumability of its orchestration capabilities.

Improved networking model
: Kubernetes mapped a single IP address to a *Pod*, which is Kubernetes' smallest unit of container aggregation and management. This approach aligned the network identity with the application identity and simplified running software on Kubernetes.[2]

2 Brendan Burns et al., "Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade". *ACM Queue* 14 (2016): 70–93.

Built-in health-checking support
: Kubernetes provided container health checking and monitoring capabilities that reduced the complexity of identifying when failures occur.

Even with all the innovative capabilities available in Kubernetes, enterprise companies were still reticent to adopt a technology that is an open source project supported by a single vendor, especially when other alternatives for container orchestration such as Docker Swarm were available. Enterprise companies would have been much more willing to adopt Kubernetes if it were instead a multiple-vendor and meritocracy-based open source project backed by a solid governance policy and a level playing field for contributing. In 2015, the Cloud Native Computing Foundation was formed to address these issues.

The Cloud Native Computing Foundation Tips the Scale for Kubernetes

In 2015, the Linux Foundation initiated the creation of the Cloud Native Computing Foundation (CNCF).[3] The CNCF's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self-healing multitenant nodes.[4] In support of this new foundation, Google donated Kubernetes to the CNCF to serve as its seed technology. With Kubernetes serving as the core of its ecosystem, the CNCF has grown to more than 250 member companies, including Google Cloud, IBM Cloud, Amazon Web Services (AWS), Docker, Microsoft Azure, Red Hat, VMware, Intel, Huawei, Cisco, Alibaba Cloud, and many more.[5] In addition, the CNCF ecosystem has grown to hosting 17 open source projects, including Prometheus, Envoy, GRPC, and many others. Finally, the CNCF also nurtures several early stage projects and has eight projects accepted into its Sandbox program for emerging technologies.

3 Vaughan-Nicholls, Steven J. (2015-07-21). "Cloud Native Computing Foundation seeks to forge cloud and container unity", ZDNet.

4 Check out the "Cloud Native Computing Foundation ("CNCF") Charter" on the Cloud Native Computing Foundation website.

5 See the list of members on the Cloud Native Computing Foundation website.

With the weight of the vendor-neutral CNCF foundation behind it, Kubernetes has grown to have more than 2,300 contributors from a wide range of industries. In addition to hosting several cloud-native projects, the CNCF provides training, a Technical Oversight Board, a Governing Board, a community infrastructure lab, and several certification programs. In the next section, we describe CNCF's highly successful Kubernetes Conformance Certification, which is focused on improving Kubernetes interoperability and workload portability.

CNCF Kubernetes Conformance Certification Keeps the Focus on User Needs

A key selling point for any open source project is that different vendor distributions of the open source project are interoperable. Customers are very concerned about vendor lock-in: being able to easily change the vendor that provides a customer their open source infrastructure is crucial. In the context of Kubernetes, it needs to be easy for the customer to move its Kubernetes workloads from one vendor's Kubernetes platform to a different vendor's Kubernetes platform. In a similar fashion, a customer might have a workload that normally runs on an on-premises Kubernetes private cloud, but during holiday seasons, the workload might merit obtaining additional resources on a public Kubernetes cloud as well. For all these reasons, it is absolutely critical that Kubernetes platforms from different vendors be interoperable and that workloads are easily portable to different Kubernetes environments.

Fortunately, the CNCF identified this critical requirement early on in the Kubernetes life cycle before any serious forks in the Kubernetes distributions had occurred. The CNCF formed the Kubernetes Conformance Certification Workgroup. The mission of the Conformance Certification Workgroup is to provide a software conformance program and test suite that any Kubernetes implementation can use to demonstrate that it is conformant and interoperable.

As of this writing, 60 vendor distributions had successfully passed the Kubernetes Conformance Certification Tests. The Kubernetes Conformance Workgroup continues to make outstanding progress, focusing on topics such as increased conformance test coverage, automated conformance reference test documentation generation, and was even a major highlight of the KubeCon Austin 2017 Keynote presentation.

Summary

This chapter discussed a variety of factors that have contributed to Kubernetes becoming the de facto standard for the orchestration and management of cloud-native computing applications. Its declarative model, built-in support for autoscaling, improved networking model, health-check support, and the backing of the CNCF have resulted in a vibrant and growing ecosystem for Kubernetes with adoption across cloud applications and high-performance computing domains. In Chapter 2, we begin our deeper exploration into the architecture and capabilities of Kubernetes.