O'REILLY®

Cloud Native DevOps with Kubernetes

Building, Deploying, and Scaling Modern Applications in the Cloud

Grayscale Edition

For Sale in the Indian Subcontinent & Select Countries Only*

*Refer Back Cover

SPD

John Arundel & Justin Domingus

Praise for *Cloud Native DevOps with Kubernetes*

Cloud Native DevOps is an essential guide to operating today's distributed systems. A super clear and informative read, covering all the details without compromising readability. I learned a lot, and definitely have some action points to take away!

—*Will Thames, Platform Engineer, Skedulo*

The most encompassing, definitive, and practical text about the care and feeding of Kubernetes infrastructure. An absolute must-have.

—*Jeremy Yates, SRE Team, The Home Depot QuoteCenter*

I wish I'd had this book when I started! This is a must-read for everyone developing and running applications in Kubernetes.

—*Paul van der Linden, Lead Developer, vdL Software Consultancy*

This book got me really excited. It's a goldmine of information for anyone looking to use Kubernetes, and I feel like I've levelled up!

—*Adam McPartlan (@mcparty), Senior Systems Engineer, NYnet*

I really enjoyed reading this book. It's very informal in style, but authoritative at the same time. It contains lots of great practical advice. Exactly the sort of information that everybody wants to know, but doesn't know how to get, other than through first-hand experience.

—*Nigel Brown, cloud native trainer and course author*

Cloud Native DevOps with Kubernetes

Building, Deploying, and Scaling Modern Applications in the Cloud

John Arundel and Justin Domingus

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

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.
Mumbai Bangalore Kolkata New Delhi

Cloud Native DevOps with Kubernetes

by John Arundel and Justin Domingus

Copyright © 2019 John Arundel and Justin Domingus. All rights reserved. ISBN: 978-1-492-04076-7
Originally printed in the United States of America.

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://safari.oreilly.com)*. For more information, contact our corporate/ institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Rachel Roumeliotis
Production Editor: Nan Barber
Proofreader: Amanda Kersey
Interior Designer: David Futato
Proofreader: Rebecca Demarest
Developmental Editors: Virginia Wilson and Nikki McDonald
Copyeditor: Kim Cofer
Indexer: Judith McConville
Cover Designer: Karen Montgomery

Printing History:
February 2019: First Edition

Revision History for the First Edition 2019-01-24: First Release, 2019-03-08: Second Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492040767 for release details.

First Indian Reprint: April 2019

ISBN: 978-93-5213-809-8

Published by **Shroff Publishers & Distributors Pvt. Ltd.** B-103, Railway Commercial Complex, Sector 3, Sanpada (E), Navi Mumbai 400705 • TEL: (91 22) 4158 4158 • FAX: (91 22) 4158 4141 E-mail:spdorders@shroffpublishers.com•Web:www.shroffpublishers.com Printed at Jasmine Art Printers Pvt. Ltd., Mumbai.

Table of Contents

Foreword

Welcome to *Cloud Native DevOps with Kubernetes*.

Kubernetes is a real industry revolution. Just a brief look at the Cloud Native Computing Foundation's Landscape (*https://landscape.cncf.io/*), which contains data about more than 600 projects that exist in the cloud native world today, highlights the importance of Kubernetes these days. Not all these tools were developed for Kubernetes, not all of them can even be used with Kubernetes, but all of them are part of the huge ecosystem where Kubernetes is one of the flagship technologies.

Kubernetes changed the way applications are developed and operated. It's a core component in the DevOps world today. Kubernetes brings flexibility to developers and freedom to operations. Today you can use Kubernetes on any major cloud provider, on bare-metal on-premises environments, as well as on a local developer's machine. Stability, flexibility, a powerful API, open code, and an open developer community are a few reasons why Kubernetes became an industry standard, just as Linux is a standard in the world of operating systems.

Cloud Native DevOps with Kubernetes is a great handbook for people who are performing their day-to-day activities with Kubernetes or are just starting their Kubernetes journey. John and Justin cover all the major aspects of deploying, configuring, and operating Kubernetes and the best practices for developing and running applications on it. They also give a great overview of the related technologies, including Prometheus, Helm, and continuous deployment. This is a must-read book for everyone in the DevOps world.

Kubernetes is not just yet another exciting tool; it is an industry standard and the foundation for next-generation technologies including serverless (OpenFaaS, Knative) and machine learning (Kubeflow) tools. The entire IT industry is changing because of the cloud native revolution, and it's hugely exciting to be living through it.

— Ihor Dvoretskyi
Developer Advocate, Cloud Native Computing Foundation
December 2018

Preface

In the world of IT operations, the key principles of DevOps have become well understood and widely adopted, but now the landscape is changing. A new application platform called Kubernetes has become rapidly adopted by companies all over the world and in all kinds of different industries. As more and more applications and businesses move from traditional servers to the Kubernetes environment, people are asking how to do DevOps in this new world.

This book explains what DevOps means in a cloud native world where Kubernetes is the standard platform. It will help you select the best tools and frameworks from the Kubernetes ecosystem. It will also present a coherent way to use those tools and frameworks, offering battle-tested solutions that are running right now, in production, for real.

What Will I Learn?

You'll learn what Kubernetes is, where it comes from, and what it means for the future of software development and operations. You'll learn how containers work, how to build and manage them, and how to design cloud native services and infrastructure.

You'll understand the trade-offs between building and hosting Kubernetes clusters yourself, and using managed services. You'll learn the capabilities, limitations, and pros and cons of popular Kubernetes installation tools such as kops, kubeadm, and Kubespray. You'll get an informed overview of the major managed Kubernetes offerings from the likes of Amazon, Google, and Microsoft.

You'll get hands-on practical experience of writing and deploying Kubernetes applications, configuring and operating Kubernetes clusters, and automating cloud infrastructure and deployments with tools such as Helm. You'll learn about Kubernetes support for security, authentication, and permissions, including Role-Based Access

Control (RBAC), and best practices for securing containers and Kubernetes in production.

You'll learn how to set up continuous integration and deployment with Kubernetes, how to back up and restore data, how to test your cluster for conformance and reliability, how to monitor, trace, log, and aggregate metrics, and how to make your Kubernetes infrastructure scalable, resilient, and cost-effective.

To illustrate all the things we talk about, we apply them to a very simple demo application. You can follow along with all our examples using the code from our Git repo.

Who Is This Book For?

This book is most directly relevant to IT operations staff responsible for servers, applications, and services, and developers responsible for either building new cloud native services, or migrating existing applications to Kubernetes and cloud. We assume no prior knowledge of Kubernetes or containers—don't worry, we'll walk you through all that.

Experienced Kubernetes users should still find much valuable material in the book: it covers advanced topics such as RBAC, continuous deployment, secrets management, and observability. Whatever your level of expertise, we hope you'll find something useful in these pages.

What Questions Does This Book Answer?

In planning and writing this book, we spoke to hundreds of people about cloud native and Kubernetes, ranging from industry leaders and experts to complete beginners. Here are some of the questions they said they wanted a book like this to answer:

- "I'd like to learn why I should invest my time in this technology. What problems will it help to solve for me and my team?"
- "Kubernetes seems great, but it's quite a steep learning curve. Setting up a quick demo is easy, but operating and troubleshooting it seems daunting. We'd like some solid guidance on how people are running Kubernetes clusters in the real world, and what problems we're likely to encounter."
- "Opinionated advice would be useful. The Kubernetes ecosystem has too many options for beginning teams to choose between. When there are multiple ways of doing the same thing, which one is best? How do we choose?"

And perhaps the most important question of all:

- "How do I use Kubernetes without breaking my company?"

We kept these questions, and many others, firmly in mind while writing this book, and we've done our level best to answer them. How did we do? Turn the page to find out.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/cloudnativedevops/demo*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Cloud Native DevOps with Kubernetes* by John Arundel and Justin Domingus (O'Reilly). Copyright 2019 John Arundel and Justin Domingus, 978-1-492-04076-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning Platform

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/cloud-nat-dev-ops*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Our grateful thanks are due to the many people who read early drafts of this book and gave us invaluable feedback and advice, or assisted us in other ways, including (but not limited to) Abby Bangser, Adam J. McPartlan, Adrienne Domingus, Alexis Richardson, Aron Trauring, Camilla Montonen, Gabriell Nascimento, Hannah Klemme, Hans Findel, Ian Crosby, Ian Shaw, Ihor Dvoretskyi, Ike Devolder, Jeremy Yates, Jérôme Petazzoni, Jessica Deen, John Harris, Jon Barber, Kitty Karate, Marco Lancini, Mark Ellens, Matt North, Michel Blanc, Mitchell Kent, Nicolas Steinmetz, Nigel Brown, Patrik Duditš, Paul van der Linden, Philippe Ensarguet, Pietro Mamberti, Richard Harper, Rick Highness, Sathyajith Bhat, Suresh Vishnoi, Thomas Liakos, Tim McGinnis, Toby Sullivan, Tom Hall, Vincent De Smet, and Will Thames.

CHAPTER 1

Revolution in the Cloud

There was never a time when the world began, because it goes round and round like a circle, and there is no place on a circle where it begins.

—Alan Watts

There's a revolution going on. Actually, three revolutions.

The first revolution is the creation of the cloud, and we'll explain what that is and why it's important. The second is the dawn of DevOps, and you'll find out what that involves and how it's changing operations. The third revolution is the coming of containers. Together, these three waves of change are creating a new software world: the *cloud native* world. The operating system for this world is called Kubernetes.

In this chapter, we'll briefly recount the history and significance of these revolutions, and explore how the changes are affecting the way we all deploy and operate software. We'll outline what *cloud native* means, and what changes you can expect to see in this new world if you work in software development, operations, deployment, engineering, networking, or security.

Thanks to the effects of these interlinked revolutions, we think the future of computing lies in cloud-based, containerized, distributed systems, dynamically managed by automation, on the Kubernetes platform (or something very like it). The art of developing and running these applications—*cloud native DevOps*—is what we'll explore in the rest of this book.

If you're already familiar with all of this background material, and you just want to start having fun with Kubernetes, feel free to skip ahead to Chapter 2. If not, settle down comfortably, with a cup of your favorite beverage, and we'll begin.

The Creation of the Cloud

In the beginning (well, the 1960s, anyway), computers filled rack after rack in vast, remote, air-conditioned data centers, and users would never see them or interact with them directly. Instead, developers submitted their jobs to the machine remotely and waited for the results. Many hundreds or thousands of users would all share the same computing infrastructure, and each would simply receive a bill for the amount of processor time or resources she used.

It wasn't cost-effective for each company or organization to buy and maintain its own computing hardware, so a business model emerged where users would share the computing power of remote machines, owned and run by a third party.

If that sounds like right now, instead of last century, that's no coincidence. The word *revolution* means "circular movement," and computing has, in a way, come back to where it began. While computers have gotten a lot more powerful over the years—today's Apple Watch is the equivalent of about three of the mainframe computers shown in Figure 1-1—shared, pay-per-use access to computing resources is a very old idea. Now we call it *the cloud*, and the revolution that began with timesharing mainframes has come full circle.

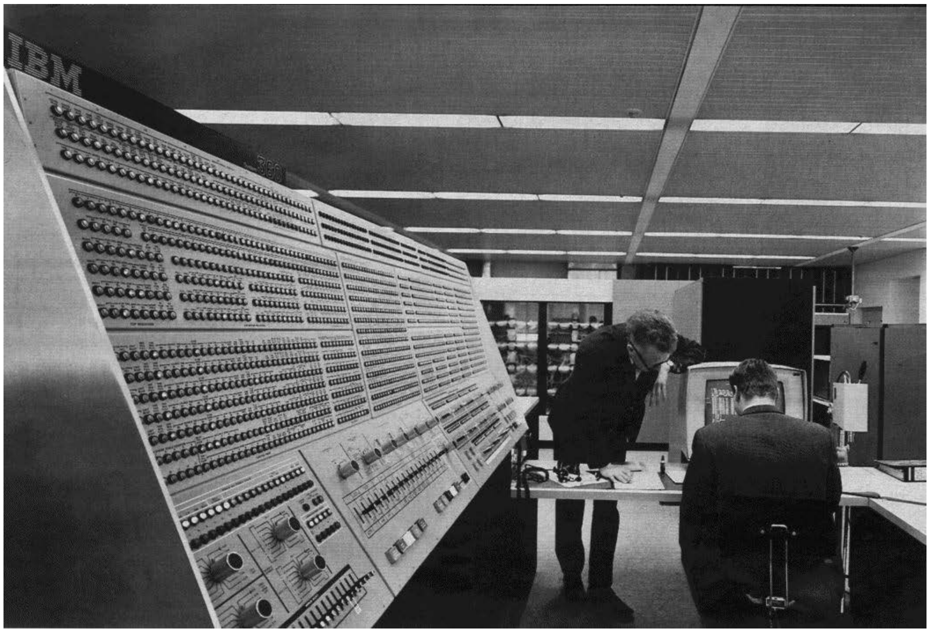

Figure 1-1. Early cloud computer: the IBM System/360 Model 91, at NASA's Goddard Space Flight Center

Buying Time

The central idea of the cloud is this: instead of buying a *computer*, you buy *compute*. That is, instead of sinking large amounts of capital into physical machinery, which is hard to scale, breaks down mechanically, and rapidly becomes obsolete, you simply buy time on someone else's computer, and let them take care of the scaling, maintenance, and upgrading. In the days of bare-metal machines—the "Iron Age", if you like—computing power was a capital expense. Now it's an operating expense, and that has made all the difference.

The cloud is not just about remote, rented computing power. It is also about distributed systems. You may buy raw compute resource (such as a Google Compute instance, or an AWS Lambda function) and use it to run your own software, but increasingly you also rent *cloud services*: essentially, the use of someone else's software. For example, if you use PagerDuty to monitor your systems and alert you when something is down, you're using a cloud service (sometimes called *software as a service*, or SaaS).

Infrastructure as a Service

When you use cloud infrastructure to run your own services, what you're buying is *infrastructure as a service* (IaaS). You don't have to expend capital to purchase it, you don't have to build it, and you don't have to upgrade it. It's just a commodity, like electricity or water. Cloud computing is a revolution in the relationship between businesses and their IT infrastructure.

Outsourcing the hardware is only part of the story; the cloud also allows you to outsource the *software* that you don't write: operating systems, databases, clustering, replication, networking, monitoring, high availability, queue and stream processing, and all the myriad layers of software and configuration that span the gap between your code and the CPU. Managed services can take care of almost all of this *undifferentiated heavy lifting* for you (you'll find out more about the benefits of managed services in Chapter 3).

The revolution in the cloud has also triggered another revolution in the people who use it: the DevOps movement.

The Dawn of DevOps

Before DevOps, developing and operating software were essentially two separate jobs, performed by two different groups of people. *Developers* wrote software, and they passed it on to *operations* staff, who ran and maintained the software *in production* (that is to say, serving real users, instead of merely running under test conditions). Like computers that need their own floor of the building, this separation has its roots in the middle of the last century. Software development was a very specialist job, and so was computer operation, and there was very little overlap between the two.

Indeed, the two departments had quite different goals and incentives, which often conflicted with each other (Figure 1-2). Developers tend to be focused on shipping new features quickly, while operations teams care about making services stable and reliable over the long term.

Figure 1-2. Separate teams can lead to conflicting incentives (photo by Dave Roth)

When the cloud came on the horizon, things changed. Distributed systems are complex, and the internet is very big. The technicalities of operating the system—recovering from failures, handling timeouts, smoothly upgrading versions—are not so easy to separate from the design, architecture, and implementation of the system.

Further, "the system" is no longer just your software: it comprises in-house software, cloud services, network resources, load balancers, monitoring, content distribution networks, firewalls, DNS, and so on. All these things are intimately interconnected and interdependent. The people who write the software have to understand how it relates to the rest of the system, and the people who operate the system have to understand how the software works—or fails.

The origins of the DevOps movement lie in attempts to bring these two groups together: to collaborate, to share understanding, to share responsibility for systems reliability and software correctness, and to improve the scalability of both the software systems and the teams of people who build them.

Nobody Understands DevOps

DevOps has occasionally been a controversial idea, both with people who insist it's nothing more than a modern label for existing good practice in software development, and with those who reject the need for greater collaboration between development and operations.

There is also widespread misunderstanding about what DevOps actually is: A job title? A team? A methodology? A skill set? The influential DevOps writer John Willis has identified four key pillars of DevOps, which he calls culture, automation, measurement, and sharing (CAMS). Another way to break it down is what Brian Dawson has called the DevOps trinity: people and culture, process and practice, and tools and technology.

Some people think that cloud and containers mean that we no longer need DevOps—a point of view sometimes called *NoOps*. The idea is that since all IT operations are outsourced to a cloud provider or another third-party service, businesses don't need full-time operations staff.

The NoOps fallacy is based on a misapprehension of what DevOps work actually involves:

> With DevOps, much of the traditional IT operations work happens before code reaches production. Every release includes monitoring, logging, and A/B testing. CI/CD pipelines automatically run unit tests, security scanners, and policy checks on every commit. Deployments are automatic. Controls, tasks, and non-functional requirements are now implemented before release instead of during the frenzy and aftermath of a critical outage.
>
> —Jordan Bach (AppDynamics (*https://blog.appdynamics.com/engineering/is-noops-the-end-of-devops-think-again/*))

The most important thing to understand about DevOps is that it is primarily an organizational, human issue, not a technical one. This accords with Jerry Weinberg's *Second Law of Consulting*:

> No matter how it looks at first, it's always a people problem.
>
> —Gerald M. Weinberg, *Secrets of Consulting*

The Business Advantage

From a business point of view, DevOps has been described as "improving the quality of your software by speeding up release cycles with cloud automation and practices, with the added benefit of software that actually stays up in production" (The Register (*https://www.theregister.co.uk/2018/03/06/what_does_devops_do_to_decades_old_planning_processes_and_assumptions*)).

Adopting DevOps requires a profound cultural transformation for businesses, which needs to start at the executive, strategic level, and propagate gradually to every part of the organization. Speed, agility, collaboration, automation, and software quality are key goals of DevOps, and for many companies that means a major shift in mindset.

But DevOps works, and studies regularly suggest that companies that adopt DevOps principles release better software faster, react better and faster to failures and problems, are more agile in the marketplace, and dramatically improve the quality of their products:

> DevOps is not a fad; rather it is the way successful organizations are industrializing the delivery of quality software today and will be the new baseline tomorrow and for years to come.
>
> —Brian Dawson (Cloudbees), Computer Business Review (*https://www.cbronline.com/enterprise-it/applications/devops-fad-stay*)

Infrastructure as Code

Once upon a time, developers dealt with software, while operations teams dealt with hardware and the operating systems that run on that hardware.

Now that hardware is in the cloud, everything, in a sense, is software. The DevOps movement brings software development skills to operations: tools and workflows for rapid, agile, collaborative building of complex systems. Inextricably entwined with DevOps is the notion of *infrastructure as code*.

Instead of physically racking and cabling computers and switches, cloud infrastructure can be automatically provisioned by software. Instead of manually deploying and upgrading hardware, operations engineers have become the people who write the software that automates the cloud.

The traffic isn't just one-way. Developers are learning from operations teams how to anticipate the failures and problems inherent in distributed, cloud-based systems, how to mitigate their consequences, and how to design software that degrades gracefully and fails safe.

Learning Together

Both development teams and operations teams are also learning how to work together. They're learning how to design and build systems, how to monitor and get feedback on systems in production, and how to use that information to improve the systems. Even more importantly, they're learning to improve the experience for their users, and to deliver better value for the business that funds them.

The massive scale of the cloud and the collaborative, code-centric nature of the DevOps movement have turned operations into a software problem. At the same time, they have also turned software into an operations problem, all of which raises these questions:

- How do you deploy and upgrade software across large, diverse networks of different server architectures and operating systems?
- How do you deploy to distributed environments, in a reliable and reproducible way, using largely standardized components?

Enter the third revolution: the container.

The Coming of Containers

To deploy a piece of software, you need not only the software itself, but its *dependencies*. That means libraries, interpreters, subpackages, compilers, extensions, and so on.

You also need its *configuration*. Settings, site-specific details, license keys, database passwords: everything that turns raw software into a usable service.

The State of the Art

Earlier attempts to solve this problem include using *configuration management* systems, such as Puppet or Ansible, which consist of code to install, run, configure, and update the shipping software.

Alternatively, some languages provide their own packaging mechanism, like Java's JAR files, or Python's eggs, or Ruby's gems. However, these are language-specific, and don't entirely solve the dependency problem: you still need a Java runtime installed before you can run a JAR file, for example.

Another solution is the *omnibus package*, which, as the name suggests, attempts to cram everything the application needs inside a single file. An omnibus package contains the software, its configuration, its dependent software components, *their* configuration, *their* dependencies, and so on. (For example, a Java omnibus package would contain the Java runtime as well as all the JAR files for the application.)

Some vendors have even gone a step further and included the entire computer system required to run it, as a *virtual machine image*, but these are large and unwieldy, time-consuming to build and maintain, fragile to operate, slow to download and deploy, and vastly inefficient in performance and resource footprint.

From an operations point of view, not only do you need to manage these various kinds of packages, but you also need to manage a fleet of servers to run them on.

Servers need to be provisioned, networked, deployed, configured, kept up to date with security patches, monitored, managed, and so on.

This all takes a significant amount of time, skill, and effort, just to provide a platform to run software on. Isn't there a better way?

Thinking Inside the Box

To solve these problems, the tech industry borrowed an idea from the shipping industry: the *container*. In the 1950s, a truck driver named Malcolm McLean (*https://hbs.me/2Q0QCzb*) proposed that, instead of laboriously unloading goods individually from the truck trailers that brought them to the ports and loading them onto ships, trucks themselves simply be loaded onto the ship—or rather, the truck bodies.

A truck trailer is essentially a big metal box on wheels. If you can separate the box—the container—from the wheels and chassis used to transport it, you have something that is very easy to lift, load, stack, and unload, and can go right onto a ship or another truck at the other end of the voyage (Figure 1-3).

McLean's container shipping firm, Sea-Land, became very successful by using this system to ship goods far more cheaply, and containers quickly caught on (*https://www.freightos.com/the-history-of-the-shipping-container*). Today, hundreds of millions of containers are shipped every year, carrying trillions of dollars worth of goods.

Figure 1-3. Standardized containers dramatically cut the cost of shipping bulk goods (photo by Pixabay (https://www.pexels.com/@pixabay), licensed under Creative Commons 2.0)

Putting Software in Containers

The software container is exactly the same idea: a standard packaging and distribution format that is generic and widespread, enabling greatly increased carrying capacity, lower costs, economies of scale, and ease of handling. The container format contains everything the application needs to run, baked into an *image file* that can be executed by a *container runtime*.

How is this different from a virtual machine image? That, too, contains everything the application needs to run—but a lot more besides. A typical virtual machine image is around 1 GiB.[1] A well-designed container image, on the other hand, might be a hundred times smaller.

Because the virtual machine contains lots of unrelated programs, libraries, and things that the application will never use, most of its space is wasted. Transferring VM images across the network is far slower than optimized containers.

Even worse, virtual machines are *virtual*: the underlying physical CPU effectively implements an *emulated* CPU, which the virtual machine runs on. The virtualization layer has a dramatic, negative effect on performance (*https://www.stratoscale.com/blog/containers/running-containers-on-bare-metal/*): in tests, virtualized workloads run about 30% slower than the equivalent containers.

In comparison, containers run directly on the real CPU, with no virtualization overhead, just as ordinary binary executables do.

And because containers only hold the files they need, they're much smaller than VM images. They also use a clever technique of addressable filesystem *layers*, which can be shared and reused between containers.

For example, if you have two containers, each derived from the same Debian Linux base image, the base image only needs to be downloaded once, and each container can simply reference it.

The container runtime will assemble all the necessary layers and only download a layer if it's not already cached locally. This makes very efficient use of disk space and network bandwidth.

Plug and Play Applications

Not only is the container the unit of deployment and the unit of packaging; it is also the unit of *reuse* (the same container image can be used as a component of many

1 The *gibibyte* (GiB) is the International Electrotechnical Commission (IEC) unit of data, defined as 1,024 *mebibytes* (MiB). We'll use IEC units (GiB, MiB, KiB) throughout this book to avoid any ambiguity.

different services), the unit of *scaling*, and the unit of *resource allocation* (a container can run anywhere sufficient resources are available for its own specific needs).

Developers no longer have to worry about maintaining different versions of the software to run on different Linux distributions, against different library and language versions, and so on. The only thing the container depends on is the operating system kernel (Linux, for example).

Simply supply your application in a container image, and it will run on any platform that supports the standard container format and has a compatible kernel.

Kubernetes developers Brendan Burns and David Oppenheimer put it this way in their paper "Design Patterns for Container-based Distributed Systems" (*https://www.usenix.org/node/196347*):

> By being hermetically sealed, carrying their dependencies with them, and providing an atomic deployment signal ("succeeded"/"failed"), [containers] dramatically improve on the previous state of the art in deploying software in the datacenter or cloud. But containers have the potential to be much more than just a better deployment vehicle—we believe they are destined to become analogous to objects in object-oriented software systems, and as such will enable the development of distributed system design patterns.

Conducting the Container Orchestra

Operations teams, too, find their workload greatly simplified by containers. Instead of having to maintain a sprawling estate of machines of various kinds, architectures, and operating systems, all they have to do is run a *container orchestrator*: a piece of software designed to join together many different machines into a *cluster*: a kind of unified compute substrate, which appears to the user as a single very powerful computer on which containers can run.

The terms *orchestration* and *scheduling* are often used loosely as synonyms. Strictly speaking, though, *orchestration* in this context means coordinating and sequencing different activities in service of a common goal (like the musicians in an orchestra). *Scheduling* means managing the resources available and assigning workloads where they can most efficiently be run. (Not to be confused with scheduling in the sense of *scheduled jobs*, which execute at preset times.)

A third important activity is *cluster management*: joining multiple physical or virtual servers into a unified, reliable, fault-tolerant, apparently seamless group.

The term *container orchestrator* usually refers to a single service that takes care of scheduling, orchestration, and cluster management.

Containerization (using containers as your standard method of deploying and running software) offered obvious advantages, and a de facto standard container format has made possible all kinds of economies of scale. But one problem still stood in the way of the widespread adoption of containers: the lack of a standard container orchestration system.

As long as several different tools for scheduling and orchestrating containers competed in the marketplace, businesses were reluctant to place expensive bets on which technology to use. But all that was about to change.

Kubernetes

Google was running containers at scale for production workloads long before anyone else. Nearly all of Google's services run in containers: Gmail, Google Search, Google Maps, Google App Engine, and so on. Because no suitable container orchestration system existed at the time, Google was compelled to invent one.

From Borg to Kubernetes

To solve the problem of running a large number of services at global scale on millions of servers, Google developed a private, internal container orchestration system it called Borg (*https://pdos.csail.mit.edu/6.824/papers/borg.pdf*).

Borg is essentially a centralized management system that allocates and schedules containers to run on a pool of servers. While very powerful, Borg is tightly coupled to Google's own internal and proprietary technologies, difficult to extend, and impossible to release to the public.

In 2014, Google founded an open source project named Kubernetes (from the Greek word κυβερνήτης, meaning "helmsman, pilot") that would develop a container orchestrator that everyone could use, based on the lessons learned from Borg and its successor, Omega (*https://ai.google/research/pubs/pub41684.pdf*).

Kubernetes's rise was meteoric. While other container orchestration systems existed before Kubernetes, they were commercial products tied to a vendor, and that was always a barrier to their widespread adoption. With the advent of a truly free and open source container orchestrator, adoption of both containers and Kubernetes grew at a phenomenal rate.

By late 2017, the orchestration wars were over, and Kubernetes had won. While other systems are still in use, from now on companies looking to move their infrastructure to containers only need to target one platform: Kubernetes.

What Makes Kubernetes So Valuable?

Kelsey Hightower, a staff developer advocate at Google, coauthor of *Kubernetes Up & Running* (O'Reilly), and all-around legend in the Kubernetes community, puts it this way:

> Kubernetes does the things that the very best system administrator would do: automation, failover, centralized logging, monitoring. It takes what we've learned in the DevOps community and makes it the default, out of the box.
>
> —Kelsey Hightower

Many of the traditional sysadmin tasks like upgrading servers, installing security patches, configuring networks, and running backups are less of a concern in the cloud native world. Kubernetes can automate these things for you so that your team can concentrate on doing its core work.

Some of these features, like load balancing and autoscaling, are built into the Kubernetes core; others are provided by add-ons, extensions, and third-party tools that use the Kubernetes API. The Kubernetes ecosystem is large, and growing all the time.

Kubernetes makes deployment easy

Ops staff love Kubernetes for these reasons, but there are also some significant advantages for developers. Kubernetes greatly reduces the time and effort it takes to deploy. Zero-downtime deployments are common, because Kubernetes does rolling updates by default (starting containers with the new version, waiting until they become healthy, and then shutting down the old ones).

Kubernetes also provides facilities to help you implement continuous deployment practices such as *canary deployments*: gradually rolling out updates one server at a time to catch problems early (see "Canary Deployments" on page 244). Another common practice is *blue-green* deployments: spinning up a new version of the system in parallel, and switching traffic over to it once it's fully up and running (see "Blue/Green Deployments" on page 243).

Demand spikes will no longer take down your service, because Kubernetes supports autoscaling. For example, if CPU utilization by a container reaches a certain level, Kubernetes can keep adding new replicas of the container until the utilization falls below the threshold. When demand falls, Kubernetes will scale down the replicas again, freeing up cluster capacity to run other workloads.

Because Kubernetes has redundancy and failover built in, your application will be more reliable and resilient. Some managed services can even scale the Kubernetes cluster itself up and down in response to demand, so that you're never paying for a larger cluster than you need at any given moment (see "Autoscaling" on page 102).

The business will love Kubernetes too, because it cuts infrastructure costs and makes much better use of a given set of resources. Traditional servers, even cloud servers, are mostly idle most of the time. The excess capacity that you need to handle demand spikes is essentially wasted under normal conditions.

Kubernetes takes that wasted capacity and uses it to run workloads, so you can achieve much higher utilization of your machines—and you get scaling, load balancing, and failover for free too.

While some of these features, such as autoscaling, were available before Kubernetes, they were always tied to a particular cloud provider or service. Kubernetes is *provider-agnostic*: once you've defined the resources you use, you can run them on any Kubernetes cluster, regardless of the underlying cloud provider.

That doesn't mean that Kubernetes limits you to the lowest common denominator. Kubernetes maps your resources to the appropriate vendor-specific features: for example, a load-balanced Kubernetes service on Google Cloud will create a Google Cloud load balancer, on Amazon it will create an AWS load balancer. Kubernetes abstracts away the cloud-specific details, letting you focus on defining the behavior of your application.

Just as containers are a portable way of defining software, Kubernetes resources provide a portable definition of how that software should run.

Will Kubernetes Disappear?

Oddly enough, despite the current excitement around Kubernetes, we may not be talking much about it in years to come. Many things that once were new and revolutionary are now so much part of the fabric of computing that we don't really think about them: microprocessors, the mouse, the internet.

Kubernetes, too, is likely to disappear and become part of the plumbing. It's boring, in a good way: once you learn what you need to know to deploy your application to Kubernetes, you're more or less done.

The future of Kubernetes is likely to lie largely in the realm of managed services. Virtualization, which was once an exciting new technology, has now simply become a utility. Most people rent virtual machines from a cloud provider rather than run their own virtualization platform, such as vSphere or Hyper-V.

In the same way, we think Kubernetes will become so much a standard part of the plumbing that you just won't know it's there anymore.

Kubernetes Doesn't Do It All

Will the infrastructure of the future be entirely Kubernetes-based? Probably not. Firstly, some things just aren't a good fit for Kubernetes (databases, for example):

> Orchestrating software in containers involves spinning up new interchangeable instances without requiring coordination between them. But database replicas are not interchangeable; they each have a unique state, and deploying a database replica requires

> coordination with other nodes to ensure things like schema changes happen everywhere at the same time:
>
> —Sean Loiselle (*https://www.cockroachlabs.com/blog/kubernetes-state-of-stateful-apps*) (Cockroach Labs)

While it's perfectly possible to run stateful workloads like databases in Kubernetes with enterprise-grade reliability, it requires a large investment of time and engineering that it may not make sense for your company to make (see "Run Less Software" on page 47). It's usually more cost-effective to use managed services instead.

Secondly, some things don't actually need Kubernetes, and can run on what are sometimes called *serverless* platforms, better named *functions as a service*, or *FaaS* platforms.

Cloud functions and funtainers

AWS Lambda, for example, is a FaaS platform that allows you to run code written in Go, Python, Java, Node.js, C#, and other languages, without you having to compile or deploy your application at all. Amazon does all that for you.

Because you're billed for the execution time in increments of 100 milliseconds, the FaaS model is perfect for computations that only run when you need them to, instead of paying for a cloud server, which runs all the time whether you're using it or not.

These *cloud functions* are more convenient than containers in some ways (though some FaaS platforms can run containers as well). But they are best suited to short, standalone jobs (AWS Lambda limits functions to fifteen minutes of run time, for example, and around 50 MiB of deployed files), especially those that integrate with existing cloud computation services, such as Microsoft Cognitive Services or the Google Cloud Vision API.

Why don't we like to refer to this model as *serverless*? Well, it isn't: it's just somebody else's server. The point is that you don't have to provision and maintain that server; the cloud provider takes care of it for you.

Not every workload is suitable for running on FaaS platforms, by any means, but it is still likely to be a key technology for cloud native applications in the future.

Nor are cloud functions restricted to public FaaS platforms such as Lambda or Azure Functions: if you already have a Kubernetes cluster and want to run FaaS applications on it, OpenFaaS (*https://www.openfaas.com/*) and other open source projects make this possible. This hybrid of functions and containers is sometimes called *funtainers*, a name we find appealing.

A more sophisticated software delivery platform for Kubernetes that encompasses both containers and cloud functions, called Knative, is currently under active development (see "Knative" on page 240). This is a very promising project, which may mean that in the future the distinction between containers and functions may blur or disappear altogether.

Cloud Native

The term *cloud native* has become an increasingly popular shorthand way of talking about modern applications and services that take advantage of the cloud, containers, and orchestration, often based on open source software.

Indeed, the Cloud Native Computing Foundation (CNCF) (*https://www.cncf.io/*) was founded in 2015 to, in their words, "foster a community around a constellation of high-quality projects that orchestrate containers as part of a microservices architecture."

Part of the Linux Foundation, the CNCF exists to bring together developers, end-users, and vendors, including the major public cloud providers. The best-known project under the CNCF umbrella is Kubernetes itself, but the foundation also incubates and promotes other key components of the cloud native ecosystem: Prometheus, Envoy, Helm, Fluentd, gRPC, and many more.

So what exactly do we mean by *cloud native*? Like most such things, it means different things to different people, but perhaps there is some common ground.

Cloud native applications run in the cloud; that's not controversial. But just taking an existing application and running it on a cloud compute instance doesn't make it cloud native. Neither is it just about running in a container, or using cloud services such as Azure's Cosmos DB or Google's Pub/Sub, although those may well be important aspects of a cloud native application.

So let's look at a few of the characteristics of cloud native systems that most people can agree on:

Automatable
: If applications are to be deployed and managed by machines, instead of humans, they need to abide by common standards, formats, and interfaces. Kubernetes provides these standard interfaces in a way that means application developers don't even need to worry about them.

Ubiquitous and flexible
: Because they are decoupled from physical resources such as disks, or any specific knowledge about the compute node they happen to be running on, containerized microservices can easily be moved from one node to another, or even one cluster to another.

Resilient and scalable
: Traditional applications tend to have single points of failure: the application stops working if its main process crashes, or if the underlying machine has a hardware failure, or if a network resource becomes congested. Cloud native applications, because they are inherently distributed, can be made highly available through redundancy and graceful degradation.

Dynamic
: A container orchestrator such as Kubernetes can schedule containers to take maximum advantage of available resources. It can run many copies of them to achieve high availability, and perform rolling updates to smoothly upgrade services without ever dropping traffic.

Observable
: Cloud native apps, by their nature, are harder to inspect and debug. So a key requirement of distributed systems is *observability*: monitoring, logging, tracing, and metrics all help engineers understand what their systems are doing (and what they're doing wrong).

Distributed
: Cloud native is an approach to building and running applications that takes advantage of the distributed and decentralized nature of the cloud. It's about how your application works, not where it runs. Instead of deploying your code as a single entity (known as a *monolith*), cloud native applications tend to be composed of multiple, cooperating, distributed *microservices*. A microservice is simply a self-contained service that does one thing. If you put enough microservices together, you get an application.

It's not just about microservices

However, microservices are not a panacea. Monoliths are easier to understand, because everything is in one place, and you can trace the interactions of different parts. But it's hard to scale monoliths, both in terms of the code itself, and the teams of developers who maintain it. As the code grows, the interactions between its various parts grow exponentially, and the system as a whole grows beyond the capacity of a single brain to understand it all.

A well-designed cloud native application is composed of microservices, but deciding what those microservices should be, where the boundaries are, and how the different services should interact is no easy problem. Good cloud native service design consists of making wise choices about how to separate the different parts of your architecture. However, even a well-designed cloud native application is still a distributed system, which makes it inherently complex, difficult to observe and reason about, and prone to failure in surprising ways.

While cloud native systems tend to be distributed, it's still possible to run monolithic applications in the cloud, using containers, and gain considerable business value from doing so. This may be a step on the road to gradually migrating parts of the monolith outward to modern microservices, or a stopgap measure pending the redesign of the system to be fully cloud native.

The Future of Operations

Operations, infrastructure engineering, and system administration are highly skilled jobs. Are they at risk in a cloud native future? We think not.

Instead, these skills will only become more important. Designing and reasoning about distributed systems is hard. Networks and container orchestrators are complicated. Every team developing cloud native applications will need operations skills and knowledge. Automation frees up staff from boring, repetitive manual work to deal with more complex, interesting, and fun problems that computers can't yet solve for themselves.

That doesn't mean all current operations jobs are guaranteed. Sysadmins used to be able to get by without coding skills, except maybe cooking up the odd simple shell script. In the cloud, that won't fly.

In a software-defined world, the ability to write, understand, and maintain software becomes critical. If you can't or won't learn new skills, the world will leave you behind —and it's always been that way.

Distributed DevOps

Rather than being concentrated in a single operations team that services other teams, ops expertise will become distributed among many teams.

Each development team will need at least one ops specialist, responsible for the health of the systems or services the team provides. She will be a developer, too, but she will also be the domain expert on networking, Kubernetes, performance, resilience, and the tools and systems that enable the other developers to deliver their code to the cloud.

Thanks to the DevOps revolution, there will no longer be room in most organizations for devs who can't ops, or ops who don't dev. The distinction between those two disciplines is obsolete, and is rapidly being erased altogether. Developing and operating software are merely two aspects of the same thing.

Some Things Will Remain Centralized

Are there limits to DevOps? Or will the traditional central IT and operations team disappear altogether, dissolving into a group of roving internal consultants, coaching, teaching, and troubleshooting ops issues?

We think not, or at least not entirely. Some things still benefit from being centralized. It doesn't make sense for each application or service team to have its own way of detecting and communicating about production incidents, for example, or its own ticketing system, or deployment tools. There's no point in everybody reinventing their own wheel.

Developer Productivity Engineering

The point is that self-service has its limits, and the aim of DevOps is to speed up development teams, not slow them down with unnecessary and redundant work.

Yes, a large part of traditional operations can and should be devolved to other teams, primarily those that involve code deployment and responding to code-related incidents. But to enable that to happen, there needs to be a strong central team building and supporting the DevOps ecosystem in which all the other teams operate.

Instead of calling this team *operations*, we like the name *developer productivity engineering* (DPE). DPE teams do whatever's necessary to help developers do their work better and faster: operating infrastructure, building tools, busting problems.

And while developer productivity engineering remains a specialist skill set, the engineers themselves may move outward into the organization to bring that expertise where it's needed.

Lyft engineer Matt Klein has suggested that, while a pure DevOps model makes sense for startups and small firms, as an organization grows, there is a natural tendency for infrastructure and reliability experts to gravitate toward a central team. But he says that team can't be scaled indefinitely:

> By the time an engineering organization reaches ~75 people, there is almost certainly a central infrastructure team in place starting to build common substrate features required by product teams building microservices. But there comes a point at which the central infrastructure team can no longer both continue to build and operate the infrastructure critical to business success, while also maintaining the support burden of helping product teams with operational tasks.
>
> —Matt Klein (*https://medium.com/@mattklein123/the-human-scalability-of-devops-e36c37d3db6a*)

At this point, not every developer can be an infrastructure expert, just as a single team of infrastructure experts can't service an ever-growing number of developers. For larger organizations, while a central infrastructure team is still needed, there's also a case for embedding *site reliability engineers* (SREs) into each development or product team. They bring their expertise to each team as consultants, and also form a bridge between product development and infrastructure operations.

You Are the Future

If you're reading this book, it means you're going to be part of this new cloud native future. In the remaining chapters, we'll cover all the knowledge and skills you'll need as a developer or operations engineer working with cloud infrastructure, containers, and Kubernetes.

Some of these things will be familiar, and some will be new, but we hope that when you've finished the book you'll feel more confident in your own ability to acquire and master cloud native skills. Yes, there's a lot to learn, but it's nothing you can't handle. You've got this!

Now read on.

Summary

We've necessarily given you a rather quick tour of the cloud native DevOps landscape, but we hope it's enough to bring you up to speed with some of the problems that cloud, containers, and Kubernetes solve, and how they're likely to change the IT business. If you're already familiar with this, then we appreciate your patience.

A quick recap of the main points before we move on to meet Kubernetes in person in the next chapter:

- Cloud computing frees you from the expense and overhead of managing your own hardware, making it possible for you to build resilient, flexible, scalable distributed systems.
- DevOps is a recognition that modern software development doesn't stop at shipping code: it's about closing the feedback loop between those who write the code and those who use it.
- DevOps also brings a code-centric approach and good software engineering practices to the world of infrastructure and operations.
- Containers allow you to deploy and run software in small, standardized, self-contained units. This makes it easier and cheaper to build large, diverse, distributed systems, by connecting together containerized microservices.
- Orchestration systems take care of deploying your containers, scheduling, scaling, networking, and all the things that a good system administrator would do, but in an automated, programmable way.

- Kubernetes is the de facto standard container orchestration system, and it’s ready for you to use in production right now, today.
- *Cloud native* is a useful shorthand for talking about cloud-based, containerized, distributed systems, made up of cooperating microservices, dynamically managed by automated infrastructure as code.
- Operations and infrastructure skills, far from being made obsolete by the cloud native revolution, are and will become more important than ever.
- It still makes sense for a central team to build and maintain the platforms and tools that make DevOps possible for all the other teams.
- What will go away is the sharp distinction between software engineers and operations engineers. It’s all just software now, and we’re all engineers.

O'REILLY®

Cloud Native DevOps with Kubernetes

Kubernetes is the operating system of the cloud native world, providing a reliable and scalable platform for running containerized workloads. In this friendly, pragmatic book, cloud experts John Arundel and Justin Domingus show you what Kubernetes can do—and what you can do with it.

You'll learn all about the Kubernetes ecosystem, and use battle-tested solutions to everyday problems. You'll build, step by step, an example cloud native application and its supporting infrastructure, along with a development environment and continuous deployment pipeline that you can use for your own applications.

- Understand containers and Kubernetes from first principles; no experience necessary
- Run your own clusters or choose a managed Kubernetes service from Amazon, Google, and others
- Use Kubernetes to manage resource usage and the container lifecycle
- Optimize clusters for cost, performance, resilience, capacity, and scalability
- Learn the best tools for developing, testing, and deploying your applications
- Apply the latest industry practices for security, observability, and monitoring
- Adopt DevOps principles to help make your development teams lean, fast, and effective

"The most encompassing, definitive, and practical text about the care and feeding of Kubernetes infrastructure. An absolute must-have."

—**Jeremy Yates**
SRE Team, The Home Depot QuoteCenter

"A super clear and informative read, covering all the details without compromising readability."

—**Will Thames**
Platform Engineer, Skedulo

John Arundel is a consultant and author with over 30 years of experience in the computer industry. He works with companies all over the world, consulting on Kubernetes, cloud, and infrastructure.

Justin Domingus is a DevOps engineer at CareZone.com who specializes in Kubernetes and cloud computing.

CLOUD / SYS ADMIN

MRP: ₹ 1,150 .00

Twitter: @oreillymedia
facebook.com/oreilly

SPD®
SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.

ISBN : 978-93-5213-809-8

9 789352 138098

First Edition/2019/Paperback/English