Brought to you by:

VMware Tanzu™

DevSecOps

for dummies®

A Wiley Brand

Add security to DevOps practices

Use containers safely and effectively

Manage Kubernetes securely across clouds

Brad Bock

VMware Tanzu Special Edition

About VMware Tanzu

To win in the market today, you need to invest in both your people and your technology. Free up your teams to take on more complex challenges. Adopt agile, cloud native development methods and tools. Upgrade your legacy systems to new cloud native runtimes.

This kind of transformation at enterprise scale isn't easy. But with VMware Tanzu, you can accelerate the pace of change. We'll help rethink the way your business operates and turn it into a next generation enterprise.

DevSecOps

for dummies®
A Wiley Brand

DevSecOps

VMware Tanzu Special Edition

by Brad Bock

for dummies®
A Wiley Brand

DevSecOps For Dummies®, VMware Tanzu Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

ISBN 978-1-119-83256-0 (pbk); ISBN 978-1-119-83257-7 (ebk)

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Elizabeth Kuball

Acquisitions Editor: Ashley Coffey

Editorial Manager: Rev Mengle

Business Development Representative: Cynthia Tweed

Production Editor: Tamilmani Varadharaj

Special Help: Jennifer Krazit, Keith Lee, Faithe Wempen

Table of Contents

Introduction

We talk a lot about DevSecOps inside VMware Tanzu, and we're not alone. In fact, in Gartner's 2020 Hype Cycle for Application Security, DevSecOps is the only technology or process deemed "transformational," and Gartner analysts believe it will reach mainstream adoption in just two to five years. But all this attention raises two big questions: Just what on Earth does DevSecOps mean, and why is it critical that you get on board with it now?

As the name implies, DevSecOps is the inclusion of security within DevOps practices. But it's more nuanced than just inserting "security" at some predefined step in the application life cycle. Instead, it's about making security an inextricable, if not intrinsic, part of the application life cycle. An organization embracing DevSecOps is an organization that's moving fast and *not* breaking things.

And why is now the time to incorporate DevSecOps into your software development process? Because modern, cloud-native applications are complex, and their life cycles are fast paced. Traditional security practices — especially those involving human intervention — often won't cut it in a modern IT environment. In order to keep up with the pace of change, security must be automated and built into the software development life cycle. Properly materialized, teams and organizations embracing DevSecOps are teams and organizations building great software and mitigating risk simultaneously.

The best part: DevSecOps will almost certainly save you time and money. DevOps practices should be standard operating procedures for modern applications, which are all about delivering better digital experiences faster by improving development speed, automating operations, and using open-source components. Dev*Sec*Ops is all about doing these things securely, because security incidents typically cost many, many times more than the relatively simple steps that could've been taken to avoid them.

About This Book

This book explains DevSecOps in the context of modern, containerized applications — most likely leveraging a microservices architecture and running on top of Kubernetes, a very popular container orchestration platform. It provides the foundational knowledge to get up to speed on core concepts (including DevOps), and then shares the necessary requirements for safely running applications in containers and managing them on Kubernetes clusters.

It's also worth noting that DevSecOps is not a replacement for other IT security practices, nor should it override efforts to automate other layers of the IT stack. Instead, this book focuses on DevSecOps at the application runtime level (Kubernetes) and on the building of applications themselves. Ideally, the two primary personas discussed in this book — software developers and platform operators — will work in conjunction with their peers across operations, security, and other teams as part of an organization-wide effort to improve cybersecurity, while simultaneously increasing the pace and quality of software development.

Foolish Assumptions

In writing this book, we made some assumptions about who will be reading it — including that your team or organization has already adopted DevOps practices, or at least is aware of the benefits they provide and the reasons for their popularity.

We're also assuming that you fit one of these general profiles:

- You're a platform architect tasked with building and running an application platform for your development team. You either have settled on Kubernetes as the foundation of that platform or are very seriously considering it, and you know you need to improve the developer experience and security for applications running on it.
- You're an application owner or line-of-business manager in the process of modernizing your software development and application architecture. You want to ensure your team makes the right decisions to maximize velocity, performance, and security.

- You're an executive trying to get up to speed on the technologies and processes underpinning modern application development. As IT evolves from "keeping the lights on" to actually driving revenue, you know you'll need to make informed decisions to ensure the organization is able to meet new business requirements while keeping sensitive data secure.

Icons Used in This Book

This book uses icons in the margin to draw your attention to certain kinds of information. Here's a guide to the icons:

TIP

We use the Tip icon to highlight anything that'll save you time or money or just make your life a little easier.

REMEMBER

When we tell you something so important that you should commit it to memory, we mark it with the Remember icon.

WARNING

When we want you to avoid making a potentially costly mistake, we mark that material with the Warning icon.

TECHNICAL STUFF

Sometimes we get into the weeds, providing some information that's a bit more technical in nature. When we do, we mark it with the Technical Stuff icon.

Beyond the Book

After you read this book, you'll probably have some questions! Chapter 4 contains a list of resources to help educate you on some core topics related to DevSecOps, as well as to help you get started on your journey. For a thorough collection of blog posts, white papers, analyst reports, and more content explaining the concepts discussed in this book in more detail, visit `https://tanzu.vmware.com/content/devsecops-101`.

IN THIS CHAPTER

- **Getting a basic grip on DevOps**
- **Understanding important key terms and concepts**
- **Revisiting the DevOps definition**

Chapter 1 Understanding DevSecOps

The first and most important thing to understand about DevSecOps is that there's no magic bullet that will get you to an ideal state overnight. DevSecOps is a collection of technologies, techniques, and even mindsets that work together to increase velocity, while also enhancing security, at every layer of the software development process. Depending on how far an organization is down the path of application modernization, doing DevSecOps right could mean some fundamental changes to how your organization approaches both software development and IT operations.

What Is DevOps?

When preparing to implement DevSecOps, the first step is committing to the principles of DevOps — for which best practices can be found in countless books, conference talks, and websites, including `https://tanzu.vmware.com/devops`. In fact, the goal of DevSecOps is that security becomes such an intrinsic part of the application life cycle in a DevOps-centric environment that, as a nerdy joke goes, the term is still pronounced *DevOps* because the *Sec* is silent.

We share some of our practical assumptions about DevOps later in this chapter, but a simple definition is that DevOps is what happens when organizations stop treating software development and operations as separate entities and start treating them like complementary functions, working toward the shared goal of better applications. Figure 1-1 shows the DevOps loop, a common method of illustrating how these two previously distinct functions have merged.

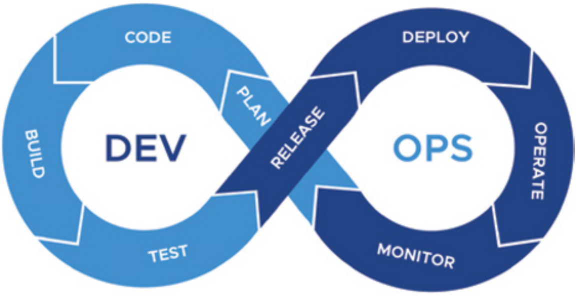


FIGURE 1-1: The DevOps loop highlights the interconnected and continuous nature of DevOps.

Why do we need better applications? Because in 2021 and into the foreseeable future, digital experiences will make or break organizations. It's applications — built in-house and designed uniquely for specific tasks — that will draw in more customers, optimize back-end processes, and otherwise help organizations compete. The COVID-19 pandemic highlighted this reality; companies that were prepared to step up their digital experiences were able to weather the storm and even thrive — while also forever digitizing some previously physical experiences.

Why do we need all that *plus* better security? Because if there's a vulnerability in your software, someone is likely to discover it and possibly exploit it. Our increasing connectivity is breaking down the traditional network barriers, meaning criminals don't need to breach the corporate network anymore. All they need in order to get a toehold is an out-of-date component buried somewhere in your software stack. And increasingly complex and dynamic application architectures mean it's more difficult than ever to keep track of everything that's running inside your network.

Implementing the security techniques described in this book is a largely preventive exercise, like taking vitamins or brushing your teeth after every meal. You can't always tell if it's working in the moment, because the whole point is that you're getting ahead of potential issues. Before we go deeper, though, let's walk through some of the fundamental concepts and terminology we'll be using throughout the rest of this book.

Learning the Lingo

Probably the most fundamental element of modern cloud-native software development (and DevSecOps by proxy) is the container. Whether you're building applications with microservices architectures or just lifting and shifting existing applications onto a new platform, the chances are good that they'll be packaged as containers.

A *container* is a self-contained unit of software that includes all the elements necessary to run an application consistently from one computing environment to the next. If you could peer inside an application container, you would find things like application code or binaries, language runtimes, dependencies, and settings — everything needed for the application to run.

Developers build containers using configuration files that instruct a container build system to gather these necessary parts and combine them with an operating system (OS), similar to the way users install an application on a desktop computer. Containers are then run using what's called a *container runtime.* A container runtime is what enables containers to be portable and run consistently in any computing environment. A container runtime is a mediator between containers and the underlying compute infrastructure.

If containers sound similar to virtual machines (VMs), that's because they are. Figure 1-2 summarizes these differences. The major difference is in how the resources are shared by each unit. VMs are complete guest OS units that share physical host server resources such as processing power, memory, and disk space. VMs run on top of a virtualization layer called a *hypervisor.* Containers are workload units that include dependencies and libraries but share an OS kernel. As a result, containers are more efficient than VMs because they don't each require a complete guest OS. They can start up quickly because they access already-running shared OS resources instead of having to wait for those resources to initialize. But the two formats are not necessarily at odds with one another: It's quite common for containers to run *on* VMs to take advantage of the dynamic, software-defined nature of both.

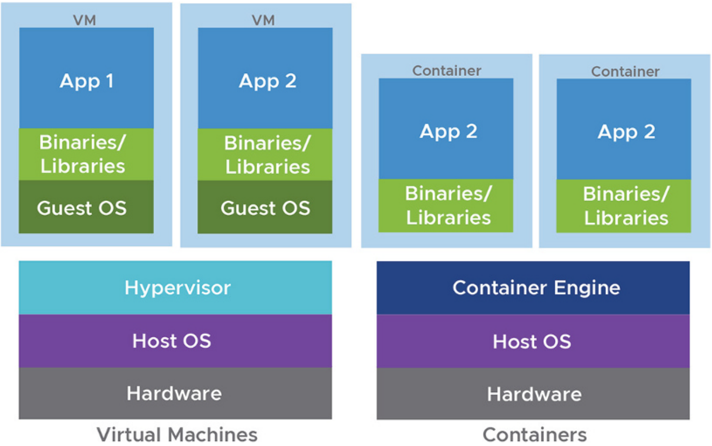


FIGURE 1-2: Note that by running a container engine inside a VM (on the left side of the diagram), the guest OS could support multiple containers.

TECHNICAL STUFF

Technically, containers are only called "containers" when they're running and consuming resources. Otherwise, the file that details what they are and what they need to run is called a *container image*. Container images are often accessed, or pulled, from a library called a container registry.

Containers are popular for many reasons, but a major one is that developers can build and test them on their laptops and then deploy them on any system running a compatible container runtime. This capability cuts down the delta between development and production environments and provides a standard method of shipping software. Because containers share the OS with the compute environment on which they run, they have smaller footprints, meaning many containers can share a single machine.

An important related technology is the *container registry,* which is basically a library that stores container images. The most popular public container registry is Docker Hub, from which you can search among tens of thousands of container images ranging from obscure tools to popular databases. In addition, many large enterprises and other organizations choose to run private container registries, which include images that have been built and vetted to meet their specific needs. When someone downloads a container image to run in their environment, it's called *pulling the image.*

TECHNICAL STUFF

Although Docker used to be synonymous with *containers* and *container runtimes*, that isn't always the case anymore (for reasons too complicated to explain in this book). Today, organizations have a variety of systems to choose from, almost all of which conform to a common container image specification and container runtime specification managed by the Open Container Initiative (OCI). This means that they're interoperable: Any OCI-compatible container image can be run on any OCI-compatible container runtime. However, Docker supports OCI container and runtime specifications and is still the most widely used container-build system.

Although containers only need a runtime in order to run, many organizations prefer to bolster them with a *container orchestrator.* Of those, Kubernetes is far and away the most popular choice. A container orchestrator handles the operational aspects of running applications in containers — things like service discovery, restarting containers when they die, allocating system resources, and managing groups of containers that must be tightly coupled, to name just a few.

Here are a few Kubernetes-specific terms worth understanding:

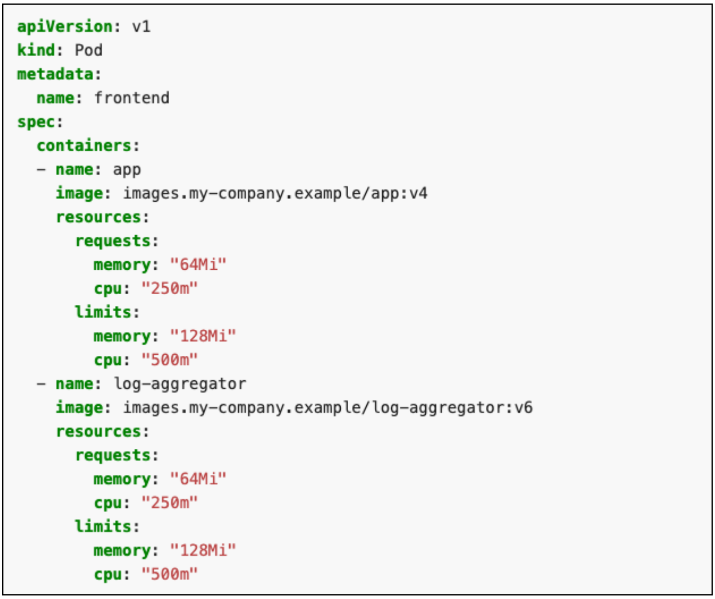
- **Cluster:** The set of machines (virtual or physical) on which Kubernetes is installed.
- **Node:** An individual machine in a Kubernetes cluster. A cluster *can* be a single node, but clusters typically run across multiple nodes for added resilience.
- **Pod:** A container or group of containers that are managed as a single entity and share storage and network resources.
- **Service:** An application that's exposed to the network. Service discovery is the method by which services can locate each other so they can send traffic or communicate.

OTHER DEVSECOPS TERMS TO KNOW

Here are a few terms that won't come up too much in this book but that are still critical to understand:

- **Application programming interfaces (APIs):** The method (and language) through which applications communicate with each other. If it weren't for APIs, all software would run in a vacuum and would, therefore, be useless. Imagine, for example, having an OS that couldn't run applications.
- **Continuous integration/continuous delivery (CI/CD):** The process of regularly writing new code and integrating it into the codebase rather than shipping new updates quarterly, yearly, or on some other prescribed timeline. CI typically involves a series of automated tests to ensure new features or bug fixes work well and are otherwise compliant before they're integrated into the codebase. Following CI, CD updates running applications with those new features or bug fixes. Together, the process that begins with a change to application code and ends with changes to production applications is called a *CI/CD pipeline.*
- **Authentication/authorization:** Authentication (sometimes referred to as AuthN or authn) is the act of verifying whether users really are who they say they are or that processes (likely running as microservices for the purpose of this book) really are what they claim to be. Authorization (sometimes referred to as AuthZ or authz) is the act of determining whether an authenticated entity is allowed to access the resource in question. These problems can be difficult in the context of DevSecOps because of design limitations of microservices and because networks composed of containerized microservices can be complex and dynamic.

- **YAML:** The language through which humans define what's running on their Kubernetes clusters and container images. (*YAML* is a recursive acronym, short for *YAML Ain't Markup Language.*) In Figure 1-3, the YAML file indicates a pod consisting of two containers, as well as their memory and CPU requirements.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

FIGURE 1-3: A YAML file declaring a pod consisting of two containers (app and log-aggregator), their locations, and minimum and maximum requirements for memory and CPU resources.

Microservices are a big reason why container orchestration is so popular. As opposed to traditional monolithic applications, where a single codebase includes all the business logic, microservices are loosely coupled and typically only serve a single function. So, for example, a large monolith encompassing 400 different functions might be broken down into 400 smaller microservices, each running in its own container. We cover some other modern technologies for managing microservices in Chapters 2 and 3, but suffice it to say that using a platform like Kubernetes is a critical first step toward bringing order and automation to these complex environments.

REMEMBER

Before container orchestration platforms like Kubernetes were widely available, managing microservices could be a very daunting task. Legacy techniques for managing them included using spreadsheets to track IP addresses and manually mapping which services connected to which other services.

A final term worth defining in the context of DevSecOps is *vulnerability* or, even more broadly, *security.* We've all read stories about scary exploits at every layer of the software stack, but in this book we're primarily concerned with vulnerabilities within the application code and containerized components. We're also concerned with securing the communications between containers or microservices within the network, and between those same services and the outside world.

The goal of DevSecOps is to automate as much of the container- and microservice-security life cycle as possible, so that organizations can meet their DevOps objectives without slowing down. This applies while containers are being built and deployed, as well as when managing them and their components while they're running in production.

TECHNICAL STUFF

You've probably come across the acronym *CVE* when reading about security incidents or patches. The term comes from the Common Vulnerabilities and Exposures database (`https://cve.mitre.org`), which is funded by the U.S. government and tracks known vulnerabilities in software products and open-source projects.

Although the terms *vulnerabilities* and *exposures* technically have different meanings, the distinctions are less relevant today, and *vulnerability* is generally used as an umbrella term that covers everything. Every vulnerability is assigned a label that includes the year it was added and a unique identifying number (for example, CVE-2019-5736), as well as a severity score (10 is the highest) and details on the vulnerability itself and which software versions or configurations are at risk.

Defining a New Era for DevOps

If you look at the definition of DevOps shared at the beginning of this chapter, you may notice that it has a lot of wiggle room. This is why the term means so many different things to so many different people. (If you don't believe us, start asking your engineering team what it means!) There's probably some programmable infrastructure, some CI/CD, and, yes, some agile development involved, but even those terms can be sliced a thousand different ways given the wide array of available technologies.

So, perhaps it’s easier to begin with what DevOps is not — or, at least, is not *any longer.* DevOps is not software development and operations that exist as independent, siloed entities. That old way of doing things inevitably led to the classic situation of development and operations teams perpetually at odds with each other, viewing the other team as an obstacle to overcome.

We probably also can agree that DevOps is not confined to the popular “you build it, you run it” approach. That mentality can work for small teams and those without much legacy IT to contend with, but it can fall apart very quickly within large enterprises, where a more-centralized *platform as a product* or *platform operations* approach may work better.

It’s also easier to talk about DevOps by locking into a specific foundational technology — in this case, Kubernetes — because although the primary goal of speeding up the application life cycle may remain the same, choices around platform and the application architecture will affect both development and operational workflows. In exchange for the relative freedom of self-service resources, for example, developers must deal with complicated configuration templates. In exchange for the relative freedom of microservices and continuous deployment, they must account for service discovery, API gateways, service meshes, and CI/CD pipelines. (CI/CD pipelines should be a core component of any DevOps environment, and we explain service meshes in Chapter 2.)

INSERTING “SECURITY” INTO DEVOPS

Security teams became yet another perceived obstacle as more workloads moved online and data breaches became commonplace. Better security, of course, is very much a good thing, but stricter protocols came at the expense of fast development cycles — which are one of the driving factors behind DevOps and a big reason behind the steady improvement of web applications. By bringing automation to DevOps security, DevSecOps removes the friction between DevOps and security teams, thus enabling faster and safer development cycles.

Operators also must learn some new tricks in a Kubernetes environment. In exchange for getting out of the business of being infrastructure gatekeepers, they need to monitor an ever-expanding network of containers, microservices, cloud instances, and VMs. If they don't want to manually configure and approve application components, they need to give developers easy access to the tools they need. In exchange for automating these processes, they need to learn Kubernetes.

When it's done well, the results are worth any additional effort. DevOps teams running on all cylinders are shipping better code, faster. Downtime is reduced and issues are resolved more quickly. And, in the case of Kubernetes, everyone speaks the same language. You can't overstate the importance of employees in different roles and even different teams all working from the same baseline of knowledge about how their systems work.

IN THIS CHAPTER

- Discovering the keys to creating secure, optimized code
- Learning about the key technologies for container security
- Automating container security and life cycle using DevSecOps techniques

Chapter 2

Fusing DevOps and Container Security

In Chapter 1, we explain the technologies that underpin modern software development. In this chapter, we show you how to build, deploy, and manage containers using a DevSecOps mindset. This means the containers will be secure at every step of the life cycle and that security will be an integrated, if not automated, part of the process. If you've heard people talk about "shifting left" with regard to DevSecOps, this is it: Developers are working with operations and security teams to integrate security earlier in the application life cycle, thus taking on more responsibility for application security as part of their workflow.

Figure 2-1 illustrates a typical application life cycle, in which security review generally happens between the Release and Deploy phases. An application life cycle following DevSecOps principles includes security measures at every step. The more automated, the better so they're not ignored and they don't slow down the path to production.

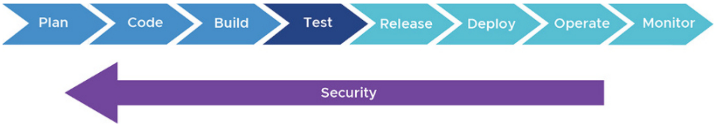


FIGURE 2-1: These are the same phases as in the DevOps loop. Although DevOps aims to make this a continuous cycle, DevSecOps aims to embed security throughout that cycle.

Building Secure, Optimized Code

Before you rush to start building containers and installing Kubernetes clusters, you'll want to address the most important part of any application: its code. You can have the world's most secure container environment, but bad decisions in the application itself can still provide opportunities for bad actors (and, of course, poor performance and user experience). The good news is that there are multiple tried-and-true methods for ensuring that developers' code is both secure and optimized for modern computing.

Application frameworks are an effective approach for developing secure and high-performing applications. The beauty of using application frameworks is that they often automate — or at least greatly simplify — the process of incorporating best practices around security (such as authentication and encryption), application architecture (such as "circuit breaker" patterns and serverless computing), and other capabilities (like testing and tracing) that should be part of any modern application.

TIP

We strongly recommend the open-source Spring framework (`https://spring.io`) for Java applications and the open-source Steeltoe framework (`https://steeltoe.io`) for .NET applications. VMware manages both projects, which are designed for cloud-native use cases. Millions of developers around the world use them — from large traditional enterprises to companies such as Netflix that were born in the cloud. These frameworks help developers focus on business logic and incorporate advanced functionality — sometimes with only a single line of code.

The tooling ecosystem that has built up around Spring handles most of this legwork for developers, including by providing a cloud-native application programming interface (API) gateway (`https://spring.io/projects/spring-cloud-gateway`) to help manage interactions between microservices and the outside world. An API gateway needs strong access control and authentication, but it shouldn't burden valid users or machines making requests — so things like enterprise identity management and single sign-on (SSO) become important. The API gateway must also be able to adequately handle rate limiting across microservices to help mitigate distributed denial-of-service (DDoS) or brute-force attacks, and other methods for breaching or disrupting systems, with automated and repetitive actions.

Code-scanning tools are another useful piece of technology when it comes to shipping secure applications. They come in various flavors but share a common capability of alerting developers to vulnerabilities in their code, such as misconfigured user permissions or misuse of keys and credentials that could allow hackers to access sensitive data. Some scanning tools will also flag vulnerabilities in open-source components and container images, a process we discuss in the "Container scanning" section, later in this chapter.

REMEMBER

A major goal of DevSecOps — and something the VMware Tanzu Labs team (`https://tanzu.vmware.com/labs`) preaches — is to make the right thing the easy thing. That applies to writing applications, managing production infrastructure, and everything in between. Consistency and automation across tools and processes go a long way toward delivering on this goal.

If you're adopting this type of technology as part of a DevSecOps transformation, you'll want to ensure that scans happen in near real time to minimize their impact on developers' workflows. And whatever tools and processes your organization chooses to use, the important thing is that developers can ship the best, most secure code possible with as little overhead as possible.

After the application code is written, the process of building containers begins.

Building Secure Containers

As we explain in Chapter 1, developers build container images by creating a file that dictates what dependencies the application needs to run and where to get those things. Creating this file and using it to build a container is easy. However, doing this well for complicated applications at any scale greater than a developer workstation is easier said than done.

For starters, creating compliant and secure image files requires writing accurate instructions for how to launch the container and sourcing the operating system and all those other dependencies from approved repositories. Because both the applications themselves and the operating systems and other required dependencies evolve separately over time, applications can break or exhibit unexpected behaviors that require continual fixing. Many developers would much rather focus on writing quality application code than on learning the intricacies of building and maintaining container images.

That learning curve can create issues if it results in poorly configured container images. Issues that may arise include the following:

- **Container bloat:** Container images that include components like compilers that are necessary for building, but not running, the application end up much larger in size than necessary. This results in containers that consume more resources and take significantly more time to launch.
- **Vulnerable dependencies:** The more dependencies an application requires, and the more sources from which to pull them, the greater the chances of downloading something that's either infected already or will ultimately require a security patch.
- **Troublesome rebuilds:** Container images are built in "layers" of dependencies, with each layer technically being its own image stored in the registry. Depending on the container-build system, updating an individual layer could require just rebuilding the affected container, or it could require a time-intensive rebuild of the entire application. An additional issue with the latter approach is that the build system may pull the latest versions of other, nonaffected dependencies during the rebuild, resulting in failed builds or images that behave differently than before.

Thankfully, organizations can use one of a number of methods to mitigate the complexity of building and updating container images. In the next few sections, we introduce you to secure container registries, container scanning, buildpacks, secure application catalogs, and service meshes. And, because this is a book about DevSecOps, you can rest assured that these sections put an emphasis on automation.

Secure container registries

As we mention earlier, container images are stored in repositories called *registries,* which can be either publicly accessible or private to a specific organization. The main security benefit of a *private registry* — especially for custom applications (that is, applications developed in-house rather than by a third party) — should be obvious. Strangers from the Internet won't be able to access your container images and, as a result, the intellectual property contained in the application code.

Private registries can also make it safer to use third-party applications that have been packaged as container images. (For the purposes of this book, we're talking largely about open-source applications. Commercial vendors often maintain their own registries from which to pull their images.) Organizations can build and update their own container images for open-source applications (such as databases, web servers, and monitoring tools) to ensure they're secure. What's more, a private registry enables an organization to enforce policies that specify who can pull which images and whether developers are able to pull outdated versions.

REMEMBER

As the name DevSecOps implies, the development, security, and operations teams should work together to establish mutually acceptable policies for using private registries. Policies that are too lax can compromise security, while policies that are too strict may drive developers to find ways around them. The goal is to ensure secure containers without unduly impeding developers' workflows.

WARNING

You definitely want developers sticking to trusted images from a private registry. According to a 2020 analysis of a popular public container registry, carried out by a private cybersecurity company, more than half of the available images contained vulnerabilities and several thousand contained malware or potentially harmful capabilities. That number may be a drop in the bucket compared to the overall number of images hosted there, but it underscores the threat of pulling images from sources outside your control. If the build process for a popular container image by a trusted source were compromised to include malware, the damage could be wide-reaching.

TIP

VMware created the open-source Harbor private registry (`https://goharbor.io`), which is now managed by the Cloud Native Computing Foundation. Harbor secures artifacts with policies and role-based access control (RBAC), ensures images are scanned and free from vulnerabilities, and signs images as trusted. It’s a core component of the VMware Tanzu approach to container security.

Container scanning

In the same way that code-scanning tools analyze application code for vulnerabilities, container-scanning tools analyze container images to identify vulnerabilities. Those vulnerabilities could be in the container image’s configuration (the instructions for how to launch it) or in any of the components included as dependencies that the application requires.

Scanning containers may seem redundant if you’re already utilizing a private and secure registry, but there’s still value in doing it — especially for custom applications. A scanning process integrated into the continuous integration/continuous delivery (CI/CD) pipeline should catch misconfigured or vulnerable container images during the build process, before they’re deployed in production. Of course, the trick is to ensure those scans happen quickly and automatically, without bogging down the build process.

You can also scan running containers to discover vulnerabilities that have been disclosed since the affected image was built, or to identify nefarious behavior (such as routing network traffic to malware servers or cryptomining) that show themselves after the container is live.

If you’re not already using secure container registries and vetting third-party container images, you really must scan containers. Even in third-party applications that have already been rebuilt internally and/or vetted for security, scanning tools can catch new vulnerabilities that your team may not have gotten around to patching. As we explain in the “Buildpacks” section, later in this chapter, keeping on top of patching can facilitate even faster and more secure methods for building and deploying containers.

TIP

VMware Carbon Black Cloud Container (`www.carbonblack.com/products/vmware-carbon-black-cloud-container`) provides the necessary capabilities to integrate container scanning into your DevSecOps pipeline, including identification of vulnerabilities and misconfigurations. Policies can enforce compliance with organizational rules, as well as with industry standards such as Center for Internet Security (CIS) benchmarking.

Buildpacks

Buildpacks are an automated approach to building container images, one that drastically reduces the effort and learning curve involved with securely building containers and keeping them compliant over time. Buildpacks simplify the process of container building and maintenance by automatically identifying dependencies from source code and pulling them from a central registry. The result is that developers can build their source code into an operable container image with a single command. Several buildpack options are available, but in this book we use the generic term to refer to a specific option, Cloud Native Buildpacks, which is now a Cloud Native Computing Foundation project.

Apart from automating the container-building process, buildpacks also bolster DevSecOps environments by automating the update process when dependencies change. As an oversimplified example, let's say your organization uses CentOS as a standard operating system for containerized applications. If you're following good security practices, you'll have a base CentOS image every container shares. (It will be its own layer of every application's container image.) When a CentOS vulnerability becomes known and someone within your organization patches and rebuilds that base image, a buildpack-based system will automatically rebuild every container with the updated base OS layer. The OS will then be secure, and the container can replace the vulnerable applications running in production environments. All that happens without any developer involvement.

Figure 2-2 illustrates how the process works: A dependency update is released. A build service rebuilds containers based on that update and updates the container registry. Updated images are pushed live via a CI/CD pipeline.

As you may have guessed from the preceding paragraph, buildpacks — like all DevSecOps technologies — work best when paired with a secure container registry. If you're automatically rebuilding containers, you'll want to know where the updated images came from and that they meet your organization's security requirements. Using a secure registry in combination with buildpacks also enables you to ensure that your developers aren't pulling sketchy components, because the automated build process will only pull approved container images from the private registry.

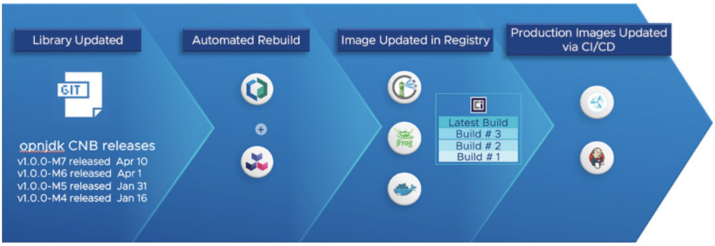

Library Updated
Automated Rebuild
Image Updated in Registry
Production Images Updated via CI/CD
GIT
opnidk CNB releases
v1.0.0-M7 released Apr 10
v1.0.0-M6 released Apr 1
v1.0.0-M5 released Jan 31
v1.0.0-M4 released Jan 16
JFrog
Latest Build
Build # 3
Build # 2
Build # 1

FIGURE 2-2: Using buildpacks to build container images saves time and improves security via automation.

TIP

The VMware Tanzu Build Service (`https://tanzu.vmware.com/build-service`) is an enterprise-grade product for automating container builds and updates, based on the Cloud Native Buildpacks technology. It's designed to work with Kubernetes, so operators can manage their build service using the same APIs they use to manage their Kubernetes clusters.

To see the potential benefits of container scanning and buildpacks on security, consider the not-so-hypothetical example of a major credit-rating bureau running an outdated web application framework as part of its software stack. In a DevSecOps environment like the one this book promotes, the component would be running in a container, and a regularly scheduled scan would likely have flagged the vulnerable version. At that point, the appropriate team could update the web framework's container image and upload it to a secure registry. In an application environment using buildpacks, a pipeline process would automatically update all affected containers instead of waiting months for a patch to be applied manually (or not at all).

Secure application catalogs

We've already discussed how organizations use private registries to store and secure their container images, so you may be wondering why they would also need an application catalog. After all, many organizations have been using "service catalogs" for years to provide vetted and secure applications to their developers.

The answer is that in a world of open source and containers, a well-designed application catalog acts as a trusted supplier to the registry — much like how toy manufacturers supplied the wares for the JCPenney holiday catalogs of yore, or how software vendors supply the applications that populate an organization's service catalog.

Figure 2-3 illustrates how a complete application catalog lets users select and customize open-source components, which are automatically updated and pushed to their chosen destinations.

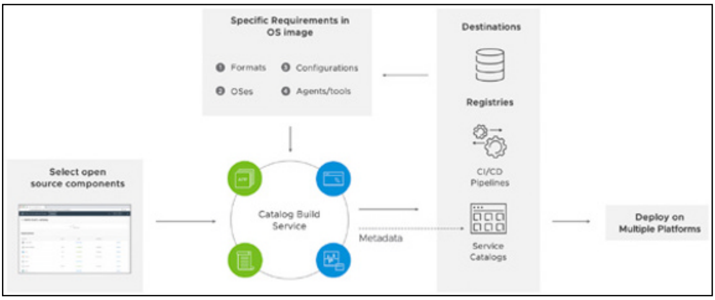


FIGURE 2-3: Finding container images for open-source tools is easy. Good application catalogs ensure they're up to date and secure.

There are multiple benefits to this arrangement, the primary one being that organizations and teams aren't responsible for building their own known-secure images for open-source applications. A good application catalog will maintain secure images for popular technologies and enable users to customize them to some degree to meet their specific requirements. Because DevSecOps centers around automation, the application catalog should also routinely update images as new versions become available and then push them to container registries and code repositories.

We highlight the open-source nature of the applications here because that's what makes an application catalog possible. Providers can serve many users by focusing on a curated collection of popular technologies, with most of the user-specific customization coming at the operating system level. Container images for software built in-house are still managed in-house, and container images for commercial software are still managed by those vendors.

TIP

VMware Tanzu Application Catalog (`https://tanzu.vmware.com/application-catalog`) integrates with your private container registry and offers more than 100 prebuilt and regularly updated container images for popular open-source applications. It also provides Helm files for direct deployment to Kubernetes clusters, as well as detailed metadata around what's running inside containers, so users can meet audit and compliance requirements. Tanzu Application Catalog is the enterprise offering of Bitnami, a trusted source of secure, prebuilt, and easy-to-use open-source applications in a variety of formats. VMware acquired Bitnami in 2019.

A large aerospace company that takes software development *and* security very seriously manages several dozen containers using Tanzu Application Catalog, including a custom version of the Postgres database designed for high availability. It updates those images daily with the latest operating system packages and upstream code, after which they're pushed to its Harbor registry and Git repository and are available to developers. Maintaining control of the provenance of its container images also enables the company to comply with government regulations.

Service meshes

Building and maintaining secure containers is only part of the DevSecOps puzzle, especially if you're also adopting microservices. In that case, you want to secure the communication between those services and the containers in which they're running. This is in addition to load balancing and other methods of ensuring performance for distributed applications. Today, many Kubernetes users do that by implementing a service mesh.

As shown in Figure 2-4, a *service mesh* works by deploying "sidecar" proxies in containers that (for lack of a better phrase) sit alongside application containers in the same Kubernetes pods. In a Kubernetes cluster with a service mesh installed, all network traffic between services running in the cluster goes through these proxies. This enables the service mesh to manage communication between the services and enforce policies. Those policies can cover a wide range of rules around traffic management, including security-specific concerns such as encryption, authentication, and access control.

One major benefit of this approach is that developers don't have to hard-code networking rules into their business logic. Another is improved application performance, because the individual containers can limit their resource usage to processing their primary tasks. In the case of a security incident or other error, service meshes log interactions between services, making it easier to track down the root cause or series of events that triggered the issue.

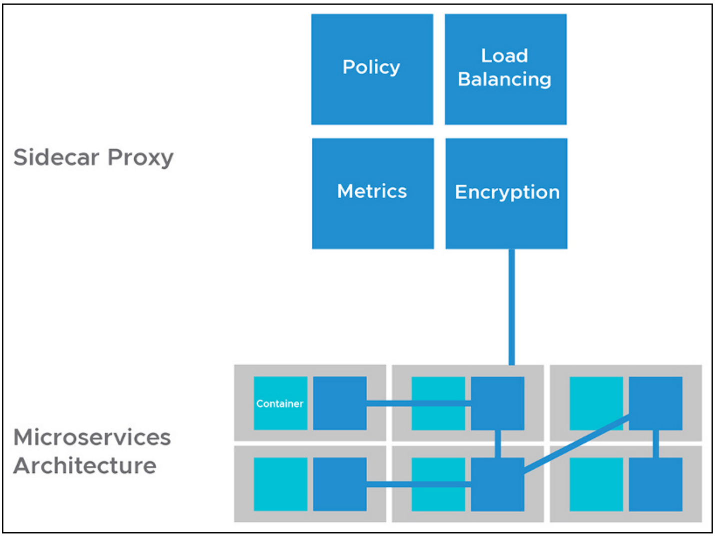


FIGURE 2-4: The service mesh proxies manage policies, logging, and other tasks, freeing up developers (and their application code) to focus on business logic.

TIP

VMware Tanzu Service Mesh (`https://tanzu.vmware.com/service-mesh`) combines standard service-mesh capabilities, based on the open-source Istio project, with advanced networking features that support applications running at a global scale. It allows for encrypted communications among services and load balancing across cloud regions, as well as autoscaling for services that reach capacity limits.

Shifting Left Respectfully with Platform Operations

As noted at the beginning of this chapter, the idea of "shifting left" refers to developers taking more responsibility than they historically have for application security.

You can think about the application life cycle as a spectrum that begins with planning and ends with monitoring production applications. In between are any number of steps (depending on the organization) that include writing code, building containers, testing, and deploying applications. In many legacy environments, developers' responsibilities stop after building their application packages and testing to ensure they work locally. This leaves security testing, production deployment, and the entire right half of the spectrum to security and operations teams. DevOps shifted certain operational tasks onto developers, and DevSecOps continues that trend by also shifting the security concerns we've discussed thus far onto developers.

However — and this is a *huge* however — DevSecOps really needs to be a partnership between developers and their peers in security and operations. Simply pushing more responsibility onto developers, very few of whom are also security experts, will likely lead to animosity and *shadow IT* (employees using unapproved tools or deploying applications in unmanaged public cloud accounts), as well as security issues. Instead, the ideal situation is one in which organizations and application teams take on a *platform operations* approach, also called *platform as a product.*

A platform operations team manages the application platform and treats it as a product for which developers are the customers. It's a shift in mindset, if not in job function, that helps ensure developers' concerns are considered when making decisions around tooling, processes, and changes. The goal is to implement guardrails that help ensure security and conformity to certain standards, while otherwise getting out of developers' way and letting them focus on their code.

In the context of DevSecOps, this means working with developers to determine which security measures they need to take on and how best to implement them. The more automatic those processes are, and the less troubleshooting the tools and systems require, the better the results will be for everybody involved.

Also, because the move to DevSecOps will likely be a learning experience for everyone — and because no system is perfect — developers and platform operations teams should plan for bumps in the road. There's a shift happening in IT toward eliminating blame and keeping open lines of communication about issues that arise. Those goals should certainly extend to the teams tasked with delivering DevSecOps. The alternative is an environment where security risks go unnoticed or unreported because people are afraid to admit mistakes or ask for help.

IN THIS CHAPTER

- **Learning the importance of Kubernetes for DevSecOps**
- **Standardizing on a Kubernetes distribution**
- **Automating and managing clusters across clouds**

Chapter 3
Managing Kubernetes Clusters with a DevSecOps Mindset

In Chapters 1 and 2, we talk mostly about technologies and processes that reside above the infrastructure layer. That's only logical considering how much of a DevSecOps workflow falls to developers. Even when a platform operations team installs the relevant tooling and keeps it all up to date, developers are the ones using it to build applications. However, operations staff still play an important role in keeping applications secure.

As you probably guessed, managing Kubernetes clusters is a big part of those responsibilities. It's important to remember that Kubernetes is software. Virtual machines, physical machines, and/or cloud instances exist to provide the underlying resources and must be kept patched. However, developers no longer deploy to those resources. Developers build container images by declaring dependencies and then deploy them to Kubernetes (which is installed on those machines and handles resource management between them and the containers).

TECHNICAL STUFF

For the platform team, the beauty of Kubernetes infrastructure abstraction is that it gives them more flexibility in how they provision resources for developers. They can improve security before DevSecOps even comes into play. A simple but effective example of this is choosing which kinds of resources a developer can access based on their level of expertise with Kubernetes. Developers or teams with limited knowledge might get a *namespace* (a pool of isolated resources on a shared cluster). More experienced developers or teams might get their own clusters. The difference is the degree of freedom they have; cluster-level access opens the doors to more controls and more opportunities to implement insecure configurations or neglect necessary patches.

Working at the Kubernetes level of abstraction results in a common language between developers and platform operations, wherever they're running workloads. Deploying to Kubernetes on the public cloud should be the same process as deploying to Kubernetes in a local data center or even on a laptop. And although operations staff need to understand the intricacies of each environment on which Kubernetes is installed, their interactions with developers can remain primarily at the Kubernetes layer.

Figure 3-1 provides a very high-level overview of a Kubernetes architecture, omitting many components necessary for running and managing a cluster. As you can see, however, Kubernetes provides a runtime for "pods" of containers and manages the interaction with the underlying infrastructure.

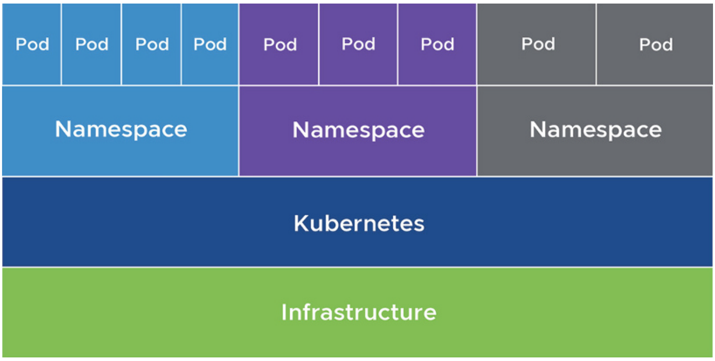


FIGURE 3-1: Individual teams can manage their own clusters or have access to a namespace on a multi-tenant, centrally managed cluster.

Standardizing on a Single Distribution

Not all Kubernetes distributions are created equal. If your organization isn't too deep down the Kubernetes path, it's worth weighing the options among upstream (or vanilla) Kubernetes, various vendor distributions, and managed services from cloud providers. The core application programming interfaces (APIs) and developer experience may be largely the same, but installation and management can be very different. From a security viewpoint, the lack of a standard Kubernetes distribution means more opportunity for configuration errors (with Kubernetes itself, as well as with its surrounding components) and a limited ability to implement organization-wide policies.

You can think about it like driving a car: Most people can perform the basic functions of driving in any car, but each car also comes with its own quirks that surface only when you're trying to adjust its settings, determine the meaning of a warning light, or make repairs. The results range from annoying to dangerous, depending on what goes wrong and where and when it happens. Likewise, it may be easy enough to install any Kubernetes cluster and start deploying applications to it, but you run into trouble with Day 2 concerns like configuration, management, and troubleshooting. When something breaks at the platform level, it's better to have a broad group of people who can fix it instead of needing to call in a specialist.

A less obvious benefit of choosing a standard Kubernetes distribution is that it limits the number of software vendors (or cloud providers) you need to manage. Despite its violent imagery, the old "single throat to choke" cliché still rings true for many organizations and probably rings even truer when talking about a platform-level technology. It's often necessary to work with many different vendors for application-specific components such as databases, but managing multiple vendors at the runtime or container-orchestration layer can add undue complexity and possibly cost. There are real benefits to consistency in terms of support, pricing, features, and integrations.

TIP

VMware Tanzu Kubernetes Grid (`https://tanzu.vmware.com/kubernetes-grid`) is an enterprise-grade distribution that checks a lot of important boxes for many organizations. It's especially well-suited to organizations already running VMware vSphere (whether internally or on public clouds), because administrators can provision Kubernetes clusters for developers using the vSphere interface. VMware Tanzu Advanced edition includes Tanzu Kubernetes Grid, as well as many other necessary components for DevSecOps and modern applications in general.

Workload portability is another major benefit of running Kubernetes, in general, that is amplified by also choosing a standard Kubernetes distribution. Although being able to move applications relatively freely across environments (public cloud, private cloud, edge, and so on) is not a core component of DevSecOps per se, it *is* an important consideration in many broader IT modernization efforts. The ease of moving workloads will vary based on your specific situation. Architecture, tooling, data gravity, and other factors will contribute to this. However, choosing to use the same Kubernetes distribution in every environment at least provides a consistent experience around installation, deployment, and maintenance.

In Chapter 2, we mention that a big goal for DevSecOps, for developers, is to make the right thing the easy thing. The same holds true for operations. Kubernetes is the part of the DevSecOps stack over which operations has the most control, and they should seek out a distribution that meets their requirements around portability, security, integrations, and whatever else they deem important.

Managing All Your Clusters from a Single Interface

Life is often a little more complex than we'd like, and standardization isn't always possible. Although it might be obvious that running a single Kubernetes distribution on a single cloud platform would make life easier, for any given organization there may be a million reasons why it can't be done — acquisitions, different teams' needs, competitive concerns, and existing vendor contracts, to name just a few.

Truth be told, though, managing clusters isn't always a walk in the park even when standardization *is* possible. Differences among the teams that own those clusters are a big part of why this is so. For example, those with more experience may be better equipped to handle updates and general cluster maintenance, while those with less experience may need more handholding. Unfortunately for platform operations teams, they're expected to maintain the security of the platform and its applications regardless of each team's relative Kubernetes skill set.

That burden can quickly become overwhelming as Kubernetes usage grows within an organization. Ops teams must track dozens — possibly hundreds — of individual clusters and namespaces to account for their compliance with security policies and to ensure that the Kubernetes runtime, operating systems, and other key components are kept up to date. If there's a vulnerability in Kubernetes itself, the platform ops team must patch each affected cluster as soon as possible. The more regulated the industry, the greater the potential damage of falling back into legacy timelines where it could take months to roll out patches across all of an organization's systems.

As one platform architect at a large brokerage firm (and VMware Tanzu customer) put it (only half-jokingly), "I can only run the Kubernetes commands across all the clusters so many times. I only have so many keystrokes in life, and they're running low."

Automating Kubernetes management

Addressing this problem with a DevSecOps approach means centralizing cluster management responsibilities within the platform operations team, and then automating and simplifying as much of the management process as possible. Ideally, you're working with a system that can handle clusters regardless of where they're hosted, what distribution they're running, and who owns them. This way, you can push updates and otherwise act accordingly depending on what Kubernetes runtime version a cluster is running, where a vulnerability resides (for example, at the distribution, server, or cloud provider layer), or, in the case of new security policies or regulatory requirements, which business units they affect.

Centralizing control of identity management and authentication is also critical. It only takes a single misconfigured permission to give attackers access to sensitive resources or data. Although sophisticated, well-funded attacks make headlines, criminals are still able to breach targets via loose permissions or, in far too many cases, resources exposed to the Internet without so much as a password. This is why it's important to ensure that in addition to being easy to deploy for development teams, Kubernetes clusters also utilize approved identity and access management systems by default.

DON'T OVERLOOK AUTHENTICATION

In 2017 and 2018, hackers were able to access Kubernetes clusters belonging to several companies, including some household names, via unsecured administrative consoles. The attackers used their access to mine cryptocurrency using corporate resources, but the damage could've been much worse. This is also the kind of thing that should never happen in a well-run DevSecOps environment, because enforcing passwords should be Step 1 in any security protocol.

Automatic cluster backup is an increasingly important capability as well. The recent spate of ransomware attacks is certainly a big reason to ensure that you can recover clusters, but ransomware is not the only threat. Cloud-computing platforms are getting more reliable by the day, but they're incredibly complex systems running workloads for, in some cases, millions of users. Outages can knock clusters offline for days and even result in data loss; regular backups can help mitigate the damage.

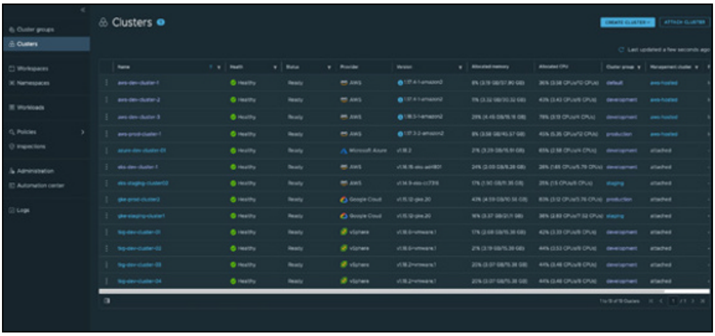
Balancing control and flexibility

A platform ops teams will ideally have as much control as possible over everything from cluster configuration to network rules, as well as the flexibility to enforce those policies as necessary. That latter part may seem contradictory to the stated goals of DevSecOps, but sometimes the best way to improve is to see what mistakes are happening. If most teams follow best practices around a certain configuration option, and if auto-enforcing it would cause unnecessary friction, then the status quo probably can hold. But if teams are ignoring critical policies even after warnings, it's probably time to step up enforcement.

Again, everything comes back to the *make the right thing the easy thing* mantra. Kubernetes can be a powerful piece of your modernization strategy, enabling you to prioritize the easy provisioning of resources to developers who want them. However, Kubernetes is a complex system in an increasingly complex IT landscape, and it's easy for vulnerabilities to creep in as usage expands across teams, clouds, and geographies. Using it safely means pairing ease of access with guardrails and the ability to enforce policies with minimal manual effort.

TIP

VMware Tanzu Mission Control (`https://tanzu.vmware.com/mission-control`), shown in Figure 3-2, is a centralized management platform for consistently operating and securing your Kubernetes infrastructure across teams, clouds, and distributions. It's based on open-source technologies and, in addition to security and management capabilities, it supports the provisioning, scaling, and deletion of Kubernetes clusters.



Name	Health	Status	Provider	Version	Allocated memory	Allocated CPU	Cluster group	Management cluster
aws-dev-cluster-1	Healthy	Ready	AWS	1.17.4-1-amazon2	[illegible]	[illegible]	default	aws-hosted
aws-dev-cluster-2	Healthy	Ready	AWS	1.17.4-1-amazon2	[illegible]	[illegible]	development	aws-hosted
aws-dev-cluster-3	Healthy	Ready	AWS	1.18.5-1-amazon2	[illegible]	[illegible]	development	aws-hosted
aws-prod-cluster-1	Healthy	Ready	AWS	1.17.3.2-amazon2	[illegible]	[illegible]	production	aws-hosted
azure-dev-cluster-01	Healthy	Ready	Microsoft Azure	v1.18.2	[illegible]	[illegible]	development	attached
eks-dev-cluster-1	Healthy	Ready	AWS	[illegible]	[illegible]	[illegible]	development	attached
eks-staging-cluster02	Healthy	Ready	AWS	[illegible]	[illegible]	[illegible]	staging	attached
gke-prod-cluster2	Healthy	Ready	Google Cloud	v1.15.12-gke.20	[illegible]	[illegible]	production	attached
gke-staging-cluster1	Healthy	Ready	Google Cloud	v1.15.12-gke.20	[illegible]	[illegible]	staging	attached
tkg-dev-cluster-01	Healthy	Ready	vSphere	v1.18.6+vmware.1	[illegible]	[illegible]	development	attached
tkg-dev-cluster-02	Healthy	Ready	vSphere	v1.18.6+vmware.1	[illegible]	[illegible]	development	attached
tkg-dev-cluster-03	Healthy	Ready	vSphere	v1.18.2+vmware.1	[illegible]	[illegible]	development	attached
tkg-dev-cluster-04	Healthy	Ready	vSphere	v1.18.2+vmware.1	[illegible]	[illegible]	development	attached

FIGURE 3-2: The Tanzu Mission Control user interface (in mid-2021). Note the ability of administrators to view and manage Kubernetes clusters across multiple platforms and to set security policies.

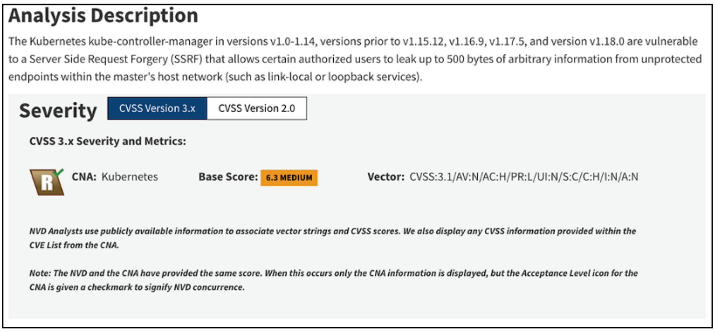
What Happens When a Kubernetes Vulnerability Is Discovered?

Although Kubernetes has proved remarkably secure over the course of its existence (around seven years as of the publication of this book), no software is perfect. As we explain earlier in this chapter, in the example of Kubernetes clusters without passwords inside large organizations, user error is a very real concern. Like all software, Kubernetes is not immune to vulnerabilities. In fact, the number has been growing each year as its codebase expands and more vendors build products around it. (In 2020, there were a few dozen vulnerabilities discovered that involved Kubernetes to some degree.)

Vulnerabilities aren't the real problem, though. The vast majority of software contains vulnerabilities, and their discovery can often be correlated with popularity. As more people use a given technology, it becomes more interesting to both cybercriminals and security researchers. More eyes on a codebase means more eyes to discover problems.

As we've noted a couple of times already, a major problem with vulnerabilities is the time it can take to patch them. This problem is made worse in an organization that runs multiple Kubernetes clusters, because different teams may be running different release versions and different distributions, and they may differ in how quickly they update after a patch is released. Kubernetes — and many popular open-source projects — can further exacerbate the issue because updates are often released at a pace faster than what many organizations are used to. This type of situation can be mitigated when platform operations teams can enforce update policies across any number of Kubernetes environments.

Let's take, for example, CVE-2020-8555, which affects the Kubernetes kube-controller-manager component and could result in unauthorized data leakage. As shown in Figure 3-3, the vulnerability has a severity score of 6.3 and affects versions v1.0–1.14, versions prior to v1.15.12, v1.16.9, v1.17.5, and version v1.18.0. A platform operations team, upon learning of this vulnerability and assessing the risk it poses, could use a tool like Tanzu Mission Control to identify which versions each team is running, update any vulnerable clusters under its control (or request that cluster owners perform those updates), or apply any security policies that might resolve the issue prior to a full upgrade.



Analysis Description

The Kubernetes kube-controller-manager in versions v1.0-1.14, versions prior to v1.15.12, v1.16.9, v1.17.5, and version v1.18.0 are vulnerable to a Server Side Request Forgery (SSRF) that allows certain authorized users to leak up to 500 bytes of arbitrary information from unprotected endpoints within the master's host network (such as link-local or loopback services).

Severity

CVSS 3.x Severity and Metrics:

CNA: Kubernetes **Base Score:** 6.3 MEDIUM **Vector:** CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:N/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: The NVD and the CNA have provided the same score. When this occurs only the CNA information is displayed, but the Acceptance Level icon for the CNA is given a checkmark to signify NVD concurrence.

FIGURE 3-3: A sample of information for CVE-2020-8555, as provided by the National Vulnerability Database (`https://nvd.nist.gov`).

By contrast, an operations team without that level of visibility and control must rely on more manual methods to track and update clusters. In a worst-case scenario, the team may have to rely on organization-wide communications imploring cluster owners to verify the releases they're running and update if necessary, or they may have to manually update each cluster on a case-by-case basis. Manual updates can be prone to error and possibly unintended downtime.

On a related note, it's worth mentioning that the Kubernetes project releases a new minor version about once a year (for example, 1.19 or 1.20). They limit long-term support for each version to one year. Prior to version 1.19, that support window was only nine months. So, it's easy for teams still operating on more traditional upgrade schedules to fall behind and continue running unsupported versions. This is another issue that an organization can mitigate by embracing DevSecOps and enabling platform operations teams to maintain a modern upgrade schedule.

Chapter 4
Ten More Resources to Guide Your Journey

We hope that this book is a useful introduction to the principles behind DevSecOps and some of the pieces that you'll want to make sure are part of your organization's environment. Just remember that the technologies that enable DevSecOps are complementary to the technological, procedural, and cultural changes that come along with a broader move to DevOps. If you're going to meaningfully improve software development, along with application quality and application security, you'll want to embrace change up, down, and across your IT organization.

With that in mind, here are some additional reading materials that add context around DevOps, cloud-native computing, and the benefits of partnering with VMware Tanzu on your DevSecOps journey:

- **3 Transformations That Are Remaking Enterprise IT (**`https://tanzu.vmware.com/three-transformations`**):** A curated collection of resources about how changes at the infrastructural, architectural, and operational levels are evolving how we build and interact with applications. Learn more about DevSecOps concepts such as Kubernetes, microservices, and a lot more.

- **11 Recommended Security Practices to Manage the Container Lifecycle (**`https://tanzu.vmware.com/content/white-papers/11-recommended-security-practices-to-manage-the-container-lifecycle`**):** More-technical dives into some of the topics and technologies discussed in this book, as well as additional guidance on container security as a whole.
- **Defining and Delivering DevSecOps Across Your IT Organization (**`https://tanzu.vmware.com/content/webinars/aug-5-defining-and-delivering-across-your-it-organization`**):** This webinar features experts in the area of DevSecOps — development, security, and operations — discussing strategies for making sure everyone has a seat at the table when rolling it out. The bigger the organization, the easier it is to overlook important voices.
- **KubeAcademy (**`https://kube.academy`**):** Free courses that cover Kubernetes from the ground up, taught by cloud-native experts. Dig deeper into some of the ideas in this book by taking the seven-lesson Kubernetes Platform Security course (`https://kube.academy/courses/kubernetes-platform-security`).
- **Paving the Road to Modern Apps (**`https://tanzu.vmware.com/modern-apps`**):** An introduction to the people, products, and vision behind VMware Tanzu, as well as case studies from a wide range of customers. It's a good place to get a sense of what's possible when organizations embrace not just DevSecOps, but modernization in general.
- **SpringOne (**`https://springone.io`**):** In name, SpringOne is an annual conference exploring best practices for software development using the Spring framework. In practice, it's that *plus* an invaluable collection of talks covering DevOps, management strategies, and everything that goes into running a modern business. Watch sessions from the most recent event on the SpringOne website, or find previous years' sessions on the VMware Tanzu YouTube channel (`www.youtube.com/c/VMwareTanzu/playlists`).
- **Tanzu Developer Center (**`https://tanzu.vmware.com/developer`**):** Guides, demos, and other technical material designed to educate readers on all things cloud native and

DevSecOps. Good starting points for expanding on this book include guides on containers (`https://tanzu.vmware.com/developer/guides/containers/`) and microservices (`https://tanzu.vmware.com/developer/guides/microservices/`).

- **Tanzu Talk and Cloud & Culture (**`https://tanzutalk.com`**):** Two podcasts sharing a single stream, covering application modernization and digital transformation from every angle. Guests range from VMware Tanzu experts — including on this episode about securing your software supply chain (`www.tanzutalk.com/222`) — to customers who have successfully evolved their software development for the cloud-native era.
- **TGIK (**`https://tanzu.vmware.com/developer/tv/tgik/`**):** The *K* stands for Kubernetes in this weekly video series targeting Kubernetes practitioners, often hosted by one of the technology's creators. If you're interested in learning more about the Harbor container registry, for example, there's an episode for that (`https://tanzu.vmware.com/developer/tv/tgik/138/`).
- **VMware Tanzu Labs (**`https://tanzu.vmware.com/labs`**):** Although its name has changed over the years, the techniques — and even some of the team — at VMware Tanzu Labs helped some of the world's most well-known organizations implement the right techniques and culture for delivering better software and happier employees. Tanzu Labs experts engage deeply with clients on everything from modernizing specific applications to data center migration.

Notes

Notes

VMware Tanzu

Advanced Edition

Make DevSecOps possible with VMware Tanzu Advanced

Reduce the complexity and risk of running modern apps in production, and transform your security team from the "Department of No" into close collaborators on better software.

DevSecOps is the key to fast, effective, and secure application development

Software has already eaten the world, and the DevOps movement helped make that happen. We're shipping software faster than ever and using technologies — including containers and Kubernetes — that blur the once-distinct lines between application development and operations. However, with modern practices and technologies come new security concerns. By embracing a DevSecOps mindset, organizations can take advantage of these advances without putting themselves or their customers at risk.

VMware Tanzu™

Brad Bock is a senior manager of product marketing for VMware Tanzu. His career in tech began at open-source cloud applications startup Bitnami, followed by product marketing roles on the Networking & Security and Modern Applications Platform teams at VMware

Inside…

- Get familiar with core DevSecOps principles
- Learn the basics of container security
- Understand how to manage Kubernetes safely
- Prepare for the next generation of enterprise IT

ISBN: 978-1-119-83256-0
Not For Resale

9 781119 832560

for dummies®
A Wiley Brand