O'REILLY®

Compliments of

snyk

Cloud Native Application Security

Embracing Developer-First Security for the Cloud Era

Guy Podjarny

REPORT

See why
2.6 million developers
choose Snyk to secure
their cloud native
applications

Cloud Native Application Security

Embracing Developer-First Security for the Cloud Era

Guy Podjarny

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

Cloud Native Application Security

by Guy Podjarny

Copyright © 2021 O'Reilly Media, Inc. All rights reserved.

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

Acquisitions Editor: Mary Preap
Development Editor: Virginia Wilson
Production Editor: Katherine Tozer
Copyeditor: nSight, Inc.
Proofreader: Audrey Doyle
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

June 2021: First Edition

Revision History for the First Edition

2021-05-18: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Cloud Native Application Security*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Snyk. See our statement of editorial independence.

978-1-098-10560-0

[LSI]

Table of Contents

Preface v

1. Digital Transformation 1
- Becoming a Technology Company 1
- Accelerating Technology Delivery 2
- Security and Cloud Native Development 5
- Conclusion 6

2. Dev-First Security 7
- What Is Dev-First Security? 8
- Shift Left Is Not Enough 14
- DevSecOps Versus Dev-First Security 15
- Conclusion 17

3. Securing the Entire Cloud Native App 19
- From IT Security to Cloud Security 19
- From Cloud Security to Cloud Native Application Security 21
- Container Application Security 22
- IaC Application Security 27
- Conclusion 31

4. Adapting to Dev-First CNAS 33
- Rethinking the Security Org Structure 33
- Rethinking Tooling 38
- Rethinking Priorities 43
- Conclusion 45

Summary 47

Preface

Cloud native applications don't just run on a different platform; they overhaul the scope of the applications, the methodologies with which they're built, and the skills and ownership around them. To stay relevant, security practices need to undergo a transformation of a similar magnitude. We have to embrace a developer-first (dev-first), Cloud Native Application Security approach and anchor our practices to this new organizational reality.

This book will help you understand the market transition to cloud native apps and the organizational changes it includes. Armed with this context, it describes the required changes for our security practices and tooling and why they matter. Last, it provides concrete examples for some of these changes, meant to both help you understand the concepts better and act as candidates for the first steps you may take on the journey.

By the end of this book, you should understand why and how to embrace a Cloud Native Application Security approach.

Why I Wrote This Book

I'm a technophile and I believe technology can help us solve many of the world's problems and tap into the opportunities they present. I'm therefore truly excited by how much better and easier software development has become, making it easier for creativity to thrive and bring innovative and high-impact changes to our daily lives.

I'm also truly worried about our ability to keep safe in a digital world. After more than 20 years in the security industry, I know all too well how fragile applications are and how easy it is to ignore

risks when building software. The drive to improve our lives and businesses with technology cannot—and should not—be stopped, but without building security in, we may hurt ourselves even more.

I believe the only solution is to build security into the fabric of software development; no other approach can keep up. This is a big change, requiring a true transformation of our security practices, technologies, and attitude. It goes against the natural evolution of cyberspace and, thus, requires a revolution, not evolution.

This is the reason I founded Snyk, a company dedicated to building security into development, and it's the reason I'm writing this book. The first step in transforming security is to catch up to the one that took place in development: the adoption of DevOps and cloud, and the resulting cloud native model.

My hope is that this book will help readers understand what it means to embrace Cloud Native Application Security, why it matters, and how to get started. If more people take on such a change, adapted to their needs, we'll be a step closer to a safer digital world.

Who Is This Book For?

At its core, this book is for anyone seeking to adapt security to the cloud era. This includes organizations big and small and practitioners from many parts of the organization, including development, operations, security, and executive leadership. The book is written to be accessible to all those readers, even if they have only light familiarity with how software is developed and operated.

More specifically, the book is aimed at leaders and senior individuals in security and development organizations because those two groups are the primary ones responsible for building secure applications, but who most need to adapt to the new reality.

When I use the term *you*, I'm referring to any technology leader in the company who shares responsibility for building secure applications. When I use the term *we*, I'm referring to the surrounding development, security, and operations (DevSecOps) community as a whole.

CHAPTER 1

Digital Transformation

Technology is reshaping practically every industry vertical, and businesses need to adapt or they will be left behind. This tidal wave affects all businesses, requiring them to perform a digital transformation to face the new times.

Digital transformation typically includes two main parts:

- Becoming a technology company if you're not one already
- Accelerating delivery of technology solutions, notably via DevOps and the cloud

The term *cloud native* is used to describe organizations, applications, and development processes designed for this new reality and its new platforms.

Let's dive deeper into what each of these bullets means.

Becoming a Technology Company

For certain industries, digital transformation requires changing how they transact with customers, turning physical goods into digital ones. Banks and shops are moving from physical branches to online portals and ecommerce, passengers hail taxis through mobile apps instead of hand waving or calling a taxi station, and media is increasingly produced through crowdsourcing and consumed over the internet instead of via cable TV.

Such massive changes also require rethinking how to secure the org (organization) and adjust priorities. As more customer data and business transactions shift online, digital disruptions or breaches become more costly, and therefore the importance of cybersecurity grows. To address the change, security budgets must allow for a bigger spend on cybersecurity, security teams need more cyber-related skills, and even the chief security officer (CSO) may be required to master different skills or be placed elsewhere in the org.

Every business aspires to become "a technology company that does X," which in turn means adopting a technology-focused security program.

Accelerating Technology Delivery

The second focus of digital transformation is speed. Although responding faster to market needs has always been a powerful edge in business, the pace has never been quicker. Thanks to technology and process innovations, businesses have gone from annual releases to shipping multiple times a day. Users have grown to expect such constant updates, and businesses that fail to adapt will struggle financially.

This speed is made possible by two primary forces: the cloud and DevOps. Each term represents a family of technologies and practices, respectively, and has a significant impact on security. Let's take a moment to define them.

The Cloud

The cloud represents a series of technological developments that turned hardware into APIs, letting developers access and control physical resources purely with software. The shift to software introduces dramatic ease and flexibility and is best exemplified by elastic compute, pioneered by the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) service. EC2 allows users to rent a virtual machine (VM) by the hour and spin it up or down with a simple API call. This allows a service to adapt its capacity dynamically, using little compute when demand is low, incurring low costs, while automatically scaling up to handle surges in demand and avoid losing business.

Elastic compute and the broader family of cloud technologies combine to give developers full control over rich infrastructure controls. These include additional APIs to control other infrastructure components, such as storage and network access; developer-oriented replacements to IT tooling, such as containers and Infrastructure-as-Code (IaC) solutions;[1] and new technology paradigms that build on this elasticity, such as serverless.

The cloud unlocked the opportunity for any company to provide operational excellence previously reserved to a select few and do so with dramatically lower costs. However, for a company to tap into the power of the cloud, it also has to change its IT processes, which is where DevOps comes in.

DevOps

DevOps changed the way software is built and maintained. It is predicated on independent development teams, who are able to own the application end to end. The most immediate manifestation of that is having developers also operate the application, dubbed "you build it, you run it," hence the name DevOps. However, the term has grown to encompass a broader autonomy for development teams, including empowerment to make more business decisions, choose the underlying technologies they use, and more.

This autonomy powers speed, notably in the form of continuous delivery (CD). Instead of shipping updates in large, infrequent batches, CD means shipping small improvements regularly, often multiple times a day. Each incremental step is verified with automated tests to avoid delays, and observability tools to flag problems in production as soon as they happen.

Such constant updates allow the business to adapt to customer needs faster, which improves its commercial performance. This continuous flow of faster delivery leading to faster market adaptation is most commonly portrayed as an infinity loop, as shown in Figure 1-1, replacing older waterfall, left-to-right models.

1 Containers are lightweight packaging of operating system and files, offering a type of virtual machine but smaller and lighter, allowing better agility and speed. IaC solutions are tools that define infrastructure in code or declarative files, often stored in repos as part of the app, and can automatically apply those definitions on an infra platform.

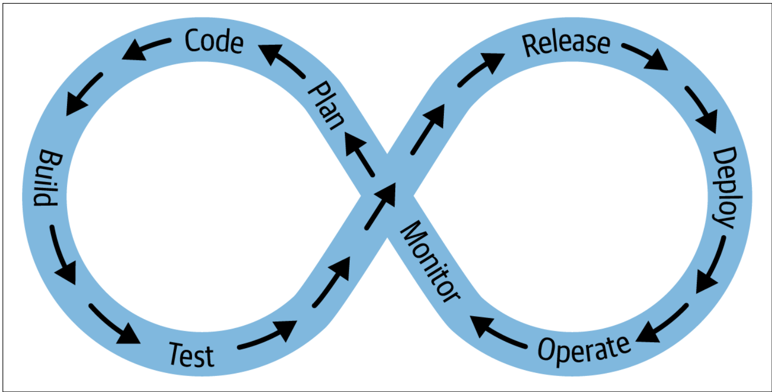

Figure 1-1. A continuous DevOps practice

Cloud Native

Organizations that started building technology in this DevOps and cloud surrounding are often referred to as cloud native companies. Their developers embrace on-call responsibilities, monitoring dashboards regularly and being paged if the service goes down, whereas their ops teams focus on building platforms that make it easier to build operable software.

Such software engineering organizations are often called cloud native development teams because they are designed—across people, process, and technology—to leverage the cloud to the max. The term has grown to also encompass organizations that modernize their practices and adopt such cloud native practices, on both public and private cloud platforms.

Similarly, applications built using cloud native development are referred to as cloud native apps. This isn't just a historical honorific, but rather, a description of how these applications are maintained. It implies that these apps are constantly updated, can scale to large volumes, and leverage infrastructure automation to provide a better customer experience.

Organizationally, it's important to understand that cloud native applications have a bigger scope than their predecessors had. Older applications are made up mostly of code and open source libraries and rely on a central IT team to provide them with infrastructure, such as hardware, VMs, network access, databases, and more.

Cloud native apps, by contrast, include such underlying infrastructure as part of their scope. Hardware and VMs are replaced by containers and managed through Dockerfiles stored in the source code's repository;[2] network configuration is specified in IaC files such as Terraform and Kubernetes configurations; and databases or similar supporting apps are provided as services in the cloud platform itself, and managed by the application team itself.

This change in the scope of the application is shown in Figure 1-2, using a somewhat arbitrary before-and-after view of pre-cloud and cloud native apps.

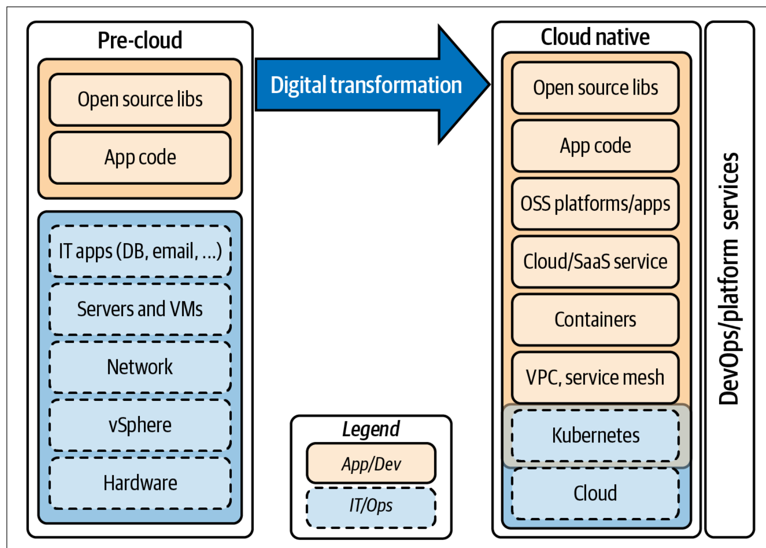

Figure 1-2. Cloud native applications include a much broader scope

Security and Cloud Native Development

The security industry, unfortunately, hasn't kept up with this market shift. In the vast majority of organizations, security teams still cling to their past methodologies. They continue to employ minimally changed practices, despite dramatic changes to those they help protect.

2 Dockerfile is the most common way to define what a container should hold. These are human editable files, typically managed as source code. Running a `build` command on a Docker image creates a container image.

Despite DevOps preaching end-to-end ownership by independent teams, security is typically owned by a separate team. At best, this split causes bottlenecks as application teams wait for a central team to assess an app or review findings. At worst, it leads to serious security flaws being overlooked because development teams lack tools and knowledge in security, and security teams lack knowledge of the app.

Infrastructure security continues to rely on the same IT-oriented practices, ignoring the fact that these layers became part of the broader cloud native application scope and are managed by developers. These practices again cause delays and security oversights, with neither development teams nor IT teams being set up to address the risk properly.

These approaches are doomed to failure because they go against the change the business strives to achieve.

Conclusion

Digital transformation is aptly named; it transforms organizations, changing their business trajectory to be technology focused, with the tools and practices required to succeed. This change is far reaching, affecting anything from org structures and business incentives to development methodologies and technology stacks.

Cloud native applications are designed to thrive in this new reality. They rely on independent teams, able to deliver customer value end to end. They use DevOps methodologies and cloud technologies to empower these teams to succeed. Unfortunately, security functions haven't kept up with this change and are continuing to operate largely the same as before, making them ineffective and a burden on the business.

To secure cloud native applications successfully, security practices need to undergo a transformation that matches the one that development took on. We have to embrace a dev-first, Cloud Native Application Security approach and anchor our practices to this new organizational reality.

In the next two chapters, we'll dive deeper into what dev-first security and Cloud Native Application Security mean, and how you can successfully implement them in your organization.

CHAPTER 2

Dev-First Security

As mentioned, the security industry hasn't been a part of the DevOps journey. As shown in Figure 2-1, security processes tend to gate the continuous process instead of merging into it. Notably, security processes are incapable of the following:

Empowering independent dev teams
: Security is owned by a separate team, dev teams are not empowered to make security decisions, and tooling is designed primarily for auditors, not builders.

Operating continuously
: Security processes still heavily rely on manual gates such as audits or result reviews, slowing down the continuous process.

Having security work against the business motivation of speed and independence can't end well. Development teams must choose between slowing down, which hurts business outcomes, and circumventing the security controls, which introduces significant risk. Neither of these is a viable long-term option, so businesses must change their security practices to match the DevOps reality.

To secure the business without slowing it down, companies must adopt a dev-first approach to security.

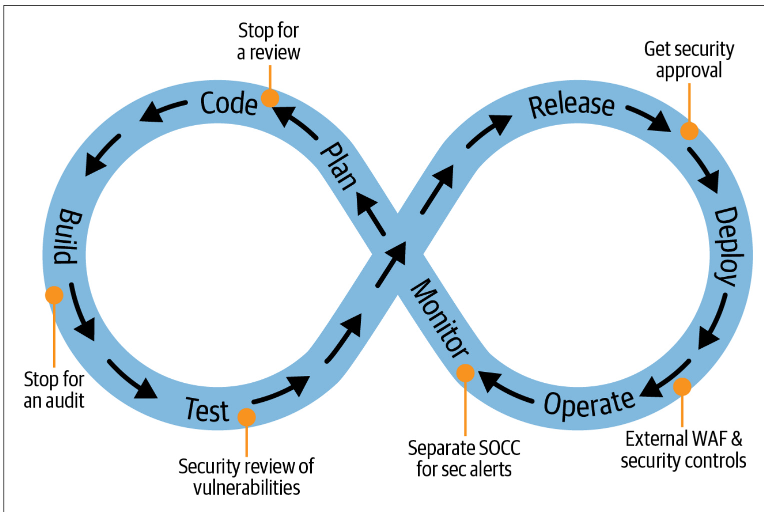

Figure 2-1. Security gates slowing down continuous delivery process

What Is Dev-First Security?

Security programs must always start with an understanding of the risks you face. If you don't know what you're looking to secure or who you're protecting yourself from, you're likely to place your guards in the wrong place. However, past that understanding, you have a choice to make: do you prioritize the need to track that risk, or the need to protect against it?

Most security organizations, whether consciously or not, choose the former. The teams tasked to secure the organization (and holding the budget to do so) are the ones in charge of audit and governance, and they naturally place the need to understand your risk at the top. This results in products that are focused on finding problems and are attuned to a security person's needs and understanding. These tools are then retrofitted to be put into the development pipelines but are unable to deliver on what developers need.

As its name implies, dev-first security means reversing the order: putting the developer's needs at the top of the priority list when looking to secure your applications. It means asking yourself, "If I'm a developer looking to build a secure application, what do I need to do so successfully?" These tools still care about the security team's needs and understanding your security posture, but they *focus* on helping *developers* secure what they build.

This is a completely different angle and results in building entirely different solutions to the same security threat. Let's review a few key differences.

Developer Context and Expertise

Developers operate in a different context and have different expertise than security folks do.

The most obvious difference is the lower level of security expertise. Most developers won't know what Common Vulnerabilities and Exposures (CVE) or Common Vulnerability Scoring System (CVSS) means,[1] or that they should ask whether a known vulnerability has a published exploit. These details must be simplified in the security tools developers are given: for instance, simplifying CVSS to three or four levels or highlighting important attributes like exploit maturity. Figure 2-2 shows an example of such built-in expertise.

A less-known difference is that developers see vulnerabilities through the lens of the app, not through risk. When a security person sees a vulnerable library, they see risk. If they zoom out, they'll want to see other vulnerable libraries or other apps with the same flaw. Therefore, most security tools focus on governance dashboards that show lists of all vulnerable assets.

Figure 2-2. Simplifying CVSS to a four-scale level and highlighting a mature exploit in the wild

1 A CVE number is a universal ID for a known vulnerability, used to synchronize between different tools referring to the same issue. The CVSS is a standard way of defining the severity of a vulnerability, using a small set of predefined criteria.

For developers, though, the first question isn't about risk, but rather, about how the library connects to and affects the app. They don't look for other vulnerable libraries, but for other properties of the same library, such as how outdated it is, so they can weigh the impact of upgrading or replacing it. Figure 2-3 shows the difference between a security-focused flat list of vulnerabilities, which focuses on the risk, and a dev-first tree view of vulnerabilities, which focuses on relation to the app.

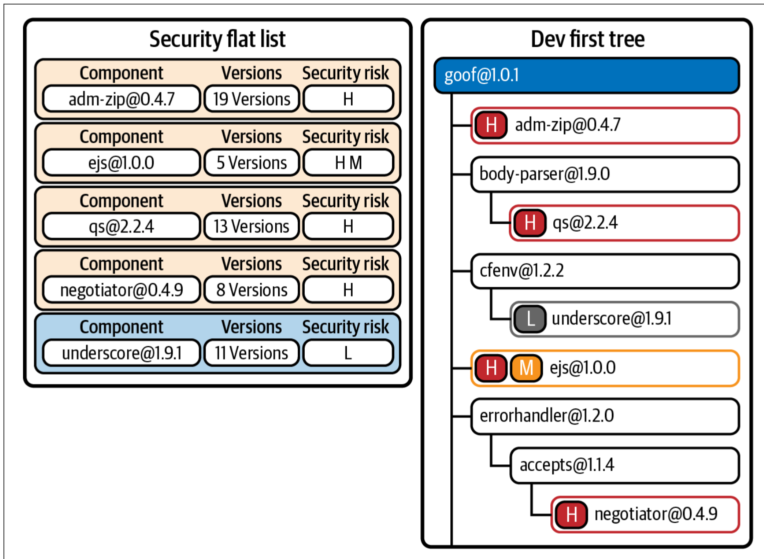

Figure 2-3. Tree view and flat view

Last, although developers may lack security expertise, their knowledge of the application is far better than that of the security team. For instance, a developer is likely to know whether a library is only used during development, or whether certain functionality is only accessible to administrative users. A good dev-first security solution can leverage this knowledge for better usability and results.

Developer Experience and Affinity

No tool lives in isolation. Developers and security people alike judge tools in the context of other tools they use around them and expect solutions to be a good citizen in their local ecosystem. A tool that deviates from the norms requires more attention and thought, requires more time to become proficient in it, and introduces

cognitive load whenever you switch to it. Unless you are very motivated to use them, such odd ducks are best avoided.

Although this statement is true for both teams, those neighboring tools are massively different. Security people use a large number of auditing, governance, and compliance tools. They therefore expect other solutions to work well in an audit context, offering functionality such as rich listing of results, exports to PDF, and integration with risk dashboards. They have high tolerance for long tasks and typically assume an expert user who wants to see all the info.

For developers, the surrounding tools are build tools. They focus on helping individuals write code faster, identify and resolve problems locally, and collaborate with teammates in a version-controlled fashion. They assume highly technical users, and look for automated and fast tests to achieve yes/no answers so that they can be added to the build.

These are just a few of many areas of difference. Developer tools also offer different user experience (UX) patterns than security tools do, are more often available to try self-serve, and typically offer rich and well-documented APIs. Beyond the product, developer tooling *companies* behave differently, leaning toward more community collaboration and transparency, focusing on building versus risk reduction, and so on.

Dev-first security solutions need to embrace the dev tooling ecosystem as their peers. This requirement applies to commercial, open source, and home-grown tools alike, because at the end of the day, they all need to be a natural part of a developer's daily routine. These tools also need to satisfy the needs of the security realm, offering the right views and integrations, but the priority has to be clear: developer experience comes first.

Security Audits, Developers Fix

> *"Whereas an auditor's job is to find and prioritize issues, a developer's job is to fix them."*

Beyond context and experience, a developer looking at a vulnerability has a different job than the security person has. Security leaders are tasked with understanding the flaws and risks in the system and helping prioritize and act on that understanding. As a result, security tools and practices excel at finding security flaws, assessing their

technical and business risks, and managing this list of vulnerabilities over time.

Whereas an auditor's job is to find and prioritize issues, a developer's job is to fix them. Good auditors and security teams also aspire to have issues fixed, but they have little control over accomplishing this. In fact, many security tools advertise logging a bug-tracking ticket as a remediation action, despite that it doesn't actually fix anything. For developers, a tool that reports problems without helping to resolve them isn't seen in a favorable light.

A dev-first security solution must therefore have a strong focus on fixing issues. For every reported issue, you should ask yourself what a developer needs to do to resolve the issue. How can the tool help simplify this task? The answers will help you walk the extra steps from the auditor's point of view to the developer's needs.

Ideally, the tool would be able to remediate the problem automatically, saving the developer precious time. When that's not possible, consider what scaffolding you can still offer to simplify remediation: for instance, by prompting developers to approve or decide on the fix but still automating the process itself. See Figure 2-4.

One note of caution, though: there's a difference between automating remediation and commandeering control. The security world has a long history of solutions that aim to detect and block attacks unilaterally, ranging from intrusion prevention systems (IPSs) to web app firewalls (WAFs). More recently, runtime application self-protection (RASP) solutions have been making the same claim, instrumenting applications to build security controls into them automatically.

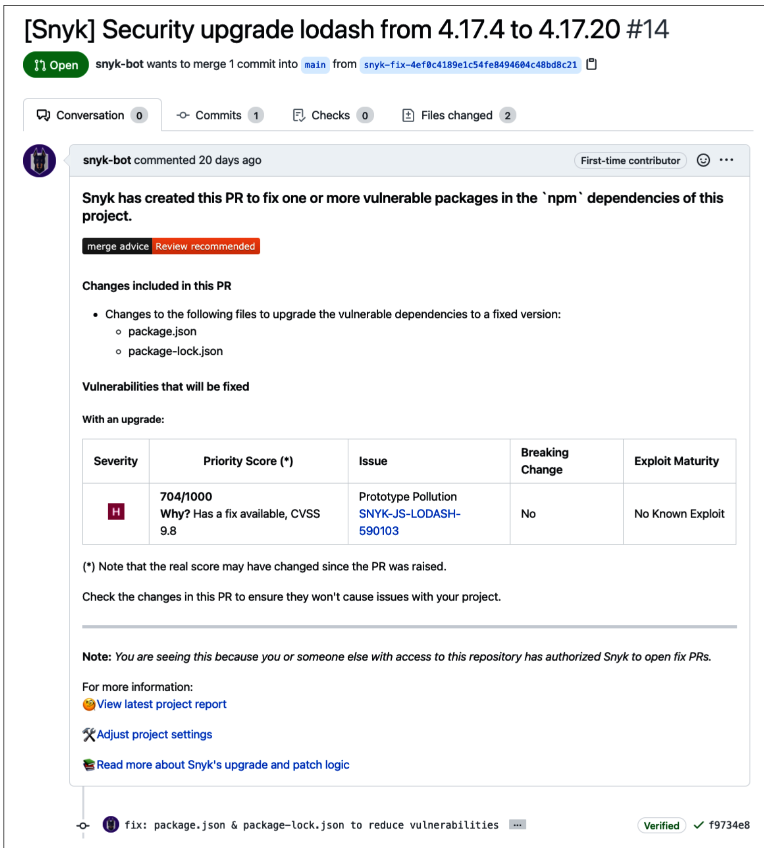

Figure 2-4. Example of Snyk fix pull request

These solutions can be very valuable in reducing risk, but they are not developer-friendly tools. They modify the originally coded functionality (typically after testing) and carry a real risk of breaking legitimate user actions. More important, they take control away from the developer to prevent or fix such functionality breakage. Instead of helping developers secure their apps, they convey a message that the apps are bound to be broken and aim to patch them after the fact.

Dev-first security solutions should simplify remediation but aim to do so as part of the developer's job, instead of taking over the developer's responsibilities.

Now that we understand what dev-first security means, let's discuss how it relates to two other common terms in this field: shift left and DevSecOps.

Shift Left Is Not Enough

The term *shift left* has been used by the AppSec industry for decades. It originates from a waterfall development process visual like the one in Figure 2-5, depicting a left-to-right release process, starting with design and coding and proceeding through building and testing to deployment and operation.

In this flow, security audits were typically done only during deployment (or in production). Security findings required going all the way back to the start, making them costly and time consuming to fix. The call to shift left advocated moving security controls earlier so that issues could be found and remediated sooner and thus with lower costs. This term is not reserved for security and is often touted in other aspects of software quality.

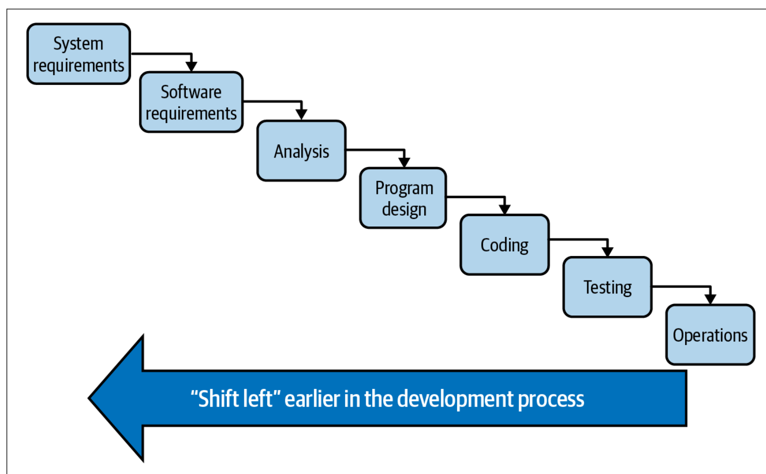

Figure 2-5. Shifting left in a waterfall development process

In the DevOps era, shifting left isn't quite as clear. The core concept behind it is as valid as ever: the shorter the gap between writing a bug and finding it, the cheaper it is to fix it. However, two key parts are missing.

First, there is no "left" in a continuous process, which is rightfully depicted as an infinite process. DevOps accepts that certain bugs

will only be found in production and is willing to sacrifice some level of verification in favor of a faster delivery cycle. It relies on methodologies like observability to help find such issues post-deployment and doesn't necessarily see that as inferior to earlier detection. In other words, it's often better to find an issue shortly after deployment than to add a costly and slow security test to continuous integration/continuous delivery (CI/CD) pipelines, even if it's "further left."[2]

Second, shift left doesn't reflect the change in ownership and drive for independent teams. The truly important change isn't whether you shift the technical testing left, but rather, whether you shift the ownership of such testing to the development team. Pipeline tests that require security teams to review their results, due to false positives or required expertise, can be more harmful than post hoc audits. Each dev team should be equipped and empowered to decide what the best place and time is to run the tests, adapting it to their workflows and skills.

If you have to pick a direction, you should focus less on shifting left and more on going top to bottom. This means replacing a controlling, dictatorial security practice with an empowering one, as I mentioned earlier.

DevSecOps Versus Dev-First Security

The other common term used to describe the transformation required in the security industry is *DevSecOps*.

In its broadest sense, DevSecOps means embedding security into DevOps practices. It's a term trying to work itself out of a job, reaching a state where security should just be a natural part of DevOps, not something that needs to be called out separately. In the meantime, it's used to represent the required change and allow security people and programs to identify themselves with it.

DevSecOps is a very broad term, and those using it typically mean one of three things:

2 CI, often called a "build," processes source code and packages it. CD ships this package to the next step, such as deploying it on cloud. Tests are often incorporated into both steps to find flaws and "break" the process if it doesn't meet certain conditions.

- Adapting security to DevOps *technologies*, such as containers, IaC, or the elastic compute cloud itself
- Adapting security to DevOps *practices*, such as continuous deployment, elastic scaling, or observability
- Adapting security to the DevOps *shared ownership mindset*, driving cultural change toward seeing security as everybody's responsibility

All three are required changes, and their order evolves from tactical to strategic. In the short term, the need to secure technologies that DevOps teams embrace is urgent and top of mind. In the long term, you have to adapt your security culture and change security practices to enable security to keep up with the rest of the business.

DevSecOps and dev-first security have similar aspirations of adapting security to the DevOps world. They have a lot in common and thus can often be used interchangeably, but they have a different starting point: one in dev, and the other in ops.

DevSecOps typically focuses on ops changes. It rotates around the convergence of SecOps and DevOps practices and the post-deployment world, covering topics such as managing cloud infrastructure, runtime observability, and incident response processes. Like modern ops teams before them, who renamed themselves DevOps teams to signal their different approach, we see modern SecOps teams calling themselves DevSecOps teams.

In contrast, dev-first security revolves more around developers and their work. It focuses on content in repositories (repos), code editing, and review processes, pipelines, and more, embedding security controls in them. It deals primarily with the work to get secure applications to production, whereas DevSecOps gives more attention to post-deployment work.

In addition, dev-first security is never used to describe a team, only an approach. Tools and practices may describe themselves as dev-first, but I've yet to encounter a dev-first security team. That said, the need to signal your different approach is felt in AppSec too, and thus many forward-thinking AppSec teams have renamed themselves product security teams, reflecting their broader scope and dev-first security thinking.

The line between DevSecOps and dev-first security is blurry. Many DevOps professionals identify as developers, and developers have significant post-deployment responsibilities. In this book, I focus on dev-first security, but I find both terms and movements valuable.

Conclusion

Dev-first security is a new approach to security. It biases in favor of adoption by those who can fix the problem versus those who manage it. However, a security program that cares *only* about the needs of developers is not good enough either. A good dev-first security program has to help security teams govern successfully and keep the organization secure.

It requires two teams working together (see Figure 2-6):

- An empowered development team, equipped with the right tools and mandate to secure what they build
- A supportive security team, focused on making security easy for developers and providing the security expertise and governance required to keep the company safe

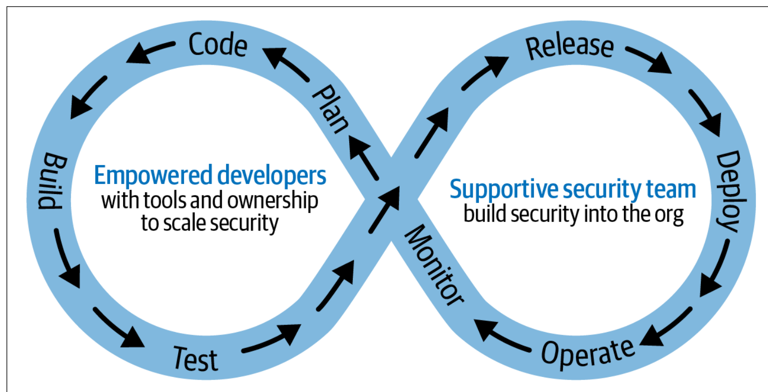

Figure 2-6. A successful dev-first security program requires collaboration between dev and security

CHAPTER 3

Securing the Entire Cloud Native App

Alongside DevOps, which drove the need for the dev-first security approach just discussed, we've seen the evolution of the cloud and the era of cloud native applications. As I mentioned in Chapter 1, cloud native apps have a broader scope than their predecessors, growing to include more elements of the underlying stack.

This change in application scope requires a change in the scope of application security too. This chapter discusses a new and expanded scope for AppSec called Cloud Native Application Security (CNAS).

Before we dig into the details, let's take a moment to understand this transition and what the new scope holds.

From IT Security to Cloud Security

Before the cloud, applications were typically made up of code and libraries and ran on a large central stack managed by the IT department. If a developer wanted a server to run the app or needed a port opened, they opened a ticket with their justification, and IT processed the request. Even after the resource was supplied, the responsibility for ongoing patching of this server or monitoring the opened ports sat with IT, who would reach back to dev only if necessary.

Most of the IT/ops and security industry focused on this reality. IT teams are, by and large, quite security minded and would balance an incoming functional request with their responsibility to keep the data center secure. To serve these teams' needs, a rich set of solutions came to be, helping them manage and secure fleets of servers and infrastructure. These included config management databases (CMDB), patch management, vulnerability management tools, and more. These solutions were designed for IT teams, tuning—as discussed in Chapter 2—for their needs, context, and surrounding tools.

The cloud introduced a very different reality. Instead of needing to ask IT for a server, a developer can just spin one up through a simple API or a few web console clicks. Instead of asking for access to an IT-managed database, the same developer can pick an open source container or use a cloud service. Opening ports, changing permissions, and many other aspects of infrastructure administration suddenly became available to developers, on demand, without delay. Removing the delay unlocked a burst of productivity and innovation and was a key force in the digital transformation tidal wave, but brought along some scary security implications.

Although developers could easily spin up infrastructure, they weren't proficient or well equipped to manage and secure it over time. For many enterprises, the result was a sprawling, ill-managed, and often vulnerable cloud surface; an alarming number of unpatched containers; and a wave of breaches due to poor security hygiene. To help regain control, a new class of container security and cloud security posture management (CSPM) tools came to be, using cloud APIs and other techniques to track what's deployed and flag potential holes, as Figure 3-1 shows.

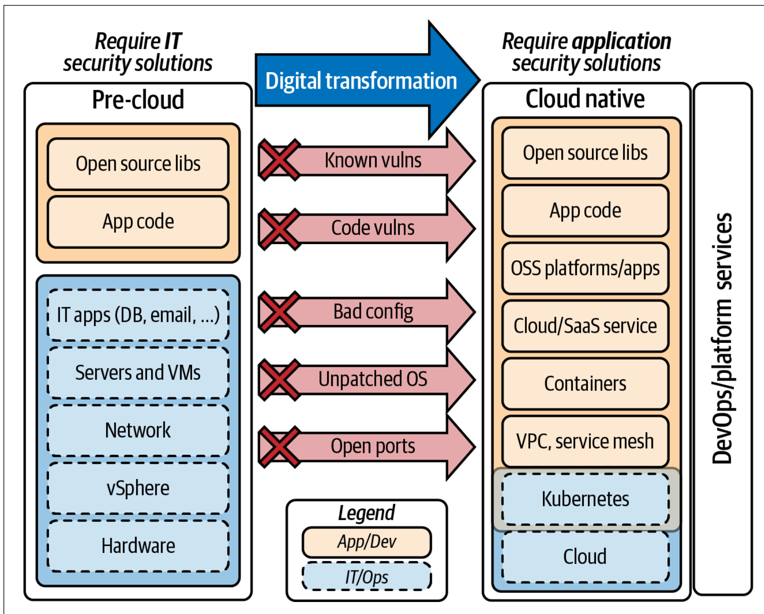

Figure 3-1. Risks that required IT security solutions before cloud now require an application security solution

From Cloud Security to Cloud Native Application Security

Alongside CSPM tools helping to get IT/ops back on their feet, developers also evolved their use of the cloud. Instead of ad hoc actions to set up a cloud environment, tools such as Docker, Terraform, and others were created to unlock IaC, a means to describe infrastructure as though it's just part of the app. These powerful tools up-leveled the previous IT/ops processes, introducing far better repeatability, local testing, version control, and other collaboration capabilities previously reserved for custom code development alone.

From a security perspective, these tools introduce a new reality. CSPM is useful, but those solutions find the problem *after the fact*, meaning the flaw was already deployed, limiting them to visibility and fast response. IaC, on the other hand, provides an opportunity to find security mistakes *before* they are deployed, by testing for flaws as part of the CI/CD pipeline, driving far better efficiency.

At first glance, security controls for containers or IaC may seem similar to their ticket-based IT security predecessors. They deal with the same risks, such as an unpatched server or an open port, and offer an opportunity to weigh a request for functionality against the security risks involved. However, in practice, the *user* for the two solutions is entirely different.

Containers, IaC, and their brethren are managed by developer tools. They reside as files in a source code repository, are edited through Git-based collaboration, and get applied by running a build. Furthermore, they are often managed in the *same* repository as the rest of the app, and the logic of the app's custom code is often intertwined with the underlying containers and infrastructure (infra). Last, they are increasingly managed by the same developers building the rest of the app.

Securing containers and IaC requires an application security solution, not an IT security solution. Just like the rest of their AppSec peers, these solutions must be dev-first and fit elegantly in the developer's workflow and ecosystem of tools. Over time, as the lines blur between the app's custom code and underlying cloud infra, we must similarly combine securing those parts in one holistic view.

This combined approach of securing all parts of a cloud native app is what we call CNAS, an expanded scope of the historic application security space.

Securing operating systems and infrastructure as AppSec (instead of IT security) requires some rethinking. The next sections shed light on some key differences in the two biggest new players in the AppSec landscape: containers and IaC.

Container Application Security

Container images look an awful lot like VM images, but they are lighter in file and memory footprint and thus faster and cheaper to scale. Just like VMs, they hold within them a filesystem and, notably, an operating system, on top of which applications are installed and run. They offer similar operational advantages, including running multiple software machines on a single hardware device, just with a different performance profile and interface. Last, from a security perspective, they need to be secured in a similar fashion, ranging from keeping that operating system patched to monitoring a run-

ning container to ensure that it wasn't compromised, just like you would a VM.

As a result, security teams asked to secure containers, typically after business units already started adopting them, default to applying their VM security practices to the task. More specifically, since containers are used mostly in cloud environments, security teams tend to apply the same security controls used to secure *cloud* VMs to containers.

This default motion is applied both for runtime container security and for securing container images, but it's only effective for the former.

Runtime Security for Containers Versus Cloud VMs

For the purpose of securing containers in runtime and identifying and reacting to attacker activity, treating containers like cloud VMs is a pretty good starting point. The two entities are similar in most of the areas that matter.

- Containers and cloud VMs are both ephemeral (short-lived) and scale elastically, requiring monitoring software that can identify machines wisely and cluster them well.
- Attacks on cloud VMs and containerized systems mostly target the operating system and apps they hold, with very few targeting specific VM or container vulnerabilities, meaning attack or compromise indicators are similar for both.
- Both containers and cloud VMs are API driven and designed to be ephemeral, allowing automated recovery and resetting of machines.

That said, containers do require cloud VM security systems to evolve to cope with a new level of scale and speed. Most container clusters run an order of magnitude more VMs than cloud VM setups, are updated on a far more frequent basis, and run more versions in parallel. Containers tend to be more immutable than cloud VMs, scale up and down much faster, and are more likely to be deployed across multiple clouds.

Thus, runtime container security solutions need to be *better* than cloud VM ones, but along the same path. A great runtime container security solution can easily handle securing cloud VMs too; it just

needs to add support for a different virtualization and orchestration platform. In other words, runtime container security solutions are an *evolution* of their cloud VM predecessors, whereas securing container images requires a *revolution*.

Securing Container Images Versus Cloud VMs

If their operations seemed similar, building containers couldn't be more different from building cloud VMs.

Cloud VMs are built via Secure Shell (SSH) and IT tools. The most common pattern is to create golden images by IT manually or by using tools like Puppet and Chef and store them as reusable VMs (e.g., Amazon Machine Images [AMI]). Apps are installed on top of those VMs, either creating their own images ahead of launch or installing the app part at boot time. Over time, IT maintains those golden images, patching them and asking downstream users to update too.

This process has pros and cons but has one big fundamental flaw: it's separate from the workflows for building the app. This results in myriad common problems, such as the following:

- Golden images are modified without testing the apps that run on them, but these underlying changes can cause those apps to break unexpectedly.
- Golden images change on a schedule independent of the app, introducing risk and noise at potentially sensitive times in the app's own rollout schedule.
- App-specific images have no inherent traceability to the golden image they were built from, relying on error-prone manual tracking to know when they need to be patched due to golden image security updates.
- Golden images are often updated inline, with little or no versioning or storage of past versions, making it hard to track patching or rollback.

Containers, especially after the introduction of Docker, are designed to address this gap. Containers are declared as source code (usually using a Dockerfile), benefiting from Git versioning and collaboration. Their layers and, notably, base image are clearly declared, as Figure 3-2 shows, offering traceability to external or centrally

managed golden images. They are built using a standard CI/CD process and stored in a versioned registry, providing the tracking and rollback support needed for proper continuous deployment.

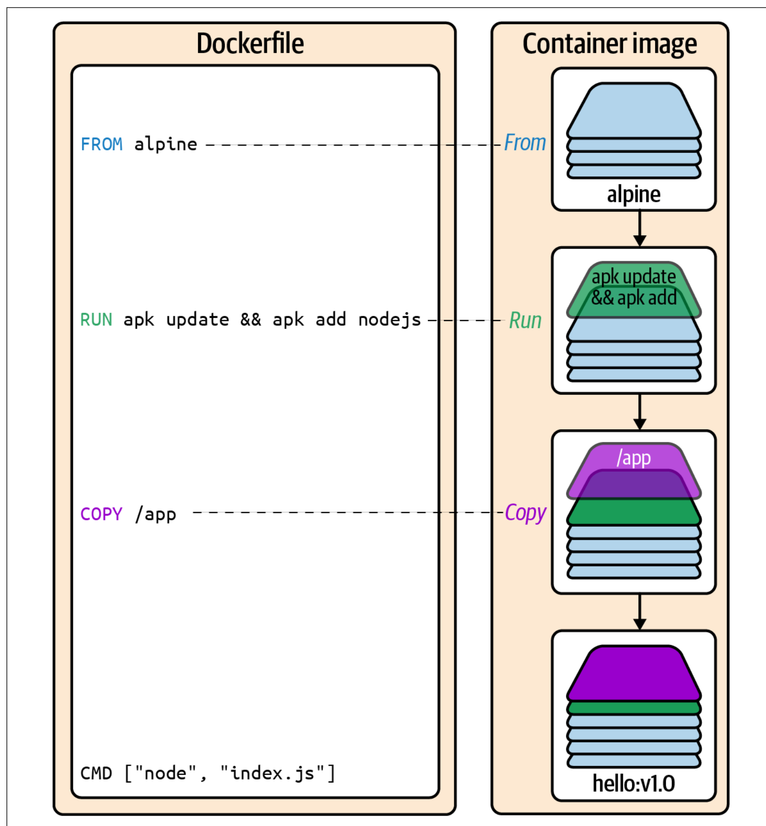

Figure 3-2. Visual of a Dockerfile and the layered container image built from it

Overall, containers are built as applications, not infrastructure, and securing them requires an application security solution. This means that instead of evolving your VM patching flows to include containers, you need to evolve your AppSec practices to do so. You need to tackle the same risk—having an unpatched or misconfigured server—through an entirely different lens.

Container Security Ownership

As a step further, it's important to realize that containers aren't just built the same way apps are built; they're built as *part* of an app. The Dockerfile defining a container usually sits in the same repository as the app that runs on it and is edited by the same developer changing the code in the file next door. The same build compiles both the app's code and the container it will be stored in, and the combined result is stored together in the registry. Containers are a core part of the application being built and shipped.

This creates great alignment between apps and their underlying OS. When updating a container, the ensuing build process will ensure that the app running on it will keep functioning as it should. When a problem does occur, the application can be rolled back to a stable state, including its underlying virtual servers. But it also creates a coupling; containers have to be built and secured by the same people building and securing the application's code.

This is a change in ownership that shouldn't be taken lightly. We're asking developers to take on a new responsibility, one they weren't expected to do a decade ago. We need developers to see it as part of their job to pick a secure operating system, minimize its content and user permissions, and patch it on a regular basis. We need to educate dev teams on how they can take on these new risks and equip them with the tools and mandate to do so successfully.

Last, moving container security into AppSec requires rethinking our priorities. Should your dev team prioritize avoiding and fixing vulnerabilities in their own custom code, or patching the container they use to address known vulnerabilities? Historically, developers were heavily focused on the code they wrote, but not patching your server is a far more common way to be breached. In the past, these two risks were part of different backlogs; now they're on the same list.

Securing Containers as Apps

Rethinking container security through a dev-first AppSec lens is a big endeavor, but there are a few key areas to focus on to get the change started. These represent necessary changes, but also opportunities to use this transition to be more secure with less disruption.

First, *test containers early, even before CI/CD kicks in*. The fact that Dockerfile is a source code file means it can be inspected when

edited in the IDE to flag security mistakes. It can be reviewed automatically in pull-requests before it's even built. Such early detection can flag known vulnerabilities or bad base images before they are merged, saving time and effort to fix them and reducing the chance of a disruptive broken build.

Second, *focus on fixing flaws, not just finding them*. As mentioned earlier, although an auditor's job may be to find issues, a developer's job is to fix them. Said differently, it's easier for developers to fix problems than for auditors to, and we should make the most of this. Make sure your security tools invest in making remediation easy, whether through clear guidance, automated-fix pull-requests that modify Dockerfiles, or automated build triggers to drive patching. This will clearly help you fix more holes but will also improve developer adoption of the tools.

Third, *invest in base image management and relating vulnerabilities to them*. When you audit an image as a filesystem and find a vulnerable artifact, you don't care which layer it arrived from. If you're auditing the same image as source code, the origin of each vulnerability becomes key. A vulnerability in a base image requires updating the base image, or perhaps simply rebuilding to get the new patch. A vulnerable line in the Dockerfile requires editing your source code, an entirely different action. Make sure you clearly separate the two and invest in managing those base images, because most vulnerabilities arrive through them.

Last, *introduce container security gates into your CI/CD*. Although detecting and fixing locally or in Git is ideal, security flaws can still slip past those opt-in steps. Take advantage of the automation container platforms that offer to introduce guardrails into your pipeline, preventing artifacts that introduce severe security issues from being deployed to production. Just keep in mind that breaking the build is a disruptive action, so be careful when picking the security thresholds that require it.

IaC Application Security

Hot on the heels of container adoption is the growing use of IaC.

IaC came to be in two waves. The first wave was led by tools such as Puppet, Chef, and Ansible and introduced much-needed automation to the world of VMs. It replaced sysadmins SSH-ing into a

remote machine and setting it up each time with consistent and automated scripts, and unlocked the ability to patch machines regularly and apply security configuration policies at scale. I'll refer to this wave of solutions as IT automation tools.

The second wave came with cloud and was led primarily by Terraform, which excels in configuring cloud services. Although its predecessors were used primarily by IT to automate their work, Terraform became the tool of choice for developers and DevOps teams to tune the infrastructure to the application that ran on it.

Puppet and others have since adapted to this new world, and new IaC players like Pulumi came to be. In addition, alongside Terraform, platform-specific IaC solutions were created, such as Kubernetes Helm charts, AWS CloudFormation, and Azure's ARM. Today, most IaC solutions are cloud and application oriented, and IaC is the clear best practice for defining application infrastructure in the cloud.

As with containers, this new IaC wave introduced a bigger change than new declarative languages or cloud-related features. IaC is now managed as part of a continuous software pipeline, with source files in Git repos, build pipelines applying changes, and even increasingly using standard programming languages. In other words, IaC files are managed as applications, not as IT—and need to be secured accordingly.

Securing Infrastructure as Applications

> *"We need to shift from securing the infrastructure to securing the code that creates it."*

Securing infrastructure before IaC involved a lot of manual work, ad hoc scripts, and spreadsheets to track what was installed where and keep the perimeter secure. With the advent of IT automation tools, discovery of the state of your infrastructure became feasible, as did applying security policies at scale.

However, though rules could be applied more easily, defining them remained difficult. Applications require many network access paths and system permissions to operate, and those needs change frequently. Communicating those needs through tickets led to slow execution, complex tracking, and a lot of frustration. At the end of the day, infrastructure was often overly permissive, which increased

risk, or was too strict, which caused applications to break—a bad outcome either way.

IaC offers an opportunity for much better alignment, improving security while reducing risk of breakage. By defining infrastructure needs as a natural part of the application, we can ensure that they are "just right": not too open, not too closed. To achieve this, we need to change our perspective and shift from securing the infrastructure to securing the code that creates it.

This leads us again to application security. We need to assess IaC files as source code and find security vulnerabilities in them. These flaws should be surfaced to the developers writing them, helping them understand and fix them as they code. When a weakness is found in production, it should be solved by going back to the code and fixing it, not by manually modifying the infrastructure previously deployed (though that's a legit temporary patching action).

As with containers, IaC security isn't an evolution of IT automation; it's an entirely new perspective. To secure infrastructure as code, we need to roll the application forward, anticipating how it would manifest in production, as opposed to rolling infrastructure backward. Over time, the source of truth should be your source code, not the assets deployed, and your security will depend on how well you secure that code.

Helping Developers Secure Infrastructure

Securing infrastructure is a complicated domain. Although IaC technology resembles software development, most developers don't know which configurations are secure and which ones aren't. Few developers will be familiar with Kubernetes's `securityContext` attribute or know that CloudTrail logging needs to be disabled, and the growing complexity of cloud platforms implies that this gap will never truly go away. To address this challenge, you need to invest in education and automated inspection. See Figure 3-3 for some IaC rules that Snyk applies.

For education, focus more on logical configuration than on technical implementation. For example, your dev teams should understand the importance of minimizing the permissions a service is given, reducing the scope of who can access it, and securing the data coming into it and going out of it. They should learn to appreciate how observability helps security and how to balance data retention

for forensic purposes with a privacy-driven desire not to retain personal information. These are complicated topics that will take a while for teams to truly master, but they are an important investment in upskilling your team.

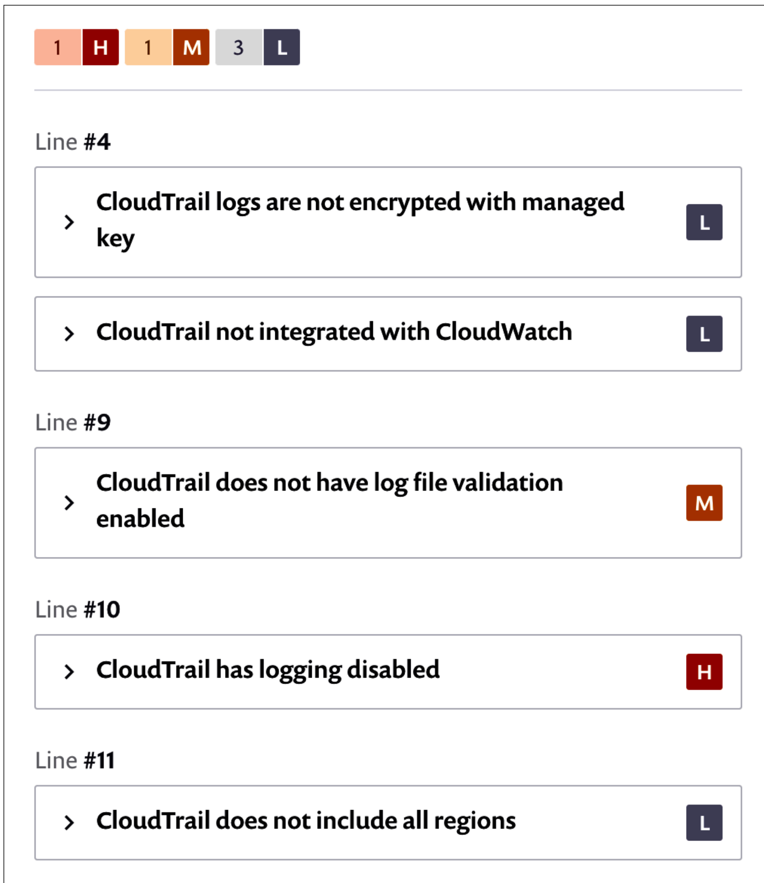

Figure 3-3. A few examples of many IaC rules that Snyk applies

For the technical implementation, however, tools should be your key solution. IaC scanners such as Snyk IaC, tfsec, and HashiCorp Sentinel can map your security requirements to every platform's implementation details and inspect your source code as it changes. Although they can't replace your policy decisions, such tools also embed industry best practices that can make it easy to get started and kick-start developer education.

Beyond this high-level advice, here are a few quick tips on how to embrace an AppSec approach to securing infrastructure, which at this point in the book should start sounding quite familiar:

- *Find issues early, before the infrastructure exists.* Analyze source code files in IDEs and Git to find problems and guide developers to fix them.
- *Invest in fixing IaC source code.* When flagging problems, help developers fix them by raising clear potential solutions, automatically creating fix pull-requests and more.
- *Automate IaC testing.* Software needs to be tested to avoid regressions,[1] and IaC is no different. Invest in unit tests, which in turn will give dev teams confidence to reduce permissions with less risk of breakage.

Conclusion

Embracing the cloud and cloud native development is about far more than technology. It's an enabler for independent teams to run faster, adapting to market demands faster and driving superior business outcomes. To achieve this, application teams are given ownership and control over more layers in the stack, and we need to make sure they can also keep those layers secure.

CNAS is the perspective we need to make this happen. It requires us to understand that the same people previously tasked solely with securing their code now make decisions around so much more. We discussed containers and infrastructure, but dev teams also need to manage data, service, API gateways, and much more, all areas where decisions can have significant security implications.

Each of these layers requires us to rethink our security controls in a dev-first fashion. As we've seen with containers and IaC, the resulting solution could look very different from its predecessors.

1 A regression occurs when a code addition unintentionally breaks previously working functionality.

CHAPTER 4

Adapting to Dev-First CNAS

Adopting CNAS requires significant changes to the way we secure applications and infrastructure. Making the shift is a journey, one that is different for every organization, and even for different parts of the same org.

Although choosing the right path is your decision to make, patterns and best practices are starting to emerge for getting it right. In this chapter, I suggest a few areas in which to consider breaking the status quo, and how.

Rethinking the Security Org Structure

Organizations are typically split based on areas of responsibility. As you shift your view to see securing parts of your infrastructure as an application security problem, reconsider how you structure the security org. More specifically, consider whether to change the scope of responsibility for the application security team.

In addition, as your security practices become more dev-first and focus on empowerment of developers, your requirements for this AppSec team change. You need more empathy and program management alongside more engineering capacity. You need more builders and fewer breakers.

To help you evaluate your security organization structure, here are the three most common team scopes I see in the AppSec world: Core AppSec, Security Engineering, and the newer Product Security

scope. These should serve as reference points for how to structure your org, not perfect models to adopt.

Core Application Security Team

Let's start from the status quo, keeping the same scope for the application security team. Since this is the default state, most organizations use this team scope—at least as a starting point.

The mandate of the core AppSec team is securing the *custom* application code and business logic as well as the open source libraries being used. They typically own the classic Application Security Testing (AST) suite, including static, dynamic, and interactive AppSec testing (SAST, DAST, and IAST) to find vulnerabilities in custom code, and software composition analysis (SCA) tools to find vulnerable open source libraries. In addition, these teams often develop security education and training and, potentially, vulnerability management or bug bounty. In some cases, they may own runtime application protection too, using RASP or WAF tools.

Core AppSec team members typically need to be subject matter experts in secure coding and have some experience running audits and security code reviews. They need good developer empathy to collaborate with dev, which in turn requires some ability to understand or relate to code but doesn't require full software development credentials.

The main advantage of sticking to a core AppSec team is the tenure it has in the industry. For hiring, it makes it easier to hire professionals who bring with them experience across the entire team domain. For tooling, it's a space where the tools and practices are well documented. And from an organizational perspective, most of the industry will assume an AppSec team looks like this core AppSec team.

Although the scope of a core AppSec team is the status quo, its methodologies have often become more dev-empowering. AppSec teams often assign individuals on the team to be the partner to several dev teams, helping foster better collaboration. Many others run Security Champions programs, helping them get scale and embed more security expertise in the dev team. Although the scope is mostly unchanged, the internal practices of core AppSec teams don't have to be traditional.

Security Engineering/Platform Team

Automating security steps is key in a modern development environment. Fast CI/CD pipelines leave no room for manual review, requiring automated pipeline tests instead. In addition, developers are not security experts; they have less time to devote to security and, thus, need tools that have embedded security expertise and can offload or facilitate security decisions.

Building and operating security tools is no easy feat, especially in large organizations where different dev teams have vastly different requirements. To help boost automation, some organizations created dedicated security engineering teams that focus on building internal tools and integrating external tools, all meant to bolster security.

Security engineering teams are made up of software engineers with a slight bias for security, and operate like a complete DevOps engineering team. They typically build, deploy, and operate the services they build, and use the same methodologies other engineering teams use to run their agile processes and manage product backlog.

If the volume of work isn't big enough to warrant its own team, this same activity is typically embedded in the core AppSec team. However, though teams titled Security Engineering are pretty consistent in their charter, individuals with the (increasingly common) Security Engineer title vary greatly in their responsibilities. Some are software engineers as described, whereas for others the "Engineer" part of the title refers to the security realm instead.

Security engineering teams are a great way to truly level up the amount of automation and are a great parallel team to ops-oriented platform or site reliability engineer (SRE) teams. In fact, in quite a few cases, the platform team's scope has been expanded to include building and operating such security tools. It's also a great way to get software engineers into the security team, helping with the talent shortage problem and with building more dev empathy in the security team.

Product Security/Cloud Native AppSec Team

A more recent addition to the security team patterns is the product security team. These teams have a bigger scope that includes not only the application code itself, but everything to do with the

product. Most notably, two key additions are capturing the full CNAS scope and helping build security features in the product itself.

Full CNAS scope

Growing to include the CNAS scope is a natural result of rethinking certain infrastructure risks as application security. As discussed in Chapter 3, technologies like containers and IaC are driven by the same developers writing custom code and using the same practices and tools. To support this change, AppSec teams need to support those engineers in doing so successfully. Teams that embrace this broader scope often refer to themselves as product security teams.

This expanded CNAS scope means product security teams work across a bigger portion of the software development life cycle. It includes more engagement with production deployments and even operations, leading to overlap with the more ops-focused cloud security team. In practice, cloud native development means that cloud security is affected by both dev and ops teams, and product security teams cover the former.

Note that many core AppSec teams are expanding to embrace the full CNAS scope without officially changing their name and mandate. Choosing and implementing solutions to scan container images for vulnerabilities and audit IaC files is increasingly the AppSec team's domain. Although it's safe to assume product security teams capture this complete scope, such a rename is not strictly necessary, and many AppSec teams have evolved without such a declaration.

Product security features

A bit unrelated to CNAS, but still noteworthy, is the involvement of product security teams with a more user-oriented part of security: security features. As user awareness of the importance of security grows, many products look to build dedicated security features and differentiate through them. Deciding what security capabilities are of value requires a level of security understanding that dev teams may not have, but security teams do. Product security teams often embrace an explicit role here, collaborating with product managers (PMs), who own the complete product functionality and value proposition, more than ever before.

This responsibility plays an important role in the relationship between application and security teams. Security controls are a means to mitigate risk, but being able to present this risk mitigation as a security *feature* means it can help grow revenue instead. Growing revenue is another shared goal for the two teams, and one that is far more visible than risk reduction, making it easier to celebrate success.

The evolution of product security

Product security is a new title and scope and is still being defined. Given its broader scope, it's often a parent title or group, which encompasses the other mentioned teams. In some cloud native organizations, product security is the primary scope of the chief security officers (CSOs), whereas a few others started naming leaders as chief product security officers (CPSOs).

Adrian Ludwig, Atlassian's chief information security officer (CISO), phrased it best, saying, "Product security's goal is to improve the security posture of products and represent customers' security expectations internally to the development team". The title is used by other companies such as Twilio, Deliveroo, and Snyk, and I believe it's the right way to address CNAS.

What About the DevSecOps Team?

You may have noticed I didn't name a DevSecOps team, and that wasn't by chance. Like DevOps, DevSecOps isn't a team; it's a movement, meant to embed security into the core dev and operations work. In my opinion, it shouldn't be the title of a team.

However, just like DevOps teams exist, so do DevSecOps teams—and their mandates vary greatly. At times, they're really a cloud security team, focusing on operations and runtime security. Other times, they're more platform-like, with scopes similar to security engineering teams. Since the title doesn't imply a specific set of responsibilities, the scope of a DevSecOps team isn't something that can really be defined.

However, what all those teams have in common is that they are forward thinking. DevSecOps aims to change how we do security, and DevSecOps teams, wherever their scope, consistently see themselves as change agents. They embrace automation and cloud, favor

engineering security solutions over running audits, and aim to empower dev and ops teams to secure their work on their own.

Rethinking Tooling

Alongside organizational structure changes, CNAS and dev-first require a reevaluation of your toolkit. Given this new perspective, you should reconsider the most important traits you're looking for in a technology solution and how you want tools to be bundled.

There are many ways to reevaluate tools, but I would suggest focusing on three primary areas: dev adoption, platform scope, and governance approach.

Developer Tooling Caliber

If our goal is to get developers to build security into their daily work, we need to make sure we equip them with tools that optimize for that goal. If you buy solutions designed for auditors and ask developers to use them, you're unlikely to be successful.

Developers, unsurprisingly, are used to using *developer tools*. These tools represent an entire industry, which evolved its own best practices around what makes a great developer tool. To help you choose a security solution developers will embrace, you should evaluate these tools against developer tooling best practices and see how they fare.

Here are a few of the common traits of a great developer tool:

Successful self-serve adoption
: Practically all successful dev tools have great self-serve usage, including easy onboarding, intuitive workflows, and great documentation. This is the way developers like to consume tools, and it forces vendors to ensure that their tools relate to developers without a salesperson pushing it. Unless you want to be that salesperson promoting a tool to your developers, look for tools with a proven track record of self-serve adoption by dev.

Seamless integration into workflows

Dev tools, for the most part, look to meet developers where they are instead of asking them to open yet another tool. They integrate elegantly into their IDE, Git, and pipelines and provide value at the right point in time. Integrations are about more than technical hooks; they need to adapt to workflows and best practices, or they will be rejected.

Rich API and automation friendly

Although opinionated integrations are needed to get started, a dev tool must also be a Swiss Army knife. A rich API and automation-friendly client (CLI/software development kit [SDK]) are mandatory, both to tune the tool to every pipeline and to allow the community to build on it. If you can't build on a tool, it's not a great dev tool.

Adoption by open source and community

Developers look to fellow developers to validate a tool's credibility, and open source adoption is the best indicator of that. Seeing open source projects integrate a security tool or open source projects that build on it are great developer community validations. When inspecting a security tool, inspect its adoption amid open source and draw your own conclusions.

These are just a few of many dev tool best practices. They should be added to the security-specific recommendations around dev-first security from Chapter 2. In addition, make sure you involve actual app developers when evaluating any tool, to get a real-life perspective on how well it fits into their surroundings (see Figure 4-1).

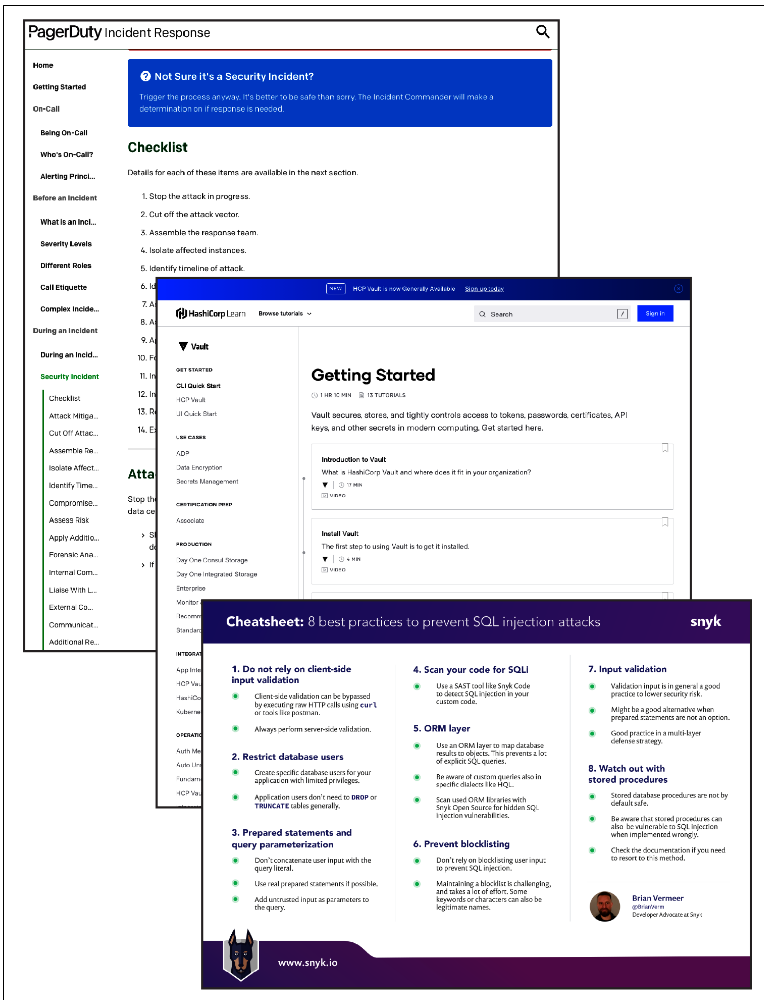

Figure 4-1. Examples of developer-friendly security tools

Platform Scope

AST platforms today are primarily focused on custom code and, perhaps, the open source libraries it consumes. Tool suites are primarily composed of SAST, DAST, and IAST, with SCA recently added to the mix. As you embrace the broader cloud native application scope, reconsider the makeup of the platform.

To start, let's consider what tools benefit from being part of a single platform. They can be divided into a few big buckets:

Shared users
: If different tools are meant for different primary users, there's little need for them to be part of a single platform because they'll be used separately anyway.

Shared workflows
: If multiple tools are used in a similar fashion and integrate in similar points of the user's workflow, you can save the integration time and user time and effort by combining them instead of requiring engagement with multiple separate tools.

Shared prioritization
: If the action items from different tools should be prioritized against one another, sharing a backlog can increase efficiency and results.

Multiplied value
: Finally, the strongest driver for tools to share a platform is when using tools together can enhance each tool.

: This criterion naturally explains why technologies like SAST and SCA fit well in a single platform. Both serve the same developer users, run scans and engage users in the same spots, and share a backlog of security vulnerabilities the same developer will need to prioritize between. In advanced SCA solutions, SAST technology can indicate whether your custom code invokes the vulnerable code in a library, providing greater accuracy—hence multiplied value.

The same logic applies when you consider adding container and IaC security to the same platform of SAST and SCA:

Shared users
: Containers and IaC are now secured by developers, just like SAST and SCA, who would rather not have multiple disparate products to learn and engage with.

Shared workflows
: Securing containers and IaC requires integration across the life cycle, such as IDE, Git, and build integrations, just like SAST and SCA.

Shared prioritization
: The same developer needs to spend cycles fixing a container or IaC vulnerability or one in the code or libraries—all one backlog for securing the app.

Multiplied value
: There are many ways in which securing these components combine. Scanning containers requires exploring the app inside, knowing the infra config helps prioritize code and library vulnerabilities, and understanding the flow across components helps mitigate issues; this is precisely what you do when triaging a vulnerability.

This logic continues to hold even as the CNAS scope expands, and I encourage you to think about your CNAS tooling as one platform. An easy guideline is that anything sitting in a Git repository probably relates to the files around it and follows the same development workflows. As a result, a CNAS platform should help secure everything in the repo—the entire cloud native app.

DAST and IAST are slightly awkward participants in such a platform. Although they deal with securing the app, they typically require different workflows; it's hard to fit them into a build pipeline or IDE scans due to the time they take to run. In my opinion, this is a technical problem, not a logical one, which will hopefully be solved once IAST and DAST solutions evolve to become more pipeline friendly.

Governance and Empowerment Approach

Embracing a dev-first security approach requires a different type of collaboration. Instead of having tools that focus on broadcasting and enforcing the top-down mandate, we need tools that help us build secure applications *collaboratively*.

This may sound obvious, but in practice it's a big departure from the way many security tools work. Let's dig into a few concrete examples to get a taste of what this means.

First, developers need to be empowered to balance business needs with security risk. This means, for example, allowing them to decide not to fix a discovered vulnerability and continue deploying it to production. This is a scary proposition to some, but it's the only way to enable independent teams. Note that ignoring a vulnerability

should still require a provided (and auditable) reason, and certain constraints should be hard lines (typically for compliance reasons), but as a default stance, the decision should be left to the developer.

Second, developers need to be able to manage their security risk, and that requires seeing all the vulnerabilities in their projects. This needs to include not only the list of vulnerabilities, but also all the information needed to prioritize well, such as exploitability and asset mapping. Being transparent with such information may introduce some risk, but it's the only way to scale security; to empower developers, you have to trust them with this sensitive data.

Third, security teams should invest in tracking and driving adoption of security controls, even more than their outputs. Dev teams should manage their vulnerability backlogs, but the security team needs to help them do so successfully. Dev-first security governance means tracking which controls are adopted and how well their output is handled—and working with dev teams to improve those stats as opposed to highlighting the vulnerabilities they should fix.

These are just a few examples of what to consider when assessing your governance tools and techniques. The key is to embrace a platform mentality; its goal is to help dev teams succeed in building secure software, not to track the security flaws themselves.

Rethinking Priorities

Last but not least, CNAS requires rethinking application security priorities. If developers need to secure the entire cloud native application, where do you want them to focus the most?

Historically, IT security controls dominated far bigger budgets and commanded more CISO attention than application security ones did. There may be historical reasons for this imbalance, but at the end of the day, it represents a CISO's view on how best to use their dollars to reduce the risk to the organization.

In other words, CISOs believe the risk of a breach due to an unpatched server or misconfigured infrastructure is greater than that of a custom vulnerability in your code. This perception has a fair bit of data to support it, showing historical breaches and their causes. In addition, it's easy to understand; it's far easier for an attacker to run known exploits at scale and look for an open door

than to reverse engineer an application and find custom flaws to let them through.

The cloud doesn't diminish this equation; in fact, it strengthens it. Adoption of cloud means more infrastructure and more servers are used, they tend to be more publicly accessible, and they are defined by teams that are less familiar with managing them (developers) and equipped with less-mature tools.

However, whereas the previous balance was between IT security budgets and AppSec budgets, now it's all in the AppSec world. When building cloud native applications, the same developers need to secure their custom code, manage vulnerabilities in their open source usage, and make the servers and infrastructure resilient. The same AppSec team needs to help them do so as well as split the same budget across them.

So we're back to the opening question: how do you want your precious developer time and limited CNAS budget distributed?

Left to their own devices, application security budgets and developer attention focus on securing custom code. AppSec maturity models, industry best practices, and the personal experience of individuals on the team are all anchored in the pre-cloud world, where custom code was the majority of what developers had to secure. Buying tools like SAST and DAST is often the first thing on a new AppSec leader's list and is rarely challenged.

However, given the CNAS scope, is that still the best use of your time and money? The risk of a breach due to a known vulnerability or a misconfiguration is still higher than that of a custom code flaw, even if developers are the ones configuring it. If you agree, shouldn't you tell your developers and AppSec team to focus first and foremost on securing those layers, and only then approach their custom code?

This isn't an easy question to answer. I won't assume to know what your priorities should be, because those change from org to org. However, I strongly encourage you to have this conversation internally and reconsider your priorities, given the broader scope of CNAS.

Conclusion

Changing how you approach application security isn't just a thought exercise. It has very real downstream implications, including people's careers, budget allocations, and a reassessment of the best way to keep the organization secure. It requires shaking off some muscle memory of how you've done things in the past and instead going back to first principles when choosing solutions, which takes precious time and energy.

The good news is that you don't have to do it all at once. The changes I outlined in this chapter are related to one another but can be applied at your own pace. I suggest you pick the ones you relate to the most, or opt for other changes you think are more important, and get the changes going. Step by step, you'll adapt your security function to the cloud native reality.

Summary

Digital transformation is a force to be reckoned with. Across every vertical, businesses strive to become technology companies and increasingly differentiate on how well they live up to that description.

The cloud and DevOps play a massive role in this transformation and overhaul how we develop and operate software. Software has never been easier to create, has never been updated as frequently as it is today, and has never innovated to adapt to customer needs so quickly.

These changes are all made possible by unlocking the power of independent teams to run fast. Agile drove these teams to be customer centric, iterate on ideas rapidly, and own the quality of their code. DevOps lets developers operate what they build, combining code and infrastructure to better deliver on customer needs. And security, as an industry, has been left behind.

In the face of such change, security has no choice but to adapt. Businesses must and will continue to strive for speed, and independent teams are the only way to achieve this. The way we secure applications has to transform, making it part of the daily work of these independent development teams. Security teams need to focus, first and foremost, on helping these teams embrace security. Security needs to become dev-first.

As we shift the security industry to the dev-first approach, we need to catch up to another change that passed us by. The cloud, and with it cloud native application development, has moved infrastructure into the hands of developers. Under the same mantle of unlocking speed, dev teams create, manage, and monitor their application

infrastructure, removing obstacles to shipping value. We need to make sure they secure these new responsibility domains well, and expand application security to encompass a broader CNAS scope.

I hope this book helped you understand the value of embracing a dev-first CNAS approach and equipped you with some techniques for how to make it happen. As we work hard to match security transformation with digital transformation, it's worth remembering there's another reward if we get it right.

DevOps has demonstrated that businesses that employ it well are rewarded with significant business success. Companies that unlocked the ability of teams to run fast proved themselves to be resilient and available and are beating their competition. And ops teams have gone from being a cost center to a valued business competency.

Security has the opportunity similarly to help the organization thrive. By helping dev teams build secure software without slowing down, you can not only reduce risk but also grow the top line. You can help your business respond to customer needs faster while differentiating on being more secure and trustworthy. This could similarly turn security from a cost center to a true business partner, driving the company's success—which is an even bigger security transformation.

About the Author

Guy Podjarny is the founder and president of Snyk, building developer-first security solutions. Guy was previously CTO at Akamai following its acquisition of his previous start-up, Blaze, and managed the first application security testing solution, AppScan.

Guy is a frequent public speaker, writer, and author. He wrote O'Reilly's *Serverless Security*, *Securing Open Source Libraries*, *High Performance Images*, and *Responsive and Fast*.