Brought to you by:

WIZ

Kubernetes Security

for dummies®
A Wiley Brand

Operate safely in a cloud-native world

Master the art of Kubernetes security

Implement a container security program

Wiz Special Edition

Steve Kaelble

Hey readers,
Welcome to the world of Kubernetes!

I'm Ami Luttwak, Co-Founder and CTO at Wiz, and I want to share this guide, which aims at simplifying a complex technology.

Since we kicked off in 2020, our crew at Wiz has been all about making sure companies of any size, using any cloud service, can keep their cloud safe. Our mission is simple: make cloud security accessible and effective for everyone. This book lines up with what we're aiming for, giving you some practical insights to secure Kubernetes environments.

K8 is an open-source platform used to orchestrate cloud-native applications. Securing Kubernetes environments is paramount due to the inherent complexity of the platform and the risk of misconfigurations, which can lead to vulnerabilities and security breaches. Robust access control, identity management, and network segmentation are crucial for preventing unauthorized access and isolating workloads.

Securing APIs and resources, implementing container isolation, and ensuring regular updates are essential measures to protect against potential exploits. Additionally, effective monitoring and incident response plans are vital for detecting and mitigating security threats in real-time, ensuring the integrity, confidentiality, and availability of applications and data within the Kubernetes cluster. We will try to cover all of those core principles and more all in this book.

We hope you find this helpful.

We are always happy to chat more about cloud and K8 security, scan our QR code to get in touch!

Cheers,
Ami
Co-Founder and CTO, Wiz.io

Kubernetes Security

Wiz Special Edition

by Steve Kaelble

for dummies®
A Wiley Brand

Kubernetes Security For Dummies®, Wiz Special Edition

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

ISBN 978-1-394-24587-1 (pbk); ISBN 978-1-394-24588-8 (ebk)

Publisher's Acknowledgments

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

Project Manager: Jennifer Bingham

Acquisitions Editor: Traci Martin

Editorial Manager: Rev Mengle

Client Account Manager:
Cynthia Tweed.

Content Refinement Specialist:
Tamilmani Varadharaj

Table of Contents

Introduction

Cloud-native applications may not be brand-new, but they're really coming into their own, according to the Cloud Native Computing Foundation. The organization says that they are, in fact, the "new normal," and why wouldn't they be? They're pretty much unsurpassed in terms of flexibility, scalability, and agility.

That said, like most good things in life, they're not risk-free. The containers they employ have vulnerabilities, the cloud where much of the action happens has lots of complexities and risks involving privileges, and the Kubernetes platform that makes it all work needs dedicated hardening.

If your organization is among the many banking on cloud-native applications running in a Kubernetes environment, you need to put security front and center. That means adopting security best practices, acquiring the best tools, and giving security a seat at the table throughout the lifecycle. Security shouldn't be relegated to its own lonely silo.

Kubernetes security starts with a strong foundation, secure infrastructure, and a well-protected operating system. It recognizes that multitenancy is the way of the world, ensuring that users and namespaces coexist harmoniously.

Your organization's approach to Kubernetes security must identify issues early, scan images and resources throughout the lifecycle, and keep runtime safe. It must hardwire controls and facilitate whatever compliance your industry demands.

Foolish Assumptions

In writing this book, we assume that your work is heavily involved in making innovative cloud-native applications happen.

» You may have a role in DevOps or DevSecOps, perhaps site reliability engineering, or maybe you're an information security leader.

- You know cloud-native applications are essential for your organization's future and that they have inherent risks.
- You're determined to rely on Kubernetes but want to sleep well at night knowing that security is under control.

Icons Used in This Book

Check the margins for guideposts spotlighting what you're reading. They're icons, and this is what they mean:

REMEMBER

Don't worry if your highlighter has run dry, because this icon underscores the most important points.

TIP

Looking for advice you can act upon? Look no further than the paragraph next to this icon.

WARNING

If you're exploring Kubernetes security, you know life is dangerous. This icon identifies an area of potential concern.

Beyond the Book

This isn't a long book, and we admit that there's more to know about these topics. For a deeper dive into Kubernetes security, here are some places to look:

Wiz: `www.wiz.io/solutions/container-and-kubernetes-security`

Kubernetes: `kubernetes.io`

Cloud Native Computing Foundation: `www.cncf.io`

IN THIS CHAPTER

» **Getting a handle on the cloud-native environment**

» **Building and running cloud-native apps**

» **Orchestrating with Kubernetes**

Chapter 1 Living in a Cloud-Native World

This chapter explores how cloud-native applications help businesses meet the need for speed and agility in a competitive environment. It explores how cloud-native app development differs from the ways of the past, how microservices and containers fit into the picture, and how Kubernetes helps make it all work.

Understanding the Cloud-Native Environment

Back in the old days, it was not a compliment to talk about people having their heads in the clouds. They weren't paying attention, they were daydreaming, or worst of all, they weren't in touch with reality.

For software engineers these days, it's the ones who *don't* have their heads in the clouds who are out of touch with reality. Everyone knows that the cloud-native environment is where it's at — where application development can be most efficient, where operations can be the most cost-effective, and where resilience and availability are the most likely.

REMEMBER

Simply put, cloud-native means building, deploying, and managing your applications in cloud computing environments. Applications that are born to live in the cloud tend to be flexible and resilient, easily scalable to meet the ups and downs of demand, and easy to update as needs change.

Indeed, being cloud-native means apps can be changed and updated quickly and frequently, with no impact on service delivery. Apps can be developed and optimized quickly and then undergo continuous improvement based on user feedback, all at the speed of business.

It's a tremendous competitive advantage that boils down to a few main pluses:

- **Greater efficiency:** Applications can be built quickly, with the help of automated tools and powerful cloud services. The cloud-native world is a great place for DevOps and continuous delivery practices to live.
- **Powerful scalability:** The components within a cloud-native app are isolated and can be scaled independently of one another. It's not uncommon for parts of an app to need faster updates than others, and it's easy to make this happen (the next section of this chapter explains more about how this works).
- **Exceptional availability and reliability:** Who wants downtime? No one, of course. In the cloud-native environment, updates can happen again and again and again, as often as you want, without impacting availability. Cloud-native apps are, as a whole, more resilient, and if one part of the app has issues, it won't bring the whole thing down.
- **Lower costs:** One thing that's missing from the cloud-native environment is a bunch of expensive physical infrastructure that your organization used to have to buy and maintain. Capital expenditures can drop, and the equally good news is that operational expenditures can see savings, too. And the scalability mentioned above means that computing and storage resources can be what they need to be right now, not what they possibly might need to be a few years from now. That saves money.

Running Cloud-Native Applications

One of the things that enables the magic of cloud-native development and improvement is the use of open-source components. The Cloud Native Computing Foundation (CNCF) helps make that happen, supporting the open-source community in the development of cloud-native components.

The CNCF has been around since 2015, and it has been regularly surveying members to see how the cloud-native world is unfolding. Its latest declaration is that 2022 was "the year cloud-native became the new normal."

Ask the CNCF what the building blocks of cloud-native architecture are, and you'll get a short list of elements that together are the secret sauce. Read on to learn more about microservices, containers, APIs, service meshes, and immutable infrastructure.

REMEMBER

Understanding the tremendous benefits of the cloud-native environment begins with an exploration of what exactly a cloud-native app is and how it's different from yesterday's applications. That understanding begins with one word — *microservices* — and in particular, the first half of that word, "micro."

Microservices are small and interdependent building blocks of a cloud-native application. Software developers break apart the various functionalities of their application into smaller microservices that each works on its own. Using these smaller pieces to create a greater whole is an incredibly powerful approach for a number of reasons.

For one thing, it allows developers to divide and conquer, with different tasks tackled by different people. Second, the concept allows developers to tweak and improve small pieces of the app without touching the rest of it. Even if one microservice fails, the application can continue to function while the developer fixes the problem.

Also, microservices make it a whole lot easier to add functionality or scale up the application's capabilities. And it doesn't take much in the way of computing resources to run any individual microservice.

Microservices stand in stark contrast to *monolithic* applications, which has been the traditional way of doing things for years. A monolithic enterprise application has one block structure that handles all of the necessary functionalities.

The monolithic method has developers working on lots of functionalities at the same time, before any of them can be released for testing. It's a far less flexible development method than the microservices approach. It takes a whole lot longer to deploy new software, and it doesn't tend to be scalable.

Think about microservices as being kind of like a marching band. The skills of each piece of the band are developed separately and then joined together to make outstanding music. Need it louder? Add more drummers. Want to build in a dazzling solo? Add the best trumpeter you can find. And if one of the flute players calls in sick, the band can still march on while the musician recovers.

The microservices approach is collaborative, just like that marching band. It's super-scalable, ready to react as the demand increases. It functions well on different platforms, too — just like the way that flutist can play a big role in a rock band, too.

Microservices developers have lots of great software tools in their toolkit, too, helping them to automate the way they build, test, and deploy functionality. Need a super-cool new function? There's a good chance it can be developed and deployed super-fast.

It takes fast and effective communication to bring together all of these separate and independent microservices onto the same page, functioning in harmony as a cloud-native application. That's where APIs come into the picture.

REMEMBER

API is short for *application programming interface*, which is the way the software programs exchange information with one another. APIs are used by a cloud-native app's independent microservices to communicate with one another. They essentially facilitate conversations, spelling out which data a microservice needs and which results it can give in return.

REMEMBER

Communication between multiple microservices is managed by a software layer in the cloud infrastructure known as a *service mesh*. By using the service mesh, it's possible to bring in new microservices with new functionality without having to write new code in the application.

The service mesh doesn't bring any of its own functionality into the application but rather handles all of the service-to-service communications. It sorts out the complexities of the complex microservices architecture to be sure that there's no cloud-native app equivalent of the famed movie quote from the late 1960s, "What we've got here is a failure to communicate."

REMEMBER

Containers are another key to the power of the cloud-native environment and are a cloud-native app's smallest compute unit. *Containers* are software components that isolate elements of cloud-native architecture, including applications and processes. A microservice's code, for example, is packed into a container, along with dependencies such as resource files, libraries, and scripts.

Each element lives in its own container, allowing it to run independent of physical resources. That means cloud-native apps are operating independently of whatever operating system and hardware on which they are running. Thanks to the flexibility that containers allow, developers can deploy cloud-native apps wherever they want — fully in the cloud, on hybrid clouds, or on premises.

Containers provide a powerful alternative to *virtual machines* (VMs), which emulate specific hardware systems by running software on top of physical servers. Each VM runs its own operating system, and it's possible to run VMs with varying operating systems on the same server.

When they came onto the scene, VMs were a great advance for running applications. Applications could be consolidated on a single system rather than needing their own servers. They reduced costs, sped up server provisioning, and improved disaster recovery.

But they also added their own complexities. Because each VM needs its own operating system image, that increases the memory and storage overhead. That makes development, testing, and production more complicated. And perhaps the biggest shortcoming when compared to containers is the limited portability when it comes to moving apps between traditional data centers and public or private clouds.

Containers, on the other hand, can be deployed practically at the snap of your fingers. They require fewer computing resources compared to what's needed with conventional app development. And the cloud computing resources they require can be scaled far more efficiently. All of these benefits greatly enhance business agility.

It's worth noting that containers can run inside virtual machines or on physical servers. They contain an application's library and processes, but they don't include the operating system, which means that they're lightweight and portable.

REMEMBER

Here's another distinction: the difference between mutable versus immutable infrastructure. *Mutable* essentially means that something can be changed, and immutable is the opposite — it can't be changed. An *immutable infrastructure* is part of the secret sauce of the cloud-native environment.

At first, that may have some people scratching their heads. With mutable infrastructure, you can upgrade key infrastructure elements without touching the apps or the data running there. And while that sounds like a good thing, upgrades are notorious for introducing issues and uncertainties.

Therein lies the benefit of unchangeability. Very long story short, it makes cloud-native deployment a whole lot more predictable. Rather than upgrade a server, you simply discard it and move to a new one.

REMEMBER

Moving from one place to another is easy when everything is containerized, as it is with a cloud-native app. That app is hosted on servers that will be unchanged once they are deployed, and if it ultimately needs more compute resources, the app is moved to a server with higher performance. It can happen instantly and automatically, with no risky manual upgrades and no downtime.

As long as we're talking about moving things around, it's worth bringing some compass points into the conversation. The cloud-native environment functions through lots and lots of what's known as east-west traffic.

REMEMBER

East-west traffic is essentially the movement of data internally, and the term can refer to data moving within a data center or network. In this case, the traffic is moving among the microservices and servers that make an app work. That traffic needs to flow instantly and without impairment, even though the various microservices might live in completely different places across public clouds, hybrid clouds, or on premises.

As a cloud-native application scales, this east-west traffic can grow exponentially. That growth brings along challenges in terms of network performance, security, and visibility. A cloud-native app won't thrive without proper management of east-west traffic.

REMEMBER

The obvious opposite is *north-south traffic,* which is the movement of data to and from external places, or to view it simply, to and from the world outside. Traditionally, these are the doors that have gotten the most attention from information-technology security professionals, because they're seen as the places that cyberthreats can find their way in. Traffic flowing in is said to be moving south, while northbound traffic is headed out.

Seeing the Importance of Kubernetes

As explained above, cloud-native applications rely on a collection of microservices that work together to do whatever it is that the application sets out to accomplish. These microservices and related elements take up residence in containers that are deployed in the cloud, on premises, or most likely, a combination of places.

It takes some serious coordination to bring it all together in the seamless way that users experience cloud-native apps. These days, Kubernetes is pretty much the standard for orchestrating cloud-native applications.

To continue with musical metaphors, move from the marching band to a symphony to talk about orchestration. Any symphony has a bunch of instruments that each has its unique part to play. It could sound like chaos, as it does when the symphony is tuning up. The conductor brings order to the chaos. Likewise, in the cloud-native world — with its multiple containers and microservices — orchestration ensures that they harmoniously interact, scale, and heal as needed.

REMEMBER

Kubernetes was originally designed by Google, was announced in 2014, and is now maintained by the CNCF. Its name is ancient Greek — it would be nice if it translated into "symphony conductor," but the reality is that it means "pilot" or "helmsman" (which when you think about it, is pretty appropriate, too). Sometimes folks will abbreviate it as *K8s,* with the numeral 8 standing in for the eight letters between the *K* at the beginning and the final *s* in Kubernetes.

In Kubernetes, global decisions come from a group of master nodes that are collectively known as the *control plane*. The control plane manages the various worker nodes and pods that make up a Kubernetes cluster and run the show in the containers. Kubernetes is in many ways like a world all its own, but it also constantly interacts with the underlying infrastructure, requiring careful tuning and configurations.

TIP

If that sounds fairly simple, well, it's not. Deploying and managing a Kubernetes cluster involves a lot of intricacies, from networking to storage configurations. That's why many cloud providers offer managed Kubernetes services, designed to ease the experience for users and really democratize the world of Kubernetes. Managed services make Kubernetes adoption accessible to both startups and big enterprises.

These managed services certainly ease the burden, but they don't eliminate all responsibilities on the part of users. In fact, Kubernetes users and infrastructure operators follow a *shared responsibility model*. Simply put, while the cloud provider handles many infrastructure-related tasks and takes responsibility for the security of the cloud, users remain accountable for specific configurations and, most important, the security of their applications and data.

That means those running cloud-native applications must ensure the security of all data they use, whether at rest or in transit. They need to defend their orchestration, which as noted earlier, these days is likely to be Kubernetes.

WARNING

Enterprises need to keep watch over their containerized applications, which are vulnerable to a whole range of exploits and threats and malware and zero-day attacks. They need to take responsibility for identity and access management, all the various configurations, encryption, network traffic protection, segmentation, and the like.

In short, cloud providers will ensure a secure foundation upon which cloud-native applications can be built and deployed. But users must take it from there and take full responsibility for the security of their cloud-native operations.

IN THIS CHAPTER

» **Seeing where security fits into the Kubernetes picture**

» **Making sure the foundation is secure**

» **Operating in a multitenant world**

Chapter 2 Securing Kubernetes

If containers and microservices are the present and the future, Kubernetes paves the path toward today's and tomorrow's successes. This chapter brings the conversation down to earth a bit for the obligatory discussion about security.

Read on to learn more about how Kubernetes fits into container orchestration from a security perspective, why security mustn't be an afterthought, what makes for a secure foundation, and how the concept of multitenancy is both a key to Kubernetes success and a challenge to its security.

Understanding Kubernetes Security

Kubernetes is an orchestrator for containers. It's the conductor responsible for helping containers and the microservices inside them make lovely music together.

Each container is a lightweight, stand-alone, portable package that includes everything needed to run what's inside, but it takes the expert orchestration of Kubernetes to effectively manage and scale the containers that make any given cloud-native app function. Kubernetes automates the deployment, scaling, and management of containers, ensuring that they're running where they should, that they're able to recover from failures, and in general, that they're working together efficiently.

And, you won't be surprised to know, that Kubernetes orchestration can allow all of this to happen in a secure way. After all, the title of this book as well as this chapter both have to do with Kubernetes in general and security in particular. But Kubernetes security doesn't just happen — if it did, there'd be no need for this book and the insights on the pages within.

WARNING

Indeed, because Kubernetes is a complex system built of many different components, it's not just a matter of switching on a security module or installing a security tool. There are various types of security risk that the layers and services in a Kubernetes cluster face, and security teams need to address these wide-ranging risks. They need to know how to secure such components as nodes, pods, networks, and data.

Kubernetes does, in fact, include some types of native security tooling, such as role-based access control (RBAC). But it isn't, in and of itself, a security platform, which means that you'll need to employ third-party security tools to help get the job done fully.

REMEMBER

But before diving more deeply into specifics about Kubernetes from a security perspective, it's worth giving more thought to the importance of security in a containerized environment in general. Security is a central consideration in any information technology deployment, but with regard to containerization, here's some food for thought:

- **Isolated workloads:** One of the key strengths of containerization is that containers operate independently. That's vital for a number of reasons, one of which being that a compromise in one shouldn't impact others on the same host. Proper security measures can help ensure that isolation.
- **Many microservices:** Containerization typically goes hand in hand with a microservices architecture that breaks the app into independent components. That inevitably means a larger attack surface, necessitating the strongest security for every container and all the communication that goes back and forth between them.
- **Rapid deployment:** Employing containers that are lightweight and easy to deploy fast is one of the big selling points

for development and operations teams. Security is essential for ensuring that haste doesn't make waste, and that only trusted and secure containers are deployed.

- **Portability:** Portability is a powerful advantage, allowing containers and microservices to move from one environment to another. It's vital, though, to ensure security in each environment, and to make sure that you don't replicate security issues between environments.
- **Immutable infrastructure:** Since you're not making modifications following deployment of individual servers, you must have a robust security posture from the start. Otherwise, vulnerabilities and misconfigurations will persist.
- **Orchestration challenges:** Container orchestration through platforms such as Kubernetes is complicated, requiring proper understanding and configuration of lots of different security features, including network policies, secrets management, and RBAC.
- **Visibility:** Ineffective monitoring is an invitation to security concerns, but the containerized world is a visibility challenge, given how dynamic it is and how any given element may be short-lived and transient.
- **Compliance needs:** Depending on the industry and the region where operations are taking place, compliance requirements may be strict, with seriously expensive penalties. Failing to properly secure containers could yield painful consequences.

WARNING

Those are some general realities about containerization that make security so essential. There are a number of security challenges that are specific to Kubernetes (although many of these are issues in other kinds of deployments, too).

- **Getting the configuration wrong:** This should come as no surprise because it's true with any IT deployment, but configuring Kubernetes incorrectly can lead to security vulnerabilities. Just one example is exposing sensitive ports to the public internet.
- **Image problems:** Containers are built from images, and if the images are insecure or outdated, it can be a security risk.
- **Getting exhausted:** If resource limits are poorly configured, that can open the door to resource exhaustion attacks.

That's an issue in which one container consumes all available resources, leading to denial of service.

- **Mismanagement of secrets:** Mishandling secrets is a sure risk for data breaches, so they must be stored securely and rotated regularly.
- **Inadequate runtime monitoring:** Real-time monitoring and alerting are vital for detecting and responding to security incidents promptly. In a complex environment such as Kubernetes, that can be a challenge.

REMEMBER

A bit of good news is that even though Kubernetes is an orchestrator rather than a security platform, it has elements that are helpful from a security perspective.

- **Staying safe with RBAC:** Kubernetes provides mechanisms for role-based access control that determine who can access and modify resources within the cluster. Only authorized people are allowed in.
- **Keeping things isolated:** Kubernetes ensures that containers are isolated from each other, which is a key to greater security. It means that they can't interfere with or access each other's data or processes. This isolation is crucial for security.
- **Employing secrets management:** As mentioned above, this is a potential challenge, but Kubernetes does have a built-in system for storing and managing sensitive information such as passwords and API keys.
- **Setting network policies:** By defining network policies to control how pods communicate with each other, you can make unauthorized network access more difficult.
- **Remaining up to date:** Perhaps the best way to stay safe is to ensure that you've got the latest version of Kubernetes running (which is, again, true with any IT deployment).

Building a Secure Foundation

A strong foundation is crucial for Kubernetes security, as it is with any IT implementation. Your organization will need to secure Kubernetes nodes using the same kinds of strategies that you use on any kind of server. It's vital to secure the underlying infrastructure, the operating system (OS), and the container images.

TIP

To begin with, the infrastructure in which Kubernetes runs must be secure from both physical and network-based attacks. Any infrastructure vulnerability or misconfiguration can have an impact on the overall security of a Kubernetes cluster. Preventive actions include such measures as network segmentation, solid firewall rules, and proper access controls.

Similarly, the host OS on which Kubernetes runs must be hardened and regularly patched in order to keep attack vectors at a minimum. To keep the OS from being a dangerous entry point, regular updates are a must, and any OS-specific security guidelines should be followed carefully. And in general, you'll want to minimize the attack surface, through such actions as removing any extraneous applications and libraries as well as user accounts that aren't needed.

TIP

A third key component to this solid foundation is confidence in secure container images. It's pretty much table stakes that container images must be free from vulnerabilities and malicious code. The application risk goes up with every insecure or unverified image. A secure image registry is a helpful solution, along with regular image scanning for vulnerabilities, establishment of best practices for building and signing images, and limiting base images to those from trusted sources.

The Kubernetes control plane can be seen as a part of the solid foundation. It is, essentially, the command center or brains of the Kubernetes cluster, managing and coordinating the activities there. It has a number of components intended to help that function be secure. One of the most critical is the *API server*.

This is, essentially, the entry point for interacting with the cluster. It's like a receptionist, receiving requests from users, developers, and other parts of the cluster, and deciding who gets in the door.

The Kubernetes API server authenticates users, service accounts, and other entities seeking to access the cluster. Acting sort of like a security guard, the API server does this through a number of authentication methods, such as client certificates and bearer tokens. Strong authentication means only authorized entities can interact with the cluster.

Just being approved entities doesn't mean they can do whatever they want, though. After authenticating an entity, the API server must decide whether that entity is allowed to perform whatever actions it is trying to do within the cluster.

Authorization policies are enforced using RBAC, attribute-based access control (ABAC), or other mechanisms. The idea is to ensure that users have the appropriate permissions to perform specific operations on cluster resources.

Control plane security components include *admission controllers,* which are there to enforce rules related to API requests. They can change a request object or deny a request altogether. And *controller managers* ensure that the cluster is running smoothly, checking to see that the actual state matches the desired state.

Then there's the *etcd,* a key value store that is the backing store for all cluster data, including highly sensitive data. And the *scheduler* determines which node does what and when based on resource requirements and other factors. All of the various control plane components have a hand in maintaining secure operations.

The Kubernetes term *data plane* generally refers to components that handle the actual data traffic and workloads within a Kubernetes cluster. Sometimes known as the *runtime plane,* it's where workloads run and communicate. Worker nodes on the data plane carry out commands that come from the control plane.

The main components of the data plane include:

- **Pods:** A pod is the smallest deployable unit that can be created and managed in Kubernetes. It can host multiple containers that together make up a single unit of deployment.
- **Services:** Services define a set of pods and establish how to access them, determining such things as load balancing traffic between pods.
- **Ingress:** External access to the services in a cluster happens by way of an application programming interface known as an ingress.
- **Kubelet:** This agent runs on each node in the cluster and ensures that the containers are running in a pod.

- **Kube proxy:** This network proxy runs on each node in the cluster. It implements part of the Kubernetes service concept, maintains network rules for pod communication, and allows network communication to and from Pods.

Implementing Multitenancy

Given that Kubernetes is the de facto standard when it comes to container orchestration, and that so much of what happens in this world lives in the cloud, any user can expect that there are neighbors out there operating nearby. Just as multitenant housing such as apartments and condos are developments where a lot of different people live in close proximity, a *multitenant cluster* is a Kubernetes cluster that's shared by multiple users and workloads, known as tenants.

REMEMBER

And just as any apartment complex has rules to ensure that all residents are safe and able to get along well with one another, a multitenant cluster has lots of mechanisms that isolate tenants from one another, allocate resources fairly, and ensure that no one is endangering neighbors.

For starters, think a bit more about that apartment complex and consider that any given unit is assigned to a specific person or people. That unit in the corner has been declared to be Mary Smith's apartment and nobody else's, and it might even have her name on the mailbox.

In a multitenant Kubernetes environment, *namespaces* serve the same kind of purpose. They're essentially like virtual clusters within a single physical cluster, providing a scope for names and offering a way to divide up cluster resources between multiple users, teams, or projects.

Here are some thoughts on how namespaces can help isolate tenants from one another:

- **Isolating resources:** It's possible to allocate resources such as CPU and memory for specific namespaces. That way one team or project doesn't monopolize all the cluster resources.

- **Controlling access:** RBAC can be employed to grant users access only to specific namespaces. A particular team can only access its own namespace and not that of a different team. As a matter of fact, it's also possible to use RBAC within a given namespace to grant even more specific permissions to users of that namespace.
- **Enforcing network policies:** Pods can, by default, communicate across namespaces, but network policies can be established and enforced to control those communications and add an extra layer of isolation.
- **Reducing naming confusion:** Just like there might be multiple people named George in a big apartment complex, it would not be a surprise if different developers in a multitenant environment coincidentally picked the same name for a pod. With namespaces, it's possible for resources in different namespaces to use the same name without any conflict.

RBAC essentially allows you to define who can do what within a Kubernetes cluster. A *role* or *cluster role* defines a set of permissions relating to Kubernetes API resources, such as get, list, create, and delete. A role is namespace-specific, but a cluster role applies cluster-wide.

Role bindings, meanwhile, assign roles to certain sets of users, groups, or service accounts. They are namespace-specific. *Cluster role bindings* serve the same purpose, except that they are cluster-wide.

A concept known as *least privilege* helps to ensure that all users and roles are minding their own business and don't have access to mind anyone else's business. It's a concept that's not unique to Kubernetes — it's a wise idea that helps ensure the tightest security in any system.

Least privilege is pretty much what it sounds like. When assigning roles and permissions, least privilege means that any entity gets only the permissions necessary to perform its tasks and nothing more. For example, if a user only needs to view pods in a namespace, then that user should not be granted permission to delete those pods (or, for that matter, to view or delete pods anywhere else beyond that namespace).

It's pretty obvious why least privilege is important from a security perspective. It's no different from that multitenant apartment complex mentioned earlier in which residents get keys only to their own space and no one else's.

But it also helps prevent mistakes. An apartment manager doesn't want a typical resident to mess with the wiring inside the walls, for the obvious reason that most residents really wouldn't know what they're doing and might end up burning the whole place down. Least privilege in a multitenancy implementation helps prevent accidental deletions or other mishaps.

Given the inherent Kubernetes complexity of distributed systems, multitenancy, and the scale of operations, network security is of paramount importance. There are a number of reasons.

REMEMBER

For one thing, services are constantly communicating with one another. Any security breach in one service can potentially compromise others if it's not properly isolated. In a multitenant implementation, network isolation and strict access controls are essential for preventing unauthorized access and breaches.

Network security also is vital for safeguarding against various attack vectors that come from constant internet exposure. And even if applications are trusted and container processes are contained, the defense-in-depth concept suggests that the network should add more layers of defense.

Network policies dictate how pods communicate with each other and with other network endpoints, enabling security at the microservices level. Pods are not isolated by default, but network policies can ensure that only allowed communications occur.

As discussed in general in Chapter 1, a service mesh is a dedicated infrastructure layer built to handle service-to-service communication in a microservices architecture. It provides such features as traffic management, observability, and security. Just as RBAC settings should follow a least privilege model, service mesh will ideally take a zero-trust stance.

REMEMBER

Zero-trust is a security concept in which no entity, internal or external, is trusted by default. It takes a skeptical eye toward all communications to ensure that nothing bad accidentally makes its way into the system. When zero-trust is the chosen approach, all access or communication requests must be authenticated and authorized.

A service mesh is an ideal place to build this kind of defense. Through a service mesh, policies can be enforced to control which services can communicate with each other, regardless of their location or network perimeter. A service mesh can enforce authentication from the originating user or service all the way to the destination service, which ensures that only legitimate communications are allowed to pass.

A zero-trust model effectively shrinks the attack surface, because even if an attacker gains access to the network, it can't freely communicate or access services. It's helpful from a compliance perspective, too, because a zero-trust model can help organizations meet whatever stringent regulatory and compliance requirements might be prevalent in their industry.

IN THIS CHAPTER

» **Learning the lifecycle of a container**

» **Spotting trouble early on**

» **Keeping a close eye on images**

» **Running Kubernetes securely**

Chapter 3 Securing Containers

Containers make the cloud-native world go around, and Kubernetes allows that to happen. This chapter digs more deeply into the container lifecycle and where security fits in, and discusses the importance of catching problems early in the game. It also points out why images must be scanned regularly, even before they're in production, and offers more detail into how to make runtime a safe time.

Exploring the Container Lifecycle

To get a solid handle on Kubernetes security, it's vital to be thinking about security throughout the container lifecycle. We'll get into more details about the security part of the story in a little bit, but first it's worth taking a big-picture look at that lifecycle, from build time to runtime.

By the time the *build time* phase gets going, developers have written the application code that will be deployed in containers, and they're now ready to build the containers. In this phase, the application code, dependencies, and runtime environment that have been specified by developers are packaged into a container image.

This typically involves creating a manifest that includes instructions for building a container image, and then using a container engine such as Docker to build the actual image. The manifest in this case will be known as a *Dockerfile*, and may start from a base image such as Ubuntu, install dependencies, and then add the application code.

REMEMBER

This container image built with these instructions includes everything the application needs to run. It's often tagged with a specific version to tell it apart from other builds. This step is important for image versioning and plays a vital role in the deployment process.

Once the image is built, it's often pushed to a container registry or image repository — perhaps Docker Hub, Google Container Registry, or Amazon Elastic Container Registry. The registry is a centralized place where built images are stored and managed.

REMEMBER

Now comes *deploy time*, which in the Kubernetes world means taking a container image and creating various objects that'll handle the lifecycle on a Kubernetes cluster. For example:

- **Pods:** The smallest deployable units in a Kubernetes cluster that can contain one or more containers.
- **Services:** These expose applications running on a set of pods as a network service.
- **Deployments:** These are controllers for managing the deployment and scaling of containers.
- **Ingress:** This is an API object that manages external access to the services in a cluster, typically by way of HTTP.

These objects are defined in manifest files and applied using Kubernetes command-line tools such as *kubectl*. This lets Kubernetes know how to handle and orchestrate the containers based on the images.

Now, the show begins — it's *runtime!* This is the phase where containers are actively running on the Kubernetes nodes. Kubernetes handles the orchestration, ensuring that the deployment's current state aligns with the desired state that has been specified in the deployment configuration. Kubernetes manages the scaling, self-healing, and updates, for these containers.

Here's just a bit more detail about the runtime part of the lifecycle:

- **Setting the schedule:** The scheduler selects a suitable node for the pod based on resource requirements, policies, and other constraints. It places the pod on a node where it should be able to efficiently operate.

- **Creating the pod:** Kubernetes interacts with the container runtime on the chosen node and pulls the relevant image from the registry. It starts the container within a pod.
- **Managing the lifecycle:** Now that everything is up and running, Kubernetes monitors the state of pods. If system failures or resource constraints cause a pod to fail, Kubernetes will restart or reschedule it. High availability is the goal in order to ensure that deployed services are reliably accessible and the desired state is maintained.
- **Scaling and updating:** Kubernetes seamlessly handles rolling service updates and scaling, doing so without causing any downtime. It watches traffic and other metrics to scale up and down by adjusting the number of pod replicas.
- **Ending the show:** If a pod needs to be replaced or a service is no longer needed, Kubernetes gently terminates processes. This frees up resources and makes the node available for new or different pods.

Identifying Issues Early

TIP

The wisdom is the same whether you're talking about a medical condition, a car problem, or any kind of information technology security consideration. The earlier you can spot a problem, the better off you'll be. Early identification of issues within Kubernetes is vital for multiple reasons:

- **Security:** Given that this is a book about security, you won't be surprised that this tops the list. The most immediate concern with vulnerabilities and misconfigurations is the security risk they pose. In Kubernetes, certain misconfigurations can leave your entire cluster exposed to attacks, and early detection can keep them from ever happening.
- **Trust and compliance:** For organizations in many industries, or those operating in certain states or countries, regulatory compliance is a huge deal. The consequences of failing to comply can be catastrophic from a financial perspective, which makes extra care early on a must.

- **Cost efficiency:** The longer a security vulnerability or misconfiguration remains in the development and deployment pipeline, the more costly it becomes to fix. Catch it early and you can save time and resources.
- **Continuity:** Downtime is costly from financial and customer-satisfaction perspectives. Preventing downtime through early detection is priceless.

TIP

These reasons underscore why it's so vital to integrate security from the earliest points in the development process and then continue them throughout. This is the "shifting left" concept that allows developers to identify and address security concerns as coding and building moves forward.

And it's just as important in runtime so that security teams can spot threats and vulnerabilities while in production, audit how things are going, and perform forensics as needed. Every organization building its future on Kubernetes needs a full-lifecycle approach that makes container security an integral part of the whole continuous integration/continuous delivery (CI/CD) pipeline.

To start with the coding phase, best practices include building code scanning fully into the CI/CD process. Doing so can keep malicious content from finding its way into production, and it can also improve code quality by spotting flaws early on. Dependency scanning is another key practice, because container apps have dependencies and libraries from various sources.

Your organization needs to be able to automatically assess one security policy across the entire cloud and throughout the development lifecycle. A healthy approach includes scanning infrastructure-as-code files for misconfigurations and security risks — across Dockerfiles, Kubernetes YAML manifests, and Helm charts.

The idea is to validate security policy compliance before container images are deployed in the cluster. Development teams are able to work autonomously to prevent risks if — right there in their usual CI/CD pipeline — they have a tool that secures container images and detects vulnerabilities and exposed secrets. With the right tool and approach, it's possible to validate compliance before images hit production but without slowing the development process.

Scanning Images Regularly

Once a cloud-native app gets to the build stage you're in the business of building containers, and it's vital to keep the closest eye possible on the container images from here on out. With a full-life-cycle solution, that means regular scanning.

TIP

Before containers are ever deployed, they should be scanned in registries. That means having a vulnerability scanner integrated into the registry to gain threat visibility. Policy checks are needed for each container, and for builds flagged as non-secure or out of compliance, security teams must decide whether to block them.

If issues are discovered, it should be simple to channel them to appropriate owners with Kubernetes-native environment segregation that's based on subscription, cluster, and namespace. When checking out tools, look for an API console and command-line interface so that developers can tackle the risks easily and quickly.

TIP

This is also where an admission controller, mentioned in Chapter 2, can come into play. An admission controller is a plugin that determines how a cluster is used, a gatekeeper that can check out API requests. It can mandate a security baseline across a whole namespace or cluster and reject deployments that are not up to snuff.

In the case of images, an admission controller can insist that images only be pulled from certain registries. It can deny unknown image registries, which is one great way to avoid ending up with noncompliant images in a Kubernetes environment. Beyond making judgments based on whether registries are trusted or not, an admission controller can take matters further by blocking unverified images.

Protecting Runtime

We mentioned "shifting left" above, which is essential as you roll out full-lifecycle security protocols. Security necessarily has to happen from the very beginning. But note that we're talking *full* lifecycle. Just because you're shifting left doesn't mean you're ignoring what's to the right.

Indeed, security teams must assess Kubernetes clusters from beginning to end, continually identifying threats and suggesting remediation. Continuously scanning Kubernetes clusters is the first step to identifying possible misconfiguration or excessive privileges threats.

Real-time scanning makes it possible to identify data access from the container to the cloud, to track lateral movement between the cluster and the cloud, or notice when privileges are escalated, which could be a sign of trouble. Indeed, a solid program must have full visibility into combinations of risks that together might be toxic, instances where two plus two equals an attack path to critical resources.

REMEMBER

Here's what runtime container security needs in terms of agentless scanning across the full container stack. It must be able to

- Map service accounts and network configurations while analyzing cluster structure and context, network, identities, and secrets. This offers insights across the cloud and workloads, not just inside the container. It reveals potential paths to and from the container as well as permissions to get total understanding of risk. It must be able to see when a cluster is exposed to the internet, and understand the compliance posture against benchmarks such as those from the Center for Internet Security (CIS).
- Scan images of all running containers, with snapshots of container hosts.
- Scan workloads themselves. That includes applications, containers, and virtual machines.
- Pull containers from the disc volume and check out their images for malware, exposed secrets, and other vulnerabilities.
- Scan registries that store container images.

An effective solution must keep a close eye on audit logs to see if anything abnormal might be happening. It should be able to gain visibility through the use of eBPF, which can enforce security policies, detect malicious behaviors, and then block unauthorized calls.

Again, the whole point is to keep a close watch for potentially malicious and unauthorized behaviors. Given the consequences of failures due to mishaps or attacks, prevention is by far the best-tasting medicine.

IN THIS CHAPTER

» **Seeing the compliance challenges of the cloud**

» **Keeping tabs on compliance**

» **Creating compliance controls**

Chapter 4 Keeping Kubernetes Compliant

No offense to the people in the compliance team, but developers want to focus on what they can do now and want to do next, not what they *aren't* allowed to do. Compliance is the last thing they want to think about, but an organization that neglects compliance isn't going to make it long-term.

This chapter focuses on keeping Kubernetes compliant. That's not always easy anywhere, but the cloud poses particular compliance challenges that are spelled out in this chapter. It also delves into compliance checks and how they can be automated and compliance controls that establish and enforce guardrails throughout the application lifecycle.

Defining the Challenges

Kubernetes security is, of course, the primary focus of this book and the reason you're flipping through the pages. Every organization that has any kind of information technology implementation has security concerns of one sort or another. Many enterprises also lose a lot of sleep over compliance concerns, especially those in highly regulated industries such as healthcare and finance, or those operating in Europe and other places where regulators are watching especially closely.

If your information technology needs include speed and agility (that's nearly everyone), you're looking to the cloud for its scalability, flexibility, and potential cost savings. But the cloud brings its own challenges to the table that can be particularly daunting for those with compliance on the brain.

WARNING

Here are some of the ways that compliance concerns can make life in the cloud problematic:

» **Protecting data privacy:** Just the acronyms GDPR and CCPA strike fear in the hearts of executives and compliance officers. Europe's General Data Protection Regulation as well as the California Consumer Privacy Act establish stringent rules for data protection, along with downright severe penalties for violations. Operating in the cloud can sometimes obscure where data is stored, processed, and transmitted — and where those things happen may make all the difference regarding which rules apply. Indeed, cloud computing often crosses international borders and happens in multiple jurisdictions.

» **Maintaining control and visibility:** Back in the old days when on-premises environments were the norm, you had much greater visibility and control over your systems and data. Data flows and access can be harder to monitor and control within a cloud infrastructure, which adds some potential challenges to compliance auditing and enforcement.

» **Dealing with data breaches:** Data breaches are a concern and a challenge anywhere and everywhere, but the multitenant nature of the cloud and the broader attack surface can ramp up the vulnerabilities. There are very specific compliance requirements when bad things happen, including breach notifications and detailed incident reporting, and those tasks have the potential to be more challenging in the cloud.

» **Keeping up to date:** The cloud environment is all about continuous deployment of applications and updates. Each update is a potential compliance risk, which means compliance checks must be built into the development pipeline.

- **Sharing responsibility:** As mentioned earlier in this book, cloud service providers operate under a shared responsibility model. The cloud provider takes responsibility for security of the cloud, but the customer is responsible for the security of what's in the cloud. From a compliance perspective, this can muddy the waters.
- **Trusting third parties:** Highly regulated entities in particular are on the hook for compliance, not just for their own activities, but for those things they farm out to third parties. Healthcare organizations, for example, must follow the Health Insurance Portability and Accountability Act or face its harsh penalties. They must be certain that their systems and software and third-party connections, including those in the cloud, are also HIPAA-compliant.

REMEMBER

Living and working in the cloud means thoroughly understanding the regulations applicable to your industry, as well as those that apply in the geographies where you're operating — and potentially the geographies where your cloud computing is happening. That necessitates a comprehensive compliance strategy for cloud applications.

This kind of strategy could include adopting specialized compliance tools, performing regular security and compliance audits, and continuous monitoring. Just one more thing to think about when making plans for Kubernetes security.

Implementing Compliance Checks

How can you be that sure your cloud-native operations are complying with all the regulations that are applicable? How do you know you're following best practices? Compliance checks. Comprehensive compliance checks, that is, all across Kubernetes, containers, and the underlying cloud.

TIP

The first step in implementing compliance checks is fully understanding what you're complying with. Does GDPR apply to your operations? Quite possibly, if you're dealing with European customers or operating in Europe in any way.

How about HIPAA? That's a no-brainer if your organization is a healthcare organization, but third parties with healthcare customers may also need to pay attention. And these days, all kinds of nontraditional companies are dipping their toes in the healthcare waters, even dollar stores out in the country.

What about PCI-DSS? Lots of organizations are savvy with that acronym, short for Payment Card Industry Data Security Standard. Their cloud-native apps need to be in step.

TIP

Beyond regulatory requirements, compliance checks can verify that your implementation is properly adhering to other controls and policies that you've established.

Policy-as-Code, for example, involves writing code in high-level languages to manage and automate compliance policies. It helps maintain consistent compliance standards across cloud infrastructure, containers, and Kubernetes.

Standardized configurations for containers and Kubernetes clusters, based on industry best practices, also provide a benchmark for meeting compliance standards. Strong access controls using role-based access controls (RBAC) in Kubernetes also establish a compliance foundation that can be checked and monitored.

As mentioned in Chapter 1, security is a continuous process. That's why, for example, you should continually scan container images and get yourself a tool that keeps a continuous watch over the whole picture, from the containers to Kubernetes to the cloud environment where it's all happening.

REMEMBER

In the same manner, compliance checking and monitoring are an ongoing, continuous need, too. Compliance checking should assess and verify your compliance posture against whatever framework or frameworks are appropriate for your business. Want to do that manually? Probably not, so you'll want a tool to do that on your behalf.

A full-fledged compliance solution will automatically and continuously assess the compliance status of cloud environments, comparing them with built-in or custom frameworks. Compliance tracking should be simple, whether it's within a single framework or cross-frameworks impacting all business units and applications in the cloud.

This is actually possible right out of the box, with the right tool. The solution from Wiz, for example, has more than 140 different built-in frameworks, a whole alphabet soup of compliance acronyms including HIPAA, NIST, SOC2, and dozens of others. It also allows creation of custom frameworks that align with a specific organization's needs. And there are dedicated Center for Internet Security (CIS) benchmarks for Kubernetes and Docker.

TIP

Reporting is, of course, a key to verifying and proving compliance. Any effective solution needs to be able to generate compliance reports that meet whatever requirements are applicable, for business units or applications, or at a high level for execs.

Building Controls

Checking compliance is vital, but building compliance controls is the key to ensuring that those compliance checks return rosy results. Implementing compliance controls means integrating policy enforcement at various stages of the lifecycle.

TIP

One way to go is using Open Source Software (OSS) standards such as the Open Policy Agent (OPA). It's a policy engine designed to enforce policies across microservices, Kubernetes, CI/CD pipelines, and API gateways, among other places.

Again, the first step is a full understanding of the specific compliance standards that your application and environment must adhere to. There's no way to enforce compliance if you don't understand the rules.

Then, deploy OPA on the Kubernetes cluster. As a flexible policy engine, OPA can enforce policies at various levels, from Kubernetes admission controllers to external APIs. OPA can function as a validating or mutating admission controller in Kubernetes, intercepting Kubernetes API server requests and modifying or rejecting them, guided by the defined policies. An admission controller can use OPA to audit or block deployments.

That leads to the next step, defining the policies. Here's where you write clear, fine-grained policies that are based on your compliance requirements, whatever they might be. These policies might specify resource limits or pod security contexts, or go so far as controlling network access between microservices or managing RBAC user access.

OPA can continuously monitor and enforce policies at runtime, but it also can be built into the CI/CD pipeline. That's how to spot violations early in the development process, before they hit production.

TIP

Like everything else in the ever-changing world of cloud-native operations, compliance requirements evolve. So, therefore, should the policies. It's important to regularly review and update OPA policies to reflect any changes in compliance standards, organizational requirements, or application design.

IN THIS CHAPTER

» **Understanding the need for container security**

» **Spotlighting the risks you must watch out for**

» **Pulling it all together with a security solution**

Chapter 5 Implementing a Container Security Solution

You picked up this book seeking answers, not just a recitation of how bad things are out there. This chapter is where the answers start to really take shape. It spells out the need for a container security solution, outlines the risks that such a solution will need to be watching for, and lets you know what to look for as you shop for a tool to handle container security and Kubernetes security posture management.

Seeing How Container Security Is Essential

Cloud Native Computing Foundation surveys have found that the vast, vast majority of organizations are using Kubernetes or evaluating it for their application deployments. More than 5 million developers are said to be in the business of developing solutions involving Kubernetes. It's pretty much everywhere — but is all this activity happening in the most secure manner possible? Good question.

WARNING

The business value of containers is abundantly clear. Here are some of the security challenges that containers pose:

- **Visibility may be completely lacking:** If visibility isn't sufficient, there's no good way to get an understanding of the containerized environment. It's an incredibly complex world out there regarding the configuration of container orchestration and cloud environments. Within the multi-step process are potential "toxic combinations" of individual risk factors that together open an attack path to critical resources.
- **The container itself may be vulnerable:** Cloud-native application development is powerful, but it's nevertheless like any other development approach in that it can allow security vulnerabilities into the libraries and software packages that are distributed in a container.
- **Scanning and monitoring are fragmented:** A complex environment means any vulnerability scanning and monitoring tools may be siloed in a way that's cloud-centric or cluster-centric or configuration-centric. On one hand, that siloed approach can totally miss key threats. On the other hand, it can generate a huge volume of alerts, without any good way to prioritize them.

WARNING

Container flexibility, which of course is a key selling point, is also a potential security problem. Containers can mount volumes and directories, and they're able to disable security features. If a hacker gains control, containers can run under their control as root in a "breakout" scenario, bypassing container isolation mechanisms and grabbing additional privileges.

Most Kubernetes users take advantage of managed Kubernetes services that cloud providers offer, but security in the cloud has always had its own challenges. The more complex the environment, with new workloads and architectures and roles and users, the harder it is to figure out which databases are exposed to the internet.

Given all that, it's hardly surprising that those CNCF surveys find security to be the top challenge associated with containers. Every business living in this world needs to prioritize container security and figure out how to:

- Identify vulnerabilities and misconfigurations
- Determine the containers that are internet-facing
- Get a handle on excessive permissions
- Put a lid on exposed secrets

That's how to proactively tackle container risk and shut down potential attack paths.

Determining Container Security Needs

So, enough with the bad news. This book would be a real downer if we just documented all of the problems without giving a hint to any solutions. The good news is that there are solutions to the complex question of container security. There are, indeed, tools that help your organization get a handle on this horror story.

TIP

As you peruse the possibilities, be sure that any tool you select is able to help you easily address the following key container security risks.

Finding container image vulnerabilities

One of the most common container risks is image vulnerabilities. That may be because of an insecure library or imported dependency, or it may reflect a threat that found its way in through a breach in the development environment.

It's crucial to keep such vulnerabilities from being introduced into the live environment. For that reason, a container risk solution must be able to scan images before they're used to create individual containers.

Spotting vulnerabilities in third-party libraries

One of the many benefits of apps built from microservices is that your developers don't have to reinvent every wheel. They can save tremendous amounts of time and effort by tapping into existing resources, including third-party libraries.

That said, every different source of code is a potential source of trouble. A lot of the third-party libraries out there run the risk of introducing bugs and other security vulnerabilities into the container environment. And with a complex cloud environment that crosses multiple platforms, diverse architectures, and thousands of different applications, it's all too easy to miss such vulnerabilities. Your container security solution must have visibility into this possibility.

Mixing up new misconfigurations

The interplay between host, cloud, container, and Kubernetes cluster is another source of potential trouble. You've got overlapping layers of configurations, networks, and identities all across the distributed cloud-native technology stack.

That's a pretty clear recipe for errors and security gaps such as accidentally exposing internal services. Your container security solution must offer a clear understanding of how all of these elements are connected to each other, to the corporate network, as well as to the internet. That's the only way to spot these inadvertent risks.

Going overboard with container privileges

Development teams need access to resources on the host, which is why they need privileged containers. But what if those privileged containers are breached? That can give a malicious actor those same generous privileges.

That's why a contained security solution must always be on the lookout for excessive privileges. If you're really going to successfully keep attackers from abusing root access in order to find and exploit vulnerabilities, you really need to limit and monitor container privileges as much as you can. Don't be handing out keys to the kingdom, because they might fall into the wrong hands.

Stashing sensitive data in the container writable layer

Each container has its own writable layer, which is where it stores such changes as the addition or modification of data. But if someone gains illicit or unauthorized access to the container, they'll be able to make changes and additions of their own, which means they could potentially find any secrets or other sensitive data that are being stored within the layer.

A container security solution must be able to scan the writable layer. It's the only way to be certain there isn't any sensitive data stored there, exposed to risk.

Painting the Big Picture

It would be simple enough to use the preceding section as a checklist for picking a container security solution, and you certainly need your solution to watch for all of these risks. But many tools still have shortcomings that you need to be aware of.

WARNING

For example, a lot of traditional tools work by installing an agent on the running resource. That's an approach that a lot of developers don't particularly like, as they see agents as a possible source of instability or performance reduction. Not to mention, agents on the running resource might have blind spots. A better approach would be API-based, which can get the job done in a cloud-native way.

Also, if security tools are siloed, they won't be able to truly correlate the three main types of risk: the container, Kubernetes, and the cloud. Without a big-picture view of that, security teams will lack the insight they need to prioritize risks and problems.

The only way to get a full understanding is with visibility into how the systems and resources are configured, how they connect to both the network and the internet, which identities have access, and which permissions have been given to those identities.

And here's a biggie: Many tools focus on production without paying attention to development. They miss key parts of the lifecycle and the continuous integration/continuous development (CI/CD) pipeline.

If a solution is only scanning a cloud-native app after it's been built, that makes security reactive. Spot a problem and you have to stop deployment, and isn't the whole point to enable safe deployment? The key to being an enabler rather than a stop sign comes from shift-left thinking, with security and development teams working together to reduce risks throughout the pipeline.

All that said, if security is going to be invited to the table throughout the feast, it can't get in the way and undermine the velocity and agility that your organization is seeking. Developers need to maintain their speed while they innovate, with well-defined security guardrails.

REMEMBER

The best bet is a single solution that handles container security and Kubernetes security posture management (KSPM). This one-stop-shop must have the capabilities for discovering and scanning containers, hosts, and clusters, all across cloud-managed and self-managed Kubernetes environments. That includes serverless containers and stand-alone containers running on virtual machines.

Sounds complicated, but we can boil it down to three essential functions you want in a container security/KSPM solution:

- **Enabling continuous visibility:** The security team needs instant visibility into what the environment looks like right now. Developers are spinning up new clusters and workloads all the time, and security personnel need to be able to keep track of this dynamic environment.
- **Providing context:** The security team needs lots of information, for sure, but they also need to be able to make sense of it. There's lots of noise and troublesome alerts from tools that monitor workloads, entitlements, compliance, vulnerabilities, and more. The container security/KSPM solution must offer a clear understanding of the risks so that responses can be properly prioritized.
- **Seeing every level:** Container security isn't enough. A helpful solution must also consider the network, the entitlements, and the cloud environment. Full context is vital.

IN THIS CHAPTER

» **Minding your own store**

» **Keeping everyone in their place**

» **Solving issues as early as possible**

» **Staying in compliance**

Chapter 6 Ten Kubernetes Security Best Practices

If you've made it this far, you've got some solid ideas in mind for Kubernetes security. If you like to read books from back to front, you're starting with the moral to the story. This chapter rounds up ten of the best practices for Kubernetes security that are detailed throughout this guide.

Sharing Responsibility

Managed Kubernetes services make implementing and running container-based applications a whole lot easier, but don't think you're off the hook for security. Under the shared responsibility model, the cloud provider takes responsibility for the security of the cloud, but users are accountable for the security of their applications and data.

REMEMBER

Your organization must keep watch over your containerized applications, which as you know are vulnerable to all kinds of exploits and threats. Identity and access management are your responsibility, along with all the various configurations, encryption, network traffic protection, segmentation, and other details.

Strengthening the Foundation

A secure Kubernetes implementation begins with a solid foundation, one that has a good handle on any infrastructure vulnerabilities or misconfigurations. Great preventive ideas in this regard include such things as network segmentation, solid firewall rules, and proper access controls.

To reduce attack vectors, harden and regularly patch the host operating system on which Kubernetes runs. Keep up with updates, follow security guidelines tied to the OS, remove extraneous applications and libraries, and delete unneeded user accounts.

Keeping Images Safe

Container images must be free of vulnerabilities and malicious code. You want a secure image registry, limits that ensure that images come from trusted sources, and solid practices for building and signing images.

REMEMBER

And, you need regular scanning. Images should be scanned in registries before containers are ever deployed. If issues are discovered there, they can be sent to the right owner.

Maintaining Isolation

Kubernetes is the dominant container orchestration solution, and the whole point of containers is isolating various parts of the puzzle so they can be moved around as needed, scaled when appropriate, easily fixed when broken, and improved when desired. They need to be orchestrated to work together, but they also need to be well isolated from one another.

Isolation ensures that containers can't access each other's data or processes, which is vital for security. Kubernetes also can allocate resources such as CPU and memory to specific namespaces so that no one hogs all the resources.

Getting a Handle on Access

One of the first steps toward keeping anything secure — information technology or your business office or your sportscar — is ensuring that only the right people have access. Kubernetes helps out with mechanisms for role-based access control (RBAC). Your enterprise determines who can access and modify resources within the cluster.

Indeed, you can use RBAC to secure individual namespaces so that only members of a specific team have access to a particular namespace. RBAC can even get more granular than that by limiting permissions within a team. The whole point is, anyone who has no business inside should stay outside, and the invitation list should be as short as possible. (Check the next item for more on that idea).

Ensuring That Privileges are Limited

This isn't just a Kubernetes thing but is a smart practice with any implementation. We're talking about the concept called least privilege. When you're assigning roles and permissions with RBAC, be sure you're not overly generous. Any given person or entity should have the absolute least amount of privilege needed to get the job done. No access to other places, no permission to do things outside of their job scope.

REMEMBER

Ensuring that people aren't accessing things that they shouldn't be accessing is important for security. But it's also important for preventing accidental mayhem. For example, perhaps a user should have permission to look at something but not change or delete it.

Trusting No One

Along the same lines, the security concept of zero trust helps minimize attack surfaces by putting extreme limits on access. The idea is that no person or entity, whether internal or external, is trusted by default. All communications must be authenticated and authorized.

Won't that hurt somebody's feelings, to feel downright untrusted? Well, sorry about that. The reality is, it's not necessarily that a person is untrustworthy — but what happens if that person's credentials are compromised? If somebody is so trusted that they have access to everything, and then a malicious actor gains access to that person's login, you've got a big problem.

Ensuring Early Warnings

Nobody wants bad news, but wouldn't you rather hear it at a point when you can do something about it? A key to Kubernetes security is spotting problems as early as possible, preferably before it's in production, before it wreaks havoc.

REMEMBER

If you can fix a misconfiguration early, you just may prevent an attack. If you can minimize the damage, you can reduce the cost of the impact and most likely also make the repair more cost-efficient. But to get that early warning, you need full and continuous visibility into the whole lifecycle, with an emphasis on getting your security processes rolling on the left side of the development cycle.

Keeping Showtime Smooth

Shifting left, as mentioned in the previous section, is vital. But so is keeping a close eye on operations once everything is in production. Runtime container security requires agentless scanning across the full stack. You want insights not just inside the container but also across the cloud and workloads. To really understand all risk, you need to know all potential paths to and from the container. Even if you prevent all the risks, being able to detect threats at runtime and correlate them with cloud events helps SOC and IR teams make quick decisions in case of a breach.

As long as you're scanning, keep scanning images of all running containers and all workloads. And keep on scanning registries that store container images, too.

Knowing Your Compliance Needs

Every organization has some set of rules with which it must comply. Certain industries have a lot of stringent rules — healthcare and financial services are two examples. And some geographic locations are heavy on regulation.

Part of your Kubernetes security picture is ensuring that you're fully in compliance with all of those rules. And the first step in that process is fully understanding which rules apply, so that when you establish regular and automated compliance checks you are monitoring everything you need to monitor.

Secure Everything You Build & Run in the Cloud

They say a demo is worth a thousand words:

Watch Wiz in action at *wiz.io/demo*

WIZ

Keep your cloud-native apps safe and secure

Cloud-native applications are flexible, scalable, and agile. They're the new normal for growing, competitive enterprises. Companies need to put security front and center, because these kinds of container-based applications have lots of vulnerabilities and complexities. This book shares best practices for hardening the platform that orchestrates your cloud-native apps, so your organization spots and fixes issues early, scans continuously, and keeps runtime safe.

Inside...

- Why cloud-native is the way to go
- How Kubernetes orchestrates the show
- Thriving in a multitenant environment
- Building security into the lifecycle
- How to stay compliant with Kubernetes
- Create a container security solution

WIZ

Steve Kaelble is the author of many books in the *For Dummies* series, and his writing has also been published in magazines, newspapers, and corporate annual reports. When not immersed in the *For Dummies* world or writing articles, he engages in healthcare communications.

ISBN: 978-1-394-24587-1
Not For Resale

9 781394 245871

for dummies®
A Wiley Brand