the GORILLA GUIDE® to...

Securing Cloud Native Applications on

Kubernetes

Foundation Edition

DAN SULLIVAN

INSIDE THE GUIDE:

- The 4 C's of Kubernetes Security
- Best Practices & Tools: Evaluating Security Controls Against Benchmarks
- Ensuring Kubernetes Application Security

KASTEN by Veeam

POWERED BY ActualTech MEDIA

Securing Cloud Native Applications on Kubernetes

By Dan Sullivan

TABLE OF CONTENTS

The 4 C's of Kubernetes Security....4

Cloud Native Security....5

Cluster Hardening....8

Container Security....11

Code Security....14

Evaluating Security Controls Against Benchmarks: Best Practices and Tools....16

Ensuring Kubernetes Application Security Requires Vigilance....17

Please register at Learning.kasten.io to get hands-on experience with Kubernetes.

ACTUALTECH MEDIA

6650 Rivers Ave Ste 105 #22489 | North Charleston, SC 29406-4829

www.actualtechmedia.com

Publisher's Acknowledgements

EDITORIAL DIRECTOR
Keith Ward

DIRECTOR OF CONTENT DELIVERY
Wendy Hernandez

CREATIVE DIRECTOR
Olivia Thomson

SENIOR DIRECTOR OF CONTENT
Katie Mohr

WITH SPECIAL CONTRIBUTIONS FROM KASTEN BY VEEAM®
Thomas Keenan
Joey Lei
Tom Leyden

ABOUT THE AUTHOR

Dan Sullivan is a principal engineer and architect specializing in cloud architecture, data engineering, and analytics. He has designed and implemented solutions for a wide range of industries and is the author.

The 4 C's of Kubernetes Security

Cyberattacks have been on the rise for several years, especially ransomware attacks, which are not only increasing in frequency but quickly gaining complexity and sophistication. As a result, modern deployments and applications are more at risk than ever. The attack surface is growing, and the intricacy of interlocking tools and environments means administrators, developers, and security pros must face the monumental task of properly securing their organization's resources.

Cloud security requires a joint effort by cloud providers and cloud users.

Kubernetes is a useful platform for deploying and maintaining applications, but it suffers from risk factors, as well. One of the biggest reasons Kubernetes environments present great security risks is that they're not monolithic. Kubernetes deployments often consist of a collage of third-party tools, in-house and open source code, and many other components that require constant updating and configuring. Also, when initially setting up Kubernetes, DevOps teams often over-permission access but forget to delete or lower these permissions later and this leaves security vulnerabilities. Securing Kubernetes also requires attention to keeping up with new releases to patch vulnerabilities and ensuring adequate backups to recover from some kinds of attacks.

This Gorilla Guide® To... Securing Cloud Native Applications on Kubernetes, Foundation Edition, covers the four primary concerns for securing Kubernetes deployments. These concerns can be thought of as the four C's of Kubernetes security:

- Cloud native security
- Cluster hardening
- Container security
- Code security

Each of these concerns can have direct impacts on the others and on the operational stability of your Kubernetes deployment. The following sections dive into the importance and security considerations of these four primary concerns.

Cloud Native Security

Cloud security requires a joint effort by cloud providers and cloud users. Cloud providers are responsible for securing their infrastructure and the supporting services they provide, while cloud users are responsible for controlling access to their cloud resources and ensuring that their cloud resources are configured properly. They must also check their applications for vulnerabilities that can compromise security.

Working with a cloud vendor to host your environment provides the benefit of built-in infrastructure security. Whether you're using Amazon Web Services (AWS), Google Cloud

Platform (GCP), or Microsoft Azure, cloud providers control access to the physical hardware that is the foundation of cloud computing. They also protect their networks to prevent unauthorized access and to mitigate the risk of network attacks, such as denial-of-service (DoS) attacks. In addition, most cloud providers provide encryption at rest and in transit for data stored in and moving through their infrastructure. Data security is just one security option offered by cloud vendors, as most providers have built out multi-layered security approaches to meet customers' compliance and regulatory requirements.

However, hosting services or environments with a cloud vendor doesn't exempt you from securing your resources. Cloud-native deployments work on a shared responsibility model, which means that both the provider and the client must secure various parts of the stack of compute, storage, and network services (see **Figure 1**). In addition to the security controls

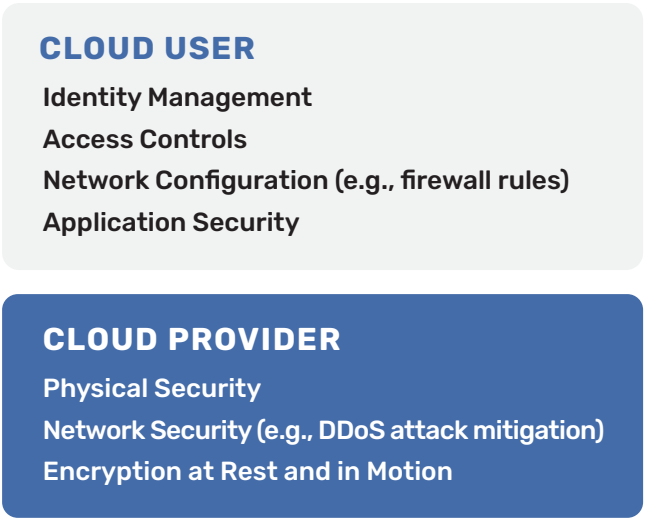


Figure 1: Cloud security is a shared responsibility

managed by cloud providers, cloud users should follow several recommended practices, including:

- Using identity and access control mechanisms to control who can perform specific operations on cloud resources.
- Configuring virtual private clouds, virtual private networks, and subnets to control the flow of network traffic.
- Monitoring and logging system operations to alert administrators of unexpected or problematic events and providing details for diagnosing and correcting problems.
- Implementing software engineering best practices, such as testing and code reviews, before releasing code into production.

The security of your Kubernetes clusters in the cloud builds on the security your cloud provider has implemented for protecting the infrastructure. It also augments the security controls that you have in place to control access to your cloud resources and to ensure your applications and networks aren't vulnerable to compromise.

Hosting services or environments with a cloud vendor doesn't exempt you from securing your resources.

Cluster Hardening

Cluster hardening is the process of securing your Kubernetes clusters, API access, and any applications running in a Kubernetes cluster. Since Kubernetes relies heavily on API communication, all API traffic within a cluster should be TLS encrypted. Securing information in motion is important in every part of a Kubernetes deployment, and API traffic is no exception. Kubernetes offers an API for managing TLS certificates and includes features for automatically rotating kubelet certificate authentication, as well as options for manually managing the certificate authority (CA) that confirms certificate authenticity. Components can be protected at many levels, such as the pod and the node (see **Figure 2**).

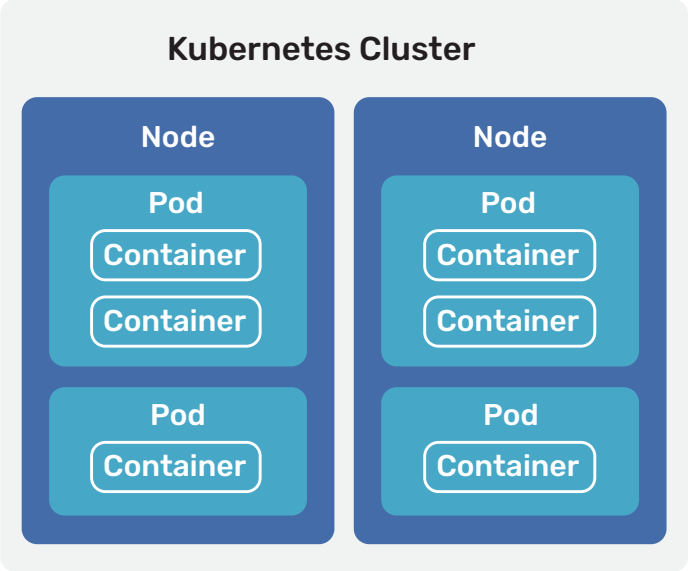


Figure 2: Cluster hardening needs to address multiple components in the Kubernetes architecture

It's important to remember that while Kubernetes supports service accounts, it doesn't support normal user accounts for authentication purposes. However, any user with a CA-approved certificate can make API calls. Every component that depends on API activity must be authenticated, including proxies, volume plugins, and the scheduler. Kubernetes includes service account automation controllers—a ServiceAccount admission controller, a Token controller, and a ServiceAccount controller—to help meet this requirement.

The admission controller automatically applies a service account and token volume to a created pod, then checks whether the pod has the appropriate configurations to receive a valid secret token. On the other side of this transaction, the Token controller creates a secret token for the newly created pods and deletes service account secrets when they're no longer needed. Finally, the ServiceAccount controller oversees service account creation within namespaces.

It's important to remember that while Kubernetes supports service accounts, it doesn't support normal user accounts for authentication purposes.

Access controls are also a part of securing API functionality. Kubernetes includes a Role-Based Access Control (RBAC) component that enables you to manage roles by defined groups. The key to properly using role-based access is to understand the scope of granted permissions. For instance, a

user whose role prevents allocating compute resources might still be able to if permissions enable the indirect manipulation of resources. Make sure to investigate any *indirect* control a given role might have to prevent users from changing aspects of a cluster not required for their job.

Limit access to the kubelet, which is the node agent that runs on each node. The kubelet agent is responsible for registering a node with the API server for using pod specifications to ensure containers are running as expected.

Take advantage of Kubernetes features to control the nodes a pod can access. Limiting this access can mitigate the risk of rogue code that makes its way onto a pod from compromising other resources running in the cluster.

The second major consideration when securing a cluster involves the applications running within that cluster. It's important to limit control privileges at the application level, just as for API-related actions. You should also closely monitor resource usage control and, like all access permissions, follow the principle of least privilege. This principle goes for container privileges, as well. The flexibility of Kubernetes architecture comes with great responsibilities for role and permission management, because simple commands such as *get* or *create* can have significant impacts on whole clusters.

Limit the loading of unwanted and unneeded kernel modules. Kernel modules extend the functionality of Kubernetes, but also increase the attack surfaces of the platform. If the functionality of a kernel module isn't needed, don't load the module. It's particularly important to periodically review the use of kernel modules. As applications and services change, the need for particular kernel modules may change, as well.

Restrict network access. You should only enable traffic that needs to reach a cluster. If services running on a cluster need to be accessible to external clients, consider using proxies to receive client requests and pass those on to the Kubernetes service rather than making the service publicly available.

Take advantage of Kubernetes features to control the nodes a pod can access. Limiting this access can mitigate the risk of rogue code that makes its way onto a pod from compromising other resources running in the cluster.

Finally, limit access to the cloud metadata API to limit the amount of information an attacker can gain about your cloud environment.

Container Security

A solid Kubernetes security strategy will address container security, as well. Containers are the mechanism for executing code, and it's crucial to secure their creation, deployment, and use.

An important part of building containers is scanning for vulnerabilities using tools such as Anchore Engine and Aqua Trivy. Modern applications build on a wide array of supporting libraries, modules, and operating systems. For example, even a simple Docker container specification can include a base operating system image, a programming framework with a variety of modules installed, and other third-party applications. Any of these components could have been built using modules known to have security vulnerabilities. By including vulnerability scanning in the image build process, you can detect known vulnerabilities before releasing an image.

A solid Kubernetes security strategy will address container security, as well. Containers are the mechanism for executing code, and it's crucial to secure their creation, deployment, and use.

In addition to scanning for vulnerabilities, it's a good practice to digitally sign containers, so that users can verify the image creator. This can help protect your Kubernetes cluster if your container registry is compromised and a malicious container is loaded into the registry. In that case, the malicious container wouldn't be signed by a trusted source and wouldn't be deployed to the cluster.

Containers are designed to isolate the operations of code running in that container, but there's a potential for malicious code to exploit vulnerabilities in containers and interfere with other containers running on a server. You can address this potential problem by using a sandboxed runtime such as gVisor to further isolate the container's behavior. gVisor doesn't allow an application to directly call system-level services. Instead, gVisor emulates system-level service calls in user space and executes calls to system-level services on behalf of the application (see **Figure 3**). This indirect execution enables gVisor to inspect system-level calls and prevent potentially problematic code from breaking out of the container.

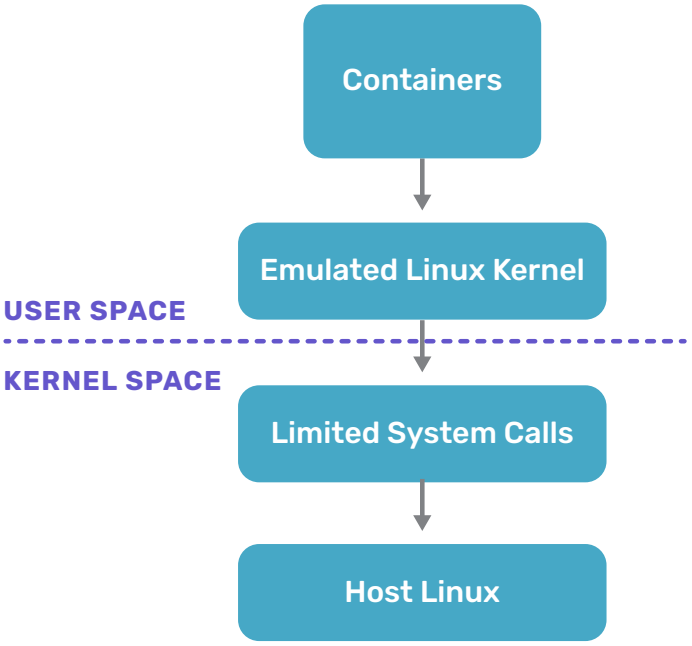


Figure 3: Container sandboxes such as gVisor provide additional controls for container security

Using network policies to isolate ingress and egress traffic to a container is a best practice. Network policies can control how a pod communicates with other resources and limit which pods can communicate and determine which namespaces and IP address blocks are accessible.

Code Security

Regardless of how you deploy your code, you must consider how to secure it. Obviously, you don't want to deploy intentionally malicious code or poorly coded applications that contain vulnerabilities. To reduce risk, software engineering practices such as code reviews are essential. Application stacks are complex; you can't manually review every line of code that's deployed into your environment. For this reason, it's important to scan third-party libraries for vulnerabilities and codebases for security errors using automated tools such as SonarQube.

To reduce risk, software engineering practices such as code reviews are essential.

You should also encrypt network traffic to help prevent the exposure of confidential information and limit an attacker's ability to understand how your systems work by analyzing unencrypted traffic.

Another good practice is limiting the port ranges used for communication to minimize the attack surface of your application.

Securing Application Code

Application code is one of the primary attack surfaces you have the most control over. While securing application code is outside the Kubernetes security topic, here are recommendations to protect application code:

- It's a good idea to encrypt network traffic between services. This can be done through a process known as mutual TLS authentication, or mTLS, which performs a two-sided verification of communication between two certificate holding services.
- Limit port ranges of communication. Wherever possible you should expose only the ports on your service that are absolutely essential for communication or metric gathering.
- Regularly scan your application's third-party libraries for known security vulnerabilities.
- Perform checks using automated tooling such as SonarQube that can scan codebases for common security errors.

Evaluating Security Controls Against Benchmarks: Best Practices and Tools

One question that might be difficult to answer is: "When have I done enough?" The four C's that we've gone over cover a lot of ground, and it can be hard to tell when your organization has created a well-secured Kubernetes environment. The Center for Internet Security (CIS) is a non-profit organization that studies IT security trends and techniques through its connections to security communities around the world. CIS provides security benchmarks for many platforms, including mobile devices, operating systems, virtualized servers, and more. Additionally, these benchmarks come as security guidelines for specific IT solutions and platforms, such as CentOS Linux, Cisco Network Devices, and AWS.

You should also encrypt network traffic to help prevent the exposure of confidential information and limit an attacker's ability to understand how your systems work by analyzing unencrypted traffic.

CIS also offers a benchmark for Kubernetes. The CIS benchmarks are a good way to measure the security of your Kubernetes deployment and pinpoint areas for improvement. They can help assess network isolation and policies, as well as several other configurations. Kube-bench provides benchmark tests for:

- Master node security configuration
- Etcd node configuration
- Control plane configuration
- Worker node security configuration
- Kubernetes policies

In addition, you can use the Docker-bench-security tool to check for common best practices when using Docker. The Aqua Trivy tool covers container images, files systems, and Git repositories, recognizing operating system packages and language-specific packages.

Ensuring Kubernetes Application Security Requires Vigilance

Kubernetes environments can be powerful but complex additions to your organization, and they must be secured. The rise in ransomware attacks in particular makes strengthening your Kubernetes security posture more important

than ever. Kasten by Veeam® was the first in the industry to introduce a Kubernetes ransomware protection solution. You can read more about that in the Kasten by Veeam podcast, "Cloud Native, Kubernetes & Ransomware: What's Your Last Line of Defense?" and blog post, "Goodbye to Ransomware: 2-Step Protection for Kubernetes Applications."

Preventing attacks on Kubernetes deployments requires a multifaceted approach that considers the inherent intricacy of large-scale cloud environments. The shared responsibility model, Kubernetes API-driven operation, access controls, container and cluster hardening, and more come together to create a secure ecosystem.

Kubernetes environments can be powerful but complex additions to your organization, and they must be secured.

This guide should be thought of as a set of recommendations. The four C's of Kubernetes security aren't a strict roadmap to perfect security; rather, they're a methodology to apply to your current and future Kubernetes deployments, and a way to think about Kubernetes security in general.

To learn more about Kubernetes tips and tools, visit the Kasten by Veeam Blog. If you'd like a hands-on Kubernetes experience, from building your first cluster to security and application training, you can find a series of labs at Learning.kasten.io. For more information on CIS benchmarks, visit its benchmark page at cissecurity.org.

About Kasten by Veeam

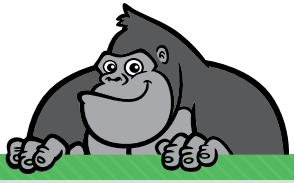
Kasten by Veeam® is the leader in Kubernetes backup. Kasten K10 is a Cloud Native data management platform for Day 2 operations. It provides enterprise DevOps teams with back-up/restore, disaster recovery and application mobility for Kubernetes applications. Kasten K10 features operational simplicity and integrates with relational and NoSQL databases, all major Kubernetes distributions, and runs in any cloud to maximize freedom of choice. Our customers are confident that their Kubernetes applications and data are protected and always available with the most easy-to-use, reliable and powerful Cloud Native data management platform in the industry. For more information, visit www.kasten.io or follow @kastenhq on Twitter.

About ActualTech Media

ActualTech Media is a B2B tech marketing company that connects enterprise IT vendors with IT buyers through innovative lead generation programs and compelling custom content services.

ActualTech Media's team speaks to the enterprise IT audience because we've been the enterprise IT audience.

Our leadership team is stacked with former CIOs, IT managers, architects, subject matter experts and marketing professionals that help our clients spend less time explaining what their technology does and more time creating strategies that drive results.



If you're an IT marketer and you'd like your own custom Gorilla Guide® title for your company, please visit https://www.gorilla.guide/custom-solutions/