vmware

Cloud Native Stack Security

ATTACK VECTORS AND COUNTERMEASURES

Abstract

This booklet highlights the risks, threats, and attack vectors associated with container technology and describes how to implement countermeasures to secure the entire cloud native stack.

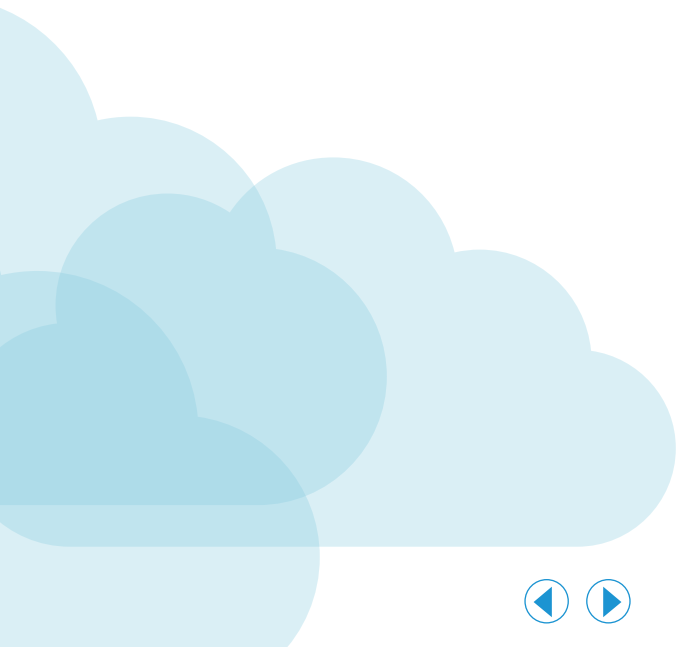

Table of Contents

Introduction

A containerized application requires full-stack security. Throughout the cloud native stack, risks and attack vectors abound, and an attacker can exploit a vulnerability to gain access to other containers, cloud resources, infrastructure, or data.

The main functional components of a cloud native stack, whether it's partly on premises or fully in the cloud, supply a prism through which security threats can be identified and solutions solidified.

Cloud Native Stack

A cloud native stack comprises the following layers:

1. Containerized applications.
2. Container management, including a container host, a container runtime, container images, and a container registry.
3. Orchestration system, such as Kubernetes.
4. Underlying infrastructure, such as VMware vSphere® or AWS.

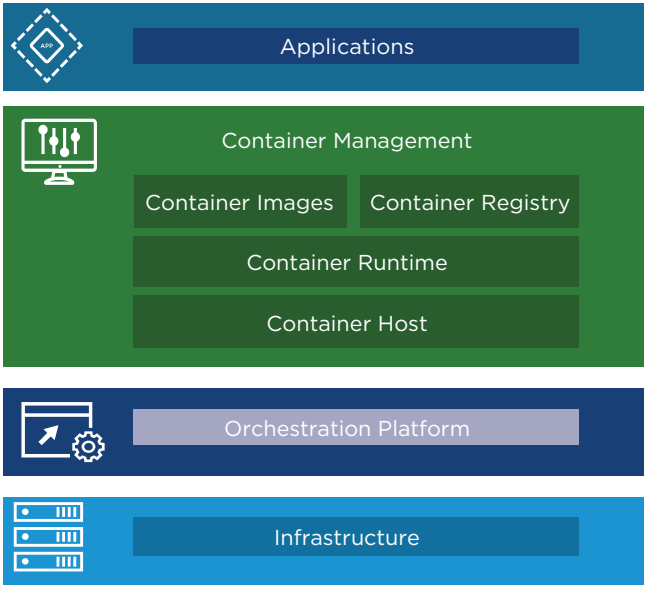

Figure 1: The layers in the cloud native stack that require security.

After exposing the stack's risks, threats, and attack vectors, this booklet explains how to secure the stack's components by applying container-specific countermeasures. A key part of the over-arching solution is to integrate the countermeasures into the container life cycle and the container platform.

Cloud Native Life Cycle

As a containerized application moves through the stages of development, testing, delivery, and deployment, engineers interact with container technology in the context of a continuous integration and continuous deployment pipeline (CI/CD) managed by DevOps—the people who help set the cultural tone for your cloud native lifestyle.

Each point of interaction with the CI/CD pipeline requires security, especially authentication and role-based access control. For full visibility, monitoring must take place across the container life cycle and the entire stack.

Attack Vectors and Countermeasures

The following sections identify the risks, threats, attack vectors, and countermeasures associated with each component in the cloud native stack. Because the container management layer includes several discrete components, each is addressed separately.

Containerized Applications

Just like traditional apps, containerized apps are vulnerable to software flaws, which an attacker can exploit to gain unauthorized access to sensitive data, attack other containers, or attack the host operating system. Without isolating containers on virtual machines, for instance, containers running on the same host can potentially connect to one another, creating a path through which an intrusion can spread.

Countermeasures should not only seek to eliminate vulnerabilities and reduce the attack service but also isolate containerized applications on virtual machines to impose strong security boundaries.

It's critical to point out that containers are not miniature VMs, and containers do not establish security boundaries as VMs do. An important implication of the NIST Application Container Security Guide is to run containerized applications on virtual machines. Containers "do not offer as clear and concrete of a security boundary

as a VM. Because containers share the same kernel and can be run with varying capabilities and privileges on a host, the degree of segmentation between them is far less than that provided to VMs by a hypervisor."[1]

But you can and should take isolation one step further—by segmenting containerized workloads by cluster. This approach is particularly powerful when a segmented cluster is a unit of tenancy in a multitenant context and when the isolation can be automatically and logically enforced in the network.

Packaging an application in a container also gives you a prescribed way to apply security principles with more depth and breadth. You can, for example, adopt core protection strategies to prevent exploits and protect an app's memory, configuration, and interfaces.

Key Countermeasures for Containerized Apps

- Profile an application and then monitor it for changes to its memory, end points, and configuration to ensure a known good state at runtime.
- Eliminate vulnerabilities and reduce the attack surface.
- Isolate containers on virtual machines to impose strong security boundaries.
- Segment each containerized application in its own cluster.

Container Images

Vulnerabilities, defects, malware, and plain-text password can mar container images—and such risks shift from possible to probable when the provenance of a container is dubious or unknown.

The portability of containers makes it easy to obtain an image and run it to solve an immediate problem, but using untrusted images in haste can introduce malware into an environment, lead to a data breach, or highlight a vulnerability. An image of unknown origin or with unknown base layers can contain malicious files.

1 NIST Special Publication 800-190, *Application Container Security Guide*, by Murugiah Souppaya, Computer Security Division Information Technology Laboratory; John Morello, Twistlock, Baton Rouge, Louisiana; Karen Scarfone, Scarfone Cybersecurity, Clifton, Virginia. September 2017. This publication is available free of charge from https://doi.org/10.6028/NIST.SP.800-190

The components in an image might be missing security updates or patches, exposing vulnerabilities that an attacker can exploit. Meanwhile, defects can take root in configurations, such as being run with more privileges than required, or in unnecessary components, such as the SSH daemon.

There is also the risk of embedded secrets that you don't know about. An image can contain plain-text passwords or private keys that let one component of an app connect to another, but anyone with access to the image can find the secret and use it to do harm.

Key Countermeasures for Container Images

- Keep images patched and up to date.
- Configure images to limit privileges and exclude unnecessary components, such as the SSH daemon.
- Securely store secrets, encrypted, in the orchestrator, not in the image.

Container Registry

A container image registry without curation and security steeps your cloud native environment in risk.

The risk starts with insecure connectivity. Allowing connections to a registry over an insecure channel gives attackers a vector that can be exploited to expose proprietary services or embedded secrets, steal credentials, or send spoofed images to an orchestrator.

Another risk is that without vigilance, vulnerable versions of images can linger in the registry, leading to accidental use. Curation is key to keeping images scanned, patched, up to date, and signed as trusted.

The most important risk centers on inadequate authentication and access control. A compromised registry can lead to contaminated containers, intellectual property theft, and many other risks. Clinical countermeasures are authentication with a standard directory service and role-based access control that applies the principle of least privilege and separation of duties.

Key Countermeasures for the Container Registry

- Scan images for vulnerabilities by using the Common Vulnerabilities and Exploits database.
- Sign images as known and trusted by using a notary.
- Set up secure, encrypted channels for connecting to the registry.
- Authenticate users and control access by using existing enterprise accounts managed in a standard directory.
- Tightly control access to the registry by using the principles of least privilege and separation of duties.
- Enact policies that let users consume only those images that meet your organization's thresholds for vulnerabilities.

Container Runtime

A containerized application with unrestricted outbound network access can let an attacker who has gained control of a container scan for other weaknesses to exploit. And one of those weaknesses can reside in an insecure runtime configuration, especially if a container is run in privileged mode or allowed to mount sensitive directories on the host.

In testing an app, developers can unleash unscanned images or misconfigured containers that turn into rogue containers, extending an open invitation to attackers, especially when such containers persist beyond a short test cycle.

Finally, there is the possibility of an attacker escaping from one container and jumping over to another. It's important to keep the container runtime itself (typically, Docker) patched and up to date to avoid the possibility of a container escape.

Key Countermeasures for the Container Runtime

- Restrict outbound network access of containers.
- Run images as non-privileged, immutable containers without SSH.
- Keep the container runtime patched and up to date.

Container Host

The container host is typically a Linux operating system that runs the Docker daemon. Without carefully selecting the right container host operating system and then taking pains to reduce its attack surface, you could give attackers a foothold into your environment.

Because containers share the Linux kernel, a hacker who escapes from one container can get into another and continue hacking away at your environment. One point of attack might be package vulnerabilities in the operating system. Are you sure all those packages are up to date and patched?

Another attack vector materializes when you manage a container through access rights granted to the underlying container host, creating unnecessary risk. A best practice is to manage containers through the orchestration system, not the container host.

To add defense in depth, another best practice is to only group containers on the same host if they have the same sensitivity level and threat posture. Grouping containers by sensitivity level and threat posture limits the attack surface and increases your likelihood to detect malfeasance.

Containers should also not require changes to basic aspects of the host operating system, such as the /etc or /boot directories, because such changes could potentially let an attacker elevate privileges, access other containers running on the host, or attack the host itself.

Key Countermeasures for the Container Host

- Use a container-optimized operating system to minimize the attack surface, reduce vulnerabilities, and add kernel-level security.
- Manage containers through the orchestration engine.
- Limit, log, and audit host OS access to detect anomalies and privileged ops.
- Add defense in depth by only running containerized applications with the same sensitivity level, purpose, and threat posture on the same host.
- Refrain from mounting sensitive directories on the host.
- Set the root file system to read-only.
- Keep the operating system fully patched and up to date.

Orchestration System

The multitenant context of orchestration systems like Kubernetes spells trouble if administrators and users have unbounded access—a malicious or just plain clueless user could sabotage container operations. In a worst-case scenario, a compromised privileged account could upend the entire system. Unauthorized access to the orchestrator can undermine not only containers but also their storage volumes.

Risk also underlies inter-node traffic, especially when it isn't monitored or when different apps share the same virtual network. A shared virtual network heightens risk because an attacker can use the network to attack the different apps using it, and possibly gain access to the most sensitive among them.

And that's why you shouldn't mix the workloads of apps with different sensitivity levels and threat postures. Placing them on the same host to, for example, use readily available resources increases the sensitive app's risk of compromise.

Unsecured Internet access to a Kubernetes Dashboard is a sure-fire way to create your own security nightmare. A recent analysis by a security consulting firm found more than 20,000 unsecured administrative dashboards for orchestration systems on the World Wide Web. At a few big companies, hackers were stealing cloud compute resources through the dashboards to mine cryptocurrency at utilization levels so low that the hijacking was difficult to detect.

Key Countermeasures for the Orchestration System

- Authenticate user access by using your standard enterprise directory service, not an ad hoc directory service dedicated to the orchestrator.
- Allow only authorized hosts to join a cluster.
- Impose role-based access control to limit user and admin access to clusters, containers, and hosts.
- Use RBAC to apply the principle of least privilege and separation of duties.
- Isolate containers on separate hosts based on the sensitivity level of the applications running in them.
- Monitor health.
- Log and monitor user access and their actions.
- Log and monitor resource consumption of containers to ensure availability of critical resources.

- Segment clusters by tenant.
- Isolate orchestrator traffic from workload traffic.
- Use logical switches and routers to isolate namespaces.
- Segment pods by applying dynamic security groups and network policies.
- Limit the interaction of pods by using micro-segmentation.
- Manage the lifecycle of the orchestration system and its components to keep nodes patched.
- Automatically repair and repave nodes when problems occur.
- Encrypt stored data.

Infrastructure

Finally, there is the infrastructure underlying the orchestration system. The infrastructure needs to be locked down, managed, controlled, and monitored just like infrastructure for anything else. Unprotected data in transit and in storage heightens the risk of a breach. Lack of logging, monitoring, and visibility not only for infrastructure itself but also across the entire stack can make it hard to identify intrusions and respond quickly.

Toward Fully Integrated Security

Many of the countermeasures outlined in the previous sections can and should be a standard part of your cloud native platform. To provide examples of how to implement countermeasures, the following sections demonstrate how VMware® Enterprise PKS and VMware® Cloud PKS secure containers and Kubernetes.

Full-Stack Security on Premises with VMware Enterprise PKS

This section demonstrates how VMware Enterprise PKS establishes four main security constructs to protect the cloud native stack:

- Private image registry
- Access control
- Micro-segmentation
- Logging and monitoring

Because these security constructs are integrated with VMware Enterprise PKS, they serve as examples of how a platform can put in place security controls and counter-measures on your behalf, saving you from integrating with existing security systems or building custom security components at great risk and expense.

Although VMware Enterprise PKS is a multi-cloud Kubernetes platform—it works with Google Compute Platform (GCP), Amazon Elastic Compute Cloud (EC2), Microsoft Azure, and VMware vSphere—this section focuses on using VMware Enterprise PKS on premises with vSphere.

Setting the Stage: VMware Enterprise PKS Architecture

The architecture of VMware Enterprise PKS combines Kubernetes, BOSH, VMware NSX®-T, and the Harbor image registry into a fully integrated cloud native stack.

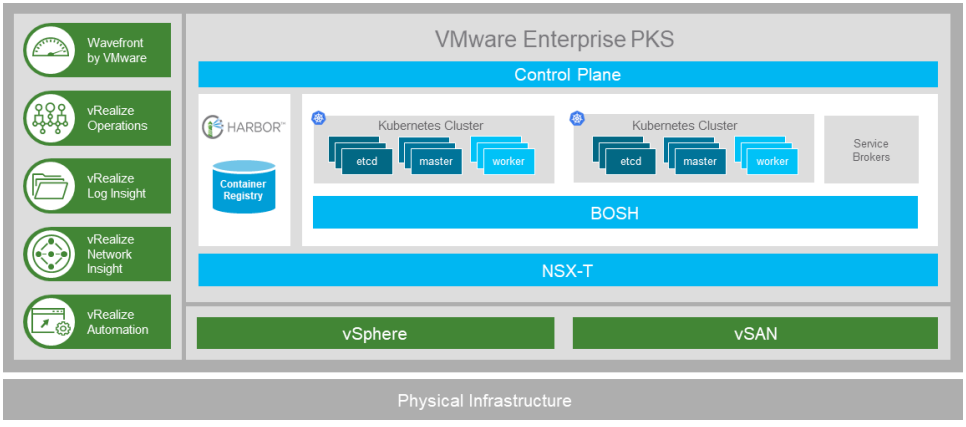

Figure 2: The architecture of VMware Enterprise PKS.

BOSH simplifies and automates the deployment and life-cycle management of Kubernetes and its components. BOSH supports Kubernetes deployments across Google Compute Platform, VMware vSphere, Microsoft Azure, and Amazon EC2. BOSH automates deployments and uses standardized, pre-hardened images. In addition, patching is automated to address CVEs, maintaining security with ease and without affecting workloads.

NSX-T lets you deploy networks with micro-segmentation and on-demand network virtualization for containers and pods.

Securing Container Images with a Private Registry

Harbor is an open-source, enterprise-class registry server that stores and distributes Docker images in a private registry on premises.

Harbor scans images for common vulnerabilities with Clair, signs images as trusted with Notary, secures images with role-based access control, and regulates the use of images with policies.

Part of the power of Harbor is that it secures images in the context of the continuous integration and delivery pipeline, as the following diagram illustrates, so the counter-measures protect resources where developers work.

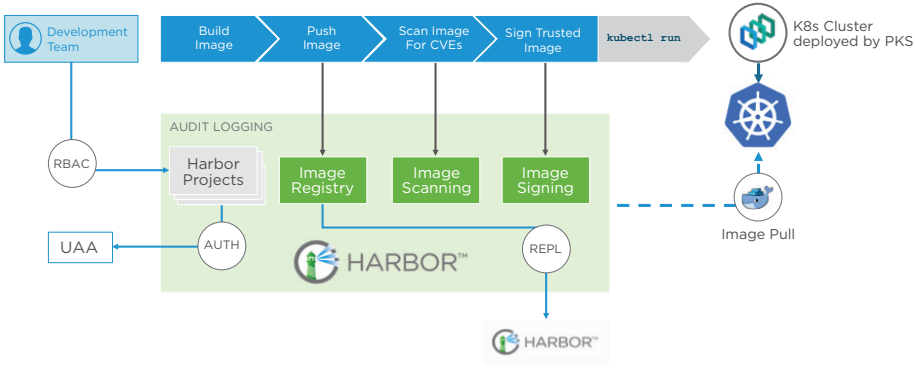

Figure 3. Harbor is a private image registry that scans images for vulnerabilities with Clair, signs images as trusted with Notary, secures images with role-based access control, and regulates image use with policies.

Controlling Access

VMware Enterprise PKS also provides an example of how a Kubernetes platform can integrate with established security systems for authentication and access control. VMware Enterprise PKS integrates with Microsoft Active Directory or LDAP to authenticate users and groups as they access the VMware Enterprise PKS command-line interface as well as the Kubernetes API. VMware Enterprise PKS controls access to Kubernetes clusters by assigning role-based access control to LDAP users and groups, letting you apply the security principles of separation of duties and least privilege.

Similarly, VMware Enterprise PKS secures access to the Kubernetes Dashboard by requiring authentication: Users need their kubectl credentials to access the dashboard. This requirement prevents unauthorized access to the Kubernetes cluster through a browser—so the dashboard isn't an open invitation for hackers to, for example, hijack your cloud compute resources to mine cryptocurrency.

Another part of the VMware Enterprise PKS platform that plays a role in access control is CredHub. It works with BOSH to provide centralized credential generation and management. You can use CredHub with BOSH to reprovision users with new credentials if a security anomaly is detected.

Automating Micro-Segmentation

VMware Enterprise PKS uses NSX-T to isolate each Kubernetes cluster with its own network segment. As a result, a Kubernetes cluster becomes a secure unit of tenancy in a multitenant context because the isolation is logically enforced in the network. Orchestrator traffic can be isolated from workload traffic.

In addition, NSX-T isolates namespaces by automatically creating logical switches and routers. Then, in this context, NSX-T uses Kubernetes network policies to establish

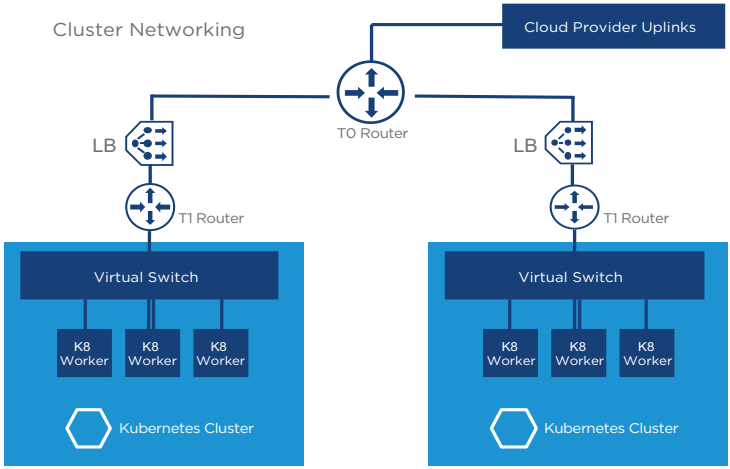

Figure 4. VMware NSX® automatically segments Kubernetes clusters with virtual routers.

micro-segmentation. Pods are segmented by applying dynamic security groups and policies. For portability, Kubernetes policies can be defined as part of an application's deployment, and NSX-T translates these policies into NSX policies and firewall rules.

Monitoring the Cloud Native Stack

Monitoring, logging, and analytics help protect a cloud native stack by giving you visibility into the use of resources, preferably across the entire stack. Optimally, you want to be able to monitor containers or pods, network traffic, clusters, hosts, the underlying infrastructure, and other components.

The Traceflow tool that comes with NSX, for instance, lets you monitor network traffic, trace packets from containers to physical networks, and observe traffic as it flows among other components in the stack.

Similarly, vRealize Operations can help you gain comprehensive visibility across applications as well as infrastructure. vRealize Log Insight analyzes logs for suspicious activity.

Wavefront by VMware monitors utilization of Kubernetes clusters. Monitoring utilization of Kubernetes clusters is important for two reasons related to security:

1. Availability is one of the three parts of the information security triangle upon which such compliance regulations as HIPAA and FISMA are founded: integrity, confidentiality, and availability.
2. There are attackers who seek to subtly hijack compute resources to, for instance, mine cryptocurrency but keep resource utilization low to avoid detection. Thus, gauging the resource utilization of your workloads and looking for anomalies, however subtle, can help protect resources.

Granular Security in the Cloud with VMware Cloud PKS

VMware Cloud PKS presents Kubernetes as a VMware Cloud Service so you can deploy and orchestrate containerized applications at any time and from any location without the overhead of setting up and managing either Kubernetes or its underlying infrastructure, which are managed for you by VMware. The public beta of VMware Cloud PKS is available through VMware Cloud Services.

VMware Cloud PKS includes several security features that embed security best practices and countermeasures in a Kubernetes-as-a-service model:

- Centrally managed user identities that are managed as part of the VMware Cloud Service.

- Multi-tenant access control policies.
- Granular role-based access control policies that encompass the entire VMware Cloud PKS service. The policies are translated into the Kubernetes RBAC model to maintain strict, granular access control on Kubernetes.
- The use of a minimalist operating system for the Linux host—called Photon OS—that is optimized and hardened for containers.

These security best practices and countermeasures are built into VMware Cloud PKS as system defaults—making VMware Cloud PKS and the Kubernetes clusters that run on it secure by default.

VMware Cloud PKS Architecture

The highly available architecture of VMware Cloud PKS includes a multitenant environment that aligns projects with your enterprise's organizational structure. Clusters exist in regions, and a region can be used by one or more organizations. Amazon Web Services provides the underlying, transparent infrastructure on which the Kubernetes clusters run, which enables you to integrate with AWS cloud services like machine learning and other application building blocks. The following diagram depicts how a global access policy framework encompasses the entire service.

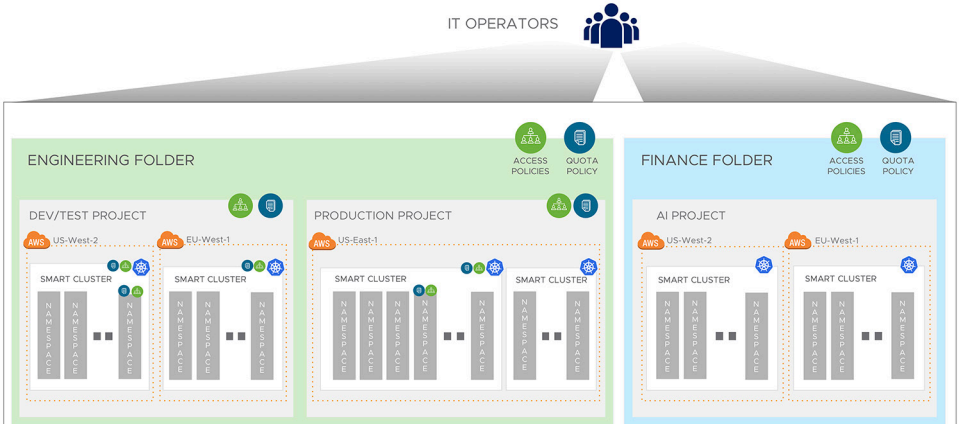

Figure 5. The architecture of VMware Cloud PKS includes a global access control policy framework that administrators can apply to organizational units, users, groups, projects, clusters, and namespaces.

Multitenant Access Policies and Kubernetes RBAC

VMware Cloud PKS includes access policies that you can apply to clusters and other managed resources, such as projects and folders. For access control, the policies are sets of roles that are bound to users and groups and inherited in the context of multitenancy. You can establish a multitenant hierarchy that can match your organization's structure.

Critically, VMware Cloud PKS pushes the policies to Kubernetes, which applies them by using the Kubernetes model of role-based access control. The roles grant permissions to perform such tasks as administer, edit, or view a cluster or namespace.

The granular role-based access control policies of VMware Cloud PKS impose key countermeasures for access and enable you to implement the principles of separation of duties and least privilege.

Lightwave Authentication and Access Control

VMware Cloud PKS uses an open-source security platform from VMware called Lightwave. It provides a directory service, a certificate authority, and a secure token service. Lightwave is a critical system component that gives VMware Cloud PKS the ability, through certificates and security protocols, to apply a range of security countermeasures for identity management, authentication, and access control.

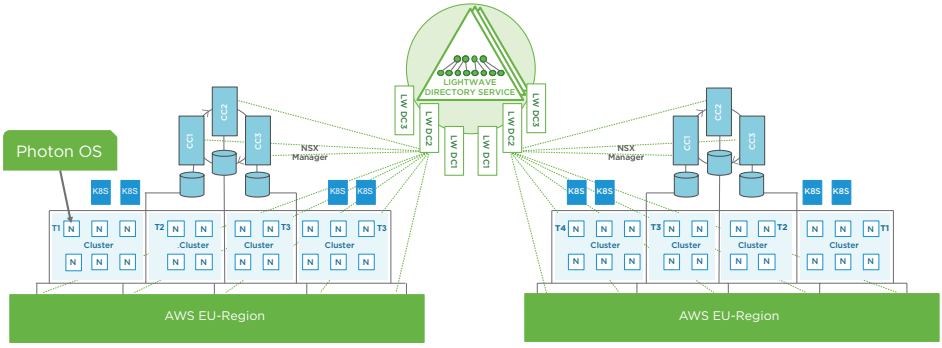

Figure 6. Lightwave is a security platform that is included with VMware Cloud PKS to authenticate users, control access, manage certificates, and provide a secure token service. Photon OS is the Linux container host for Kubernetes clusters in VMware Cloud PKS.

Linux Container-Optimized Operating System

VMware Cloud PKS demonstrates how a cloud native stack can use a minimalist container-optimized operating system to improve security for containers. Photon OS, a Linux container host from VMware, implements security countermeasures at the level of the host operating system through the following properties:

- **Minimalist:** The number of packages is limited to the minimum necessary for hosting containers.
- **Security-hardened Linux:** The kernel is configured according to the recommendations of the Kernel Self-Protection Project (KSPP).
- **Curated packages and repositories**: Packages are built with hardened security flags.
- **Advanced lifecycle management:** There are timely security patches and updates to container packages, such as Docker and Kubernetes.
- **Project Lightwave integration**: Lightwave clients are installed on Photon OS by default, which let it join a Lightwave domain and be managed by Lightwave consistently.

The critical mass of these features hardens the container host operating system and puts countermeasures in place for you as the default. By not running unnecessary system services, a minimalist operating system reduces the attack surface and mitigates risks associated with general-purpose operating systems.

Because VMware Cloud PKS is a managed service—its infrastructure, including the host operating system for Kubernetes nodes is managed by VMware on your behalf—Photon OS is automatically kept patched and up to date for you to minimize the risk vulnerabilities surfacing.

Data Encryption at Rest and in Transit

VMware Cloud PKS also applies data encryption by default to keep data safe in transit and at rest. When data moves between nodes in a Kubernetes cluster on VMware Cloud PKS, it is encrypted with TLS. Similarly, Kubernetes secrets are encrypted and stored in etcd. When VMware Cloud PKS is used with AWS, data at rest is stored on encrypted Amazon EBS volumes; the keys are managed by Amazon.

Infrastructure Isolation

Each organization is mapped to a AWS account managed by VMware Cloud PKS to isolate the organization in a multitenant environment, and each production cluster is deployed on a separate network segment to isolate the infrastructure from other organizations and workloads—structures that impose countermeasures at the level of the system on your behalf without adding additional management overhead.

Monitoring Kubernetes Clusters and Containerized Applications

VMware Cloud PKS includes a monitoring framework that is compatible with Wavefront® by VMware® for real-time visibility into the operations and performance of containerized applications and Kubernetes clusters.

Conclusion: Checklist of Countermeasures

The previous sections demonstrated how VMware Enterprise PKS and VMware Cloud PKS implement a range of countermeasures to secure the components and resources of a cloud native stack. If you are implementing a cloud native platform, here's a concise summary of the countermeasures discussed in this paper. You can use the checklist to evaluate whether countermeasures are included in a platform or component by default or whether you have to put them in place yourself. In the end, however, the result should be the same: A cloud native stack that is protected from the top to the bottom with fully integrated security.

- Implement container-specific countermeasures
- Integrate countermeasures into the container life cycle and pipeline, from build through the registry and runtime through orchestration
- Monitor containers across their life cycle and stack for full visibility
- Enforce security with policies, especially RBAC and policies for image use
- Use only the latest known, patched, scanned, and signed images
- Run images as non-privileged, immutable containers without SSH, etc.
- Manage containers through the orchestration engine, not the container host
- Securely store secrets, encrypted, in the orchestrator, not in the image

- Connect to registries and dashboards over secure, encrypted channels
- Tightly control access to registries, orchestrators, and dashboards with RBAC using principles of least privilege and separation of duties
- Control access to the Kubernetes API
- Federate existing accounts by using a standard directory and implement single sign-on
- Log, monitor, and audit registry, orchestrator, and dashboard access
- Encrypt data at rest using container-specific methods (see NIST 800-111)
- Segment orchestrator network traffic into discrete virtual networks by sensitivity level
- Only mix workloads of the same sensitivity level and threat posture on the same host
- Use a patched, up-to-date runtime
- Constrain network access from containers
- Profile and protect apps at runtime to ensure known good
- Use an up-to-date container-specific minimalist OS to narrow the attack surface; see NIST SP 800-123
- Set the root file system to read-only
- Limit, log, and audit host OS access to detect anomalies and privileged ops
- Limit resource consumption of a container to thwart denial-of-service (DoS) attacks
- Monitor the cluster and network utilization
- Monitor for suspicious activity and analyze failed login and RBAC events
- Keep your system patched and use recent versions of Kubernetes, which have stronger security than older versions
- Monitor configurations, such as dashboard access, for risks and vulnerabilities
- Routinely test for vulnerabilities and attack vectors by using standard tools

References

NIST Application Container Security Guide

NIST Security Assurance Requirements for Linux Application Container Deployments

Cloud Native Stack Security: How VMware Enterprise PKS Secures Containers and Kubernetes

Control Access with VMware Cloud PKS

Containers on Virtual Machines or Bare Metal? Deploying and Securely Managing Containerized Applications at Scale

Photon OS: A Linux Container-Optimized Operating System

A Dash of Security: Locking Down Kubernetes Admin Access

Securing Cloud Platforms with Project Lightwave

Impose Security by Default with VMware Cloud PKS

Glossary of Cloud Native Terms

vmware®