NeuVector
BY SUSE

Guide

The Ultimate Guide to Kubernetes Security

The Ultimate Guide to Kubernetes Security

How to Secure Your Kubernetes Pipeline

The Importance of Securing Container Deployments

Containers and tools like Kubernetes enable enterprises to automate many aspects of application deployment, providing tremendous business benefits. But these new deployments are just as vulnerable to attacks and exploits from hackers and insiders as traditional environments. Ransomware extortion, cryptojacking, data theft and service disruption will continue to be used against new, container-based virtualized environments in both private and public clouds.

To make matters worse, new tools and technologies like Kubernetes and managed container services in the public cloud will themselves be under attack as a gateway into an enterprise's prized assets. The recent Kubernetes man-in-the-middle vulnerability and exploit at Tesla are just the first among many container technology-based potential exploits expected to proliferate in coming months and years.

The hyper-dynamic nature of containers creates the following security challenges

1. **Vulnerabilities introduced in the CI/CD pipeline.** The heavy use of open source components and the continual discovery of critical vulnerabilities affect container images in the build phase, in registries, and in production.
2. **Explosion of east-west traffic.** While monolithic applications could be secured by traditional firewalls and host security tools, containers can be dynamically increasing the east-west, or internal traffic that must be monitored for attacks.
3. **Increased attack surface.** Each container may have a different attack surface and vulnerabilities which can be exploited. Also, the additional attack surface introduced by container orchestration tools such as Kubernetes and Docker must be considered.
4. **Automating security to keep pace.** Old models and tools for security will not be able to keep up in a constantly changing container environment. Given the automated nature of Kubernetes, containers and pods can appear and disappear in minutes or seconds. Application behaviors which can include new network connections must be instantly factored into enforced security policies. Next-generation automated security tools are needed to secure containers, declaring security policies early in the pipeline and managed as code.

While it can be argued that containers are by default more secure than traditional applications because they should have limited function and specialized interfaces, this will only be true if cybercriminals and nation-state actors launch attacks using old techniques against code and infrastructure with no vulnerabilities, and which has been locked down against all possible threat vectors. But we know that in practice this is not possible. And even if it were, you'd still want to do real-time monitoring for attacks. As time and experience have shown after repeated incidents, the sophistication of attackers always matches or outperforms new infrastructure approaches. Bad actors are constantly developing new and novel ways to attack containers.

Security-related questions to ask your kubernetes team

- Do you have a process in place to eliminate critical vulnerabilities (with fixes available) early in the pipeline, even in the build phase?
- Do you have visibility of Kubernetes pods being deployed? For example, do you know how application pods or clusters are communicating with each other?
- Do you have a way to detect bad behavior in east-west traffic between containers?
- Do you know how to determine if every individual pod is behaving normally?
- How are you alerted when internal service pods or containers start to scan ports internally or try to connect to the external network randomly?
- How would you know if an attacker gained a foothold into your containers, pods, or hosts?
- Are you able to see network connections and inspect them to the same degree as you can for your non-containerized deployments? At Layer 7, for instance?
- Are you able to monitor what's going on inside a pod or container to determine if there is a potential exploit?
- Have you reviewed access rights to the Kubernetes cluster(s) to understand potential insider attack vectors?
- Do you have a checklist for locking down Kubernetes services, access controls (RBACs), and container hosts?
- When you have compliance policies, how do you enforce the compliance at run- time? For example, to ensure encryption for your internal pod communication, how do you know when a pod is not using the encryption channel?
- When troubleshooting application communication or recording forensic data, how do you locate the problem pod and capture its logs? How do you capture raw traffic and analyze it quickly before it disappears?

This guide will present an overview for securing Kubernetes and container deployments, with a special focus on automating run-time security.

First, it's important to understand how Kubernetes works and how networking is handled.

How Kubernetes Works
The basics

For those not familiar with Kubernetes, this is an introduction into the key concepts and terms.

Kubernetes is an orchestration tool which automates the deployment, updating, and monitoring of containers. Kubernetes is supported by all major container management and cloud platforms such as Red Hat OpenShift, Docker EE, Rancher, IBM Cloud, AWS EKS, Azure, SUSE CaaS, and Google Cloud. Following are some of the key things you need to know about Kubernetes:

- **Master node.** The server which manages the Kubernetes worker node cluster and the deployment of pods on nodes. Nodes can be physical or virtual machines.
- **Worker node**. Also known as slaves or minions, these servers typically run the application containers and other Kubernetes components such as agents and proxies.
- **Pods.** The unit of deployment and addressability in Kubernetes. A pod has its own IP address and can contain one or more containers (typically one).
- **Services.** A service functions as a proxy to its underlying pods and requests can be load-balanced across replicated pods. A service can also provide an externally accessible endpoint for access to one-or-more-pods by defining an external IP or NodePort. Kubernetes also provides a DNS service, router, and load balancer.

Key components which are used to manage a Kubernetes cluster include the API Server, Kubelet, and etcd. Kubernetes also supports a browser-based management console, the Kubernetes Dashboard, which is optional. Any of these components are potential targets for attack. For example, the Tesla hijack exploited an unprotected Kubernetes console to install crypto mining software.

Kubernetes role-based access controls

Kubernetes role-based access controls (RBACs) provide granular management of resources. They can enable access to application workloads as well as Kubernetes system resources. Management tools such as OpenShift may add additional capabilities but rely on or use native Kubernetes basic security controls. It is critical to properly configure access controls to prevent unauthorized access to Kubernetes components such as the API Server or application workloads.

Kubernetes networking basics

The key networking concept in Kubernetes is that every pod is assigned its own routable IP address. Kubernetes (actually, its network plug-in) takes care of routing all requests internally between hosts to the appropriate pod. External access to Kubernetes pods can be provided through a service, load balancer, or ingress controller, which Kubernetes routes to the appropriate pod.

Kubernetes uses iptables to control the network connections between pods (and between nodes), handling many of the networking and port forwarding rules. This way, clients do not need to keep track of IP addresses to connect to Kubernetes services. Also, port mapping is greatly simplified (and mostly eliminated) since each pod has its own IP address and its container can listen on its native port.

With all of this overlay networking being handled dynamically by Kubernetes, it is extremely difficult to monitor network traffic, much less secure it. Following is an example of how Kubernetes networking works.
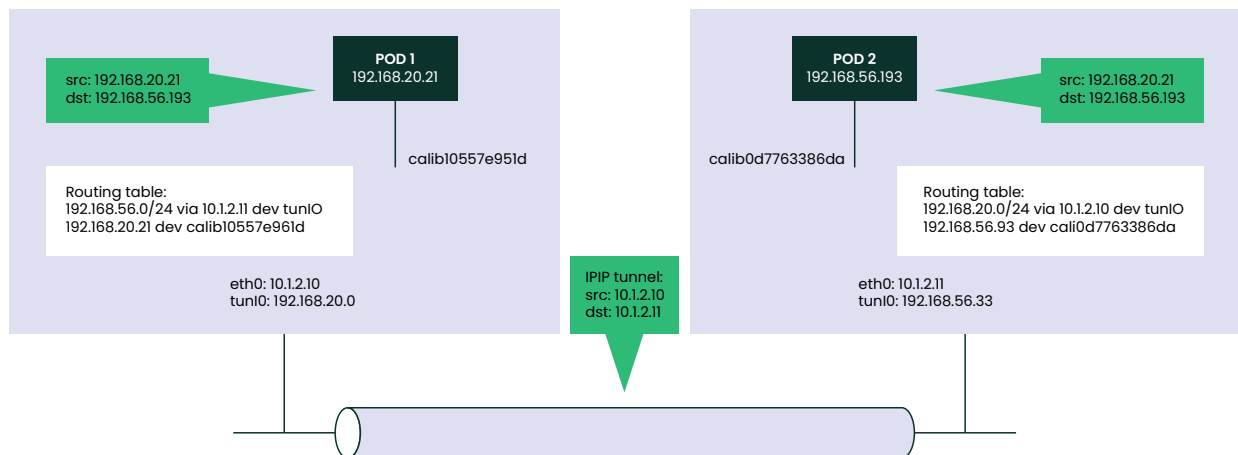
The above diagram shows how a packet traverses between pods on different nodes. In this example, the Calico CNI network plug-in is used. Every network plug-in has a different approach for how a pod IP address is assigned (IPAM), how iptables rules and cross-node networking are configured, and how routing information is exchanged between the nodes.

1. Once the CNI network plug-in receives a notification from Kubernetes that a container is deployed, it is responsible for assigning an IP address and configuring proper iptables and routing rules on the node.
2. Pod1 sends a packet to Pod2 either using Pod2's IP or Pod2's service IP as the destination. (Pod2's IP is used in the diagram.)
3. If the service IP is used, the kube-proxy performs load balancing and DNAT, and translates the destination IP to the remote pod's IP.
4. The routing table on the node determines where the packets should be routed.
   a. If the destination is a local pod on the same node, the packet is forwarded directly to the pod's interface.
   b. Otherwise, the packet is forwarded to the proper interface depending on whether overlay networking or L3 routing mechanisms are employed by the network plug-in.
   c. In the above diagram, the packet is sent to the IPIP tunnel interface and encapsulated with an IPIP tunnel header.
5. When the packet reaches the destination node, the encapsulation is removed.
6. The routing table on the remote node routes the packets to the destination Pod2.

With the routing, possible NAT, and encapsulation occurring and being managed by the network plug-in, it is extremely difficult to inspect and monitor network traffic for attacks and connection violations.

Kubernetes Vulnerabilities and Attack Vectors

Attacks on Kubernetes containers running in pods can originate externally through the network or internally by insiders, including victims of phishing attacks whose systems become conduits for insider attacks. Here are a few examples:

1. **Container compromise.** An application misconfiguration or vulnerability enables the attacker to get into a container to start probing for weaknesses in the network, process controls, or file system.
2. **Unauthorized connections between pods.** Compromised containers can attempt to connect with other running pods on the same or other hosts to probe or launch an attack. Although Layer 3 network controls whitelisting pod IP addresses can offer some protection, attacks over trusted IP addresses can only be detected with Layer 7 network filtering.
3. **Data exfiltration from a pod.** Data stealing is often done using a combination of techniques, which can include a reverse shell in a pod connecting to a command and control server and network tunneling to hide confidential data.

Worker nodes
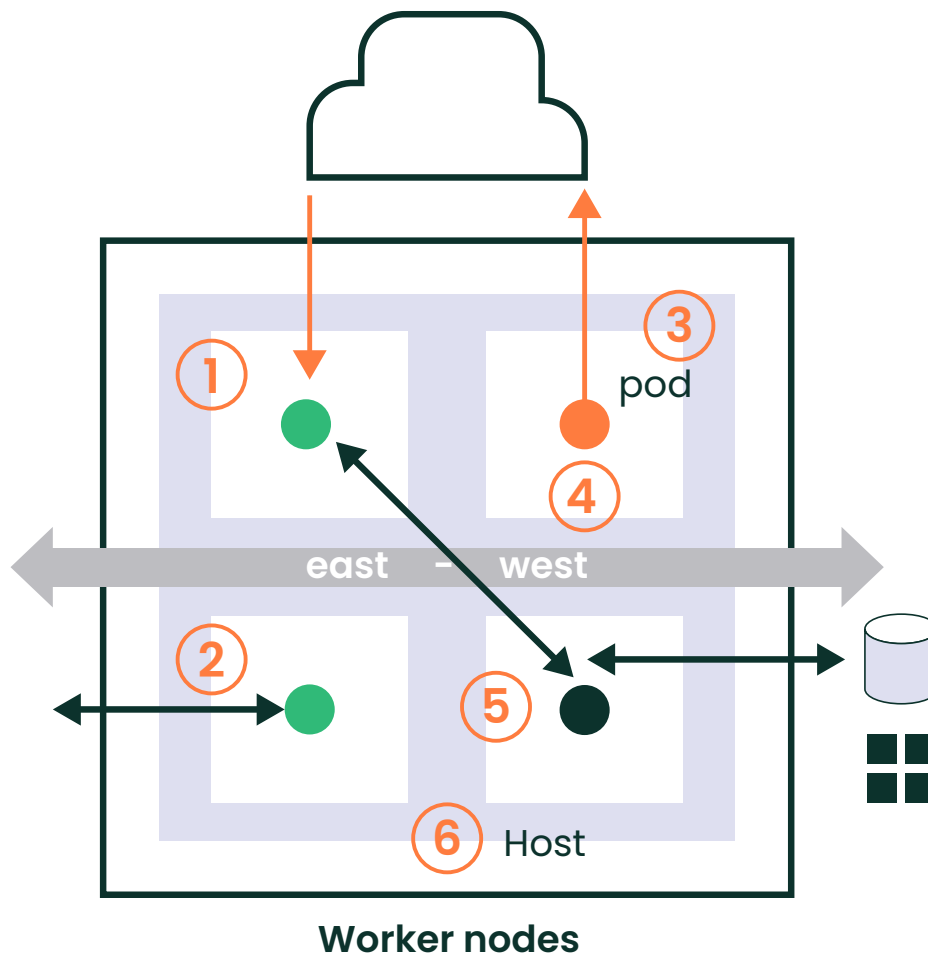
4. **Compromised container running malicious process.** Containers generally have a limited and well-defined set of processes running, but a compromised container can start malware such as crypto mining or suspicious processes such as network port scanning, or inject a binary (process exploit) that has not been seen before.
5. **Container file system compromised.** An attacker can install vulnerable libraries/packages to exploit the container. Sensitive files can also be changed. Once exploited, a privilege escalation to root or other breakout can be attempted.
6. **Compromised worker node.** The host itself can be compromised, the same as any active container. For example, the Dirty Cow Linux kernel vulnerability enabled a user to escalate to root privilege.

The attack kill chain

The most damaging attacks involve a kill chain, or series of malicious activities, which together achieve the attacker's goal. These events can occur rapidly, within a span of seconds, or can occur over days, weeks or even months.

Detecting events in a kill chain requires multiple layers of security monitoring, because different resources are used. The most critical vectors to monitor to have the best chances of detection in a production environment include:
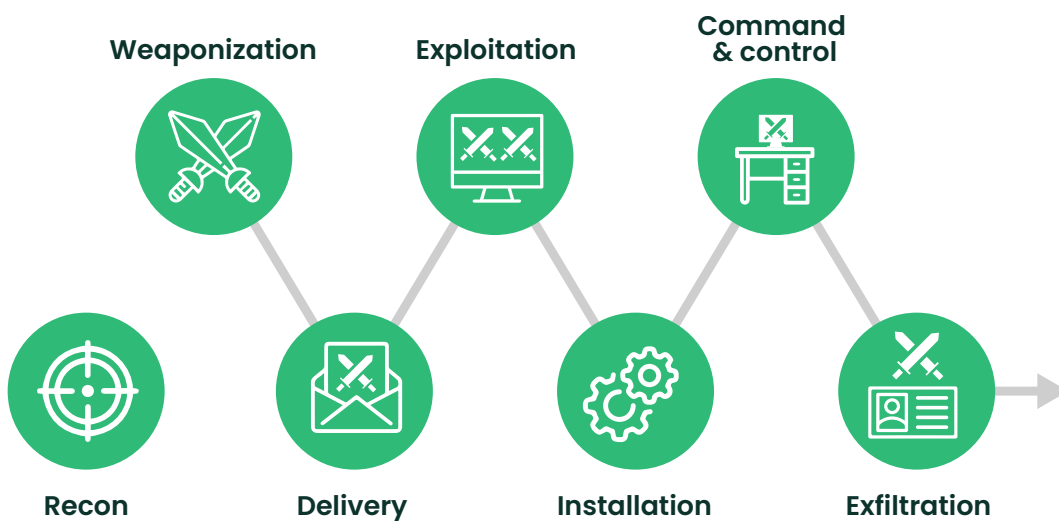
- **Network inspection.** Attackers typically enter through a network connection and expand the attack via the network. The network offers the first opportunity to an attack, subsequent opportunities to detect lateral movement, and the last opportunity to catch data stealing activity.
- **Container.** An application or system exploit can be detected by monitoring the process and syscall activity in each container to determine if a suspicious process has started or attempts are made to escalate privileges and break out of the container. File integrity monitoring and access restrictions can also detect attempts to modify files, packages or libraries.
- **Host monitoring.** This is where traditional host (endpoint) security can be useful to detect exploits against the kernel or system resources. However, host security tools must also be Kubernetes and container-aware to ensure adequate coverage. For example, new hosts can dynamically enter a Kubernetes cluster, and they must maintain the security settings and tools which Kubernetes manages.

In addition to the threat vectors above, attackers can attempt to compromise deployment tools such as the Kubernetes API Server or console to gain access to secrets or be able to take control of running pods.

Attacks on the kubernetes infrastructure itself

To disable or disrupt applications or gain access to secrets, resources, or containers, hackers can also attempt to compromise Kubernetes resources such as the API Server or Kubelets. For example, the Tesla hack exploited an unprotected console to gain access to the underlying infrastructure and run crypto mining software.

When the API Server token is stolen/hacked, or an identity is stolen to enable access to the database by impersonating an authorized user, malicious containers can be deployed or critical applications can be stopped.

By attacking the orchestration tools themselves, hackers can disrupt running applications and even gain control of the underlying resources used to run containers. In Kubernetes there are several published privilege escalation mechanisms, via the Kubelet, access to etcd or service tokens, which can enable an attacker to gain cluster admin privilege rights from a compromised container.

The Kubernetes man-in-the-middle vulnerability, for example, is a relatively new malicious security issue raising concern among security experts, and it won't be the last. The vulnerability enables an attacker to take advantage of Kubernetes built-in service definition with a less-often used option, external IPs, to initiate a man-in-the- middle attack.
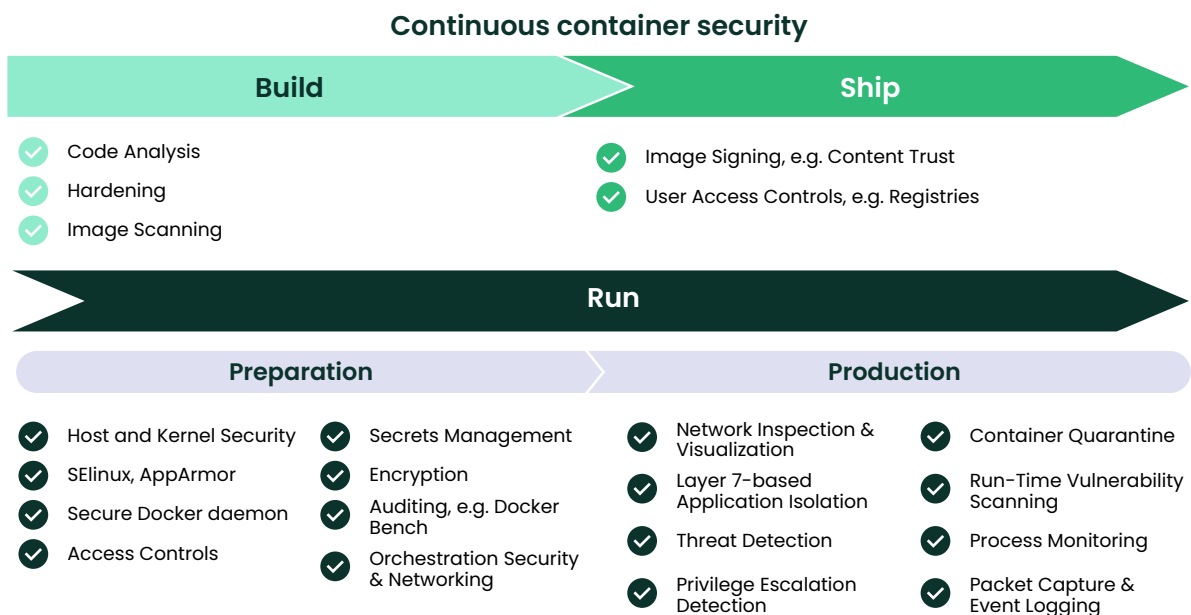
Securing the Entire Pipeline

Before we take a look at how run-time security can protect against man-in-the-middle and other exploits, let's take a step back to see how security can be integrated into the entire CI/CD pipeline.

In the **build phase**, code and image analysis is critical for removing known vulnerabilities and compliance violations before images are approved for deployment.

In the **ship phase**, enabling appropriate access controls and restricting deployment of images is critical to ensuring that vulnerabilities are not intentionally or accidentally introduced later in the pipeline.

In the **run phase**, properly locking down hosts and orchestration tools in preparation is good and required hygiene, but real-time monitoring of the container environment is critical to expose and mitigate new exploits.

Although security teams always wish for the "holy grail" of one tool to provide end-to-end security, there are many layers and steps in the pipeline to secure, and no one tool can accomplish all of it. In general, platforms such as Red Hat OpenShift, Docker EE, Rancher, SUSE CaaS and AWS EKS provide security tools and features that focus on the build, ship, and pre-deployment phases, while independent security vendors can provide end-to-end tools which include run- time security. The run-time security tools must specialize in detecting and preventing sophisticated network and container-based exploits. There are also a few open source container security projects which are summarized later in this guide.

Continuous container security

Build	Ship
✔ Code Analysis	✔ Image Signing, e.g. Content Trust
✔ Hardening	✔ User Access Controls, e.g. Registries
✔ Image Scanning	

Run

Preparation		Production	
✔ Host and Kernel Security	✔ Secrets Management	✔ Network Inspection & Visualization	✔ Container Quarantine
✔ SElinux, AppArmor	✔ Encryption	✔ Layer 7-based Application Isolation	✔ Run-Time Vulnerability Scanning
✔ Secure Docker daemon	✔ Auditing, e.g. Docker Bench	✔ Threat Detection	✔ Process Monitoring
✔ Access Controls	✔ Orchestration Security & Networking	✔ Privilege Escalation Detection	✔ Packet Capture & Event Logging

CI/CD Pipeline Security

Security should be integrated into the CI/CD pipeline as early in the pipeline as possible. For most companies, this begins when a container image is being built by developers. An immediate scan can let developers know if there are critical vulnerabilities or compliance violations which must be fixed before the image can pass through the pipeline to the next phase. There are both technology as well as process issues which should be considered, such as:

- How will scanning be enforced and triggered in the pipeline? Most tools such as Jenkins have plug-ins or extensions which can trigger the scan. Other tools can invoke scripts to trigger scans through APIs.
- What is the approved process for evaluating and remediating vulnerabilities? Who should be notified and review them?
- What criteria should be used to require remediation? Is it based on critical (high CVSS score threshold) severity?
- When should a build job be failed? When a fix is available for a critical vulnerability, but not when no fix is available?
- Should there be a grace period and/or exception (exemption) for violations? What is the process for applying for exceptions?
- Should vulnerabilities discovered later in the pipeline, for example in production registries or in running containers, trigger actions in the pipeline to remediate them?

In addition to building scanning into the CI/CD pipeline, other security measures should include:

- Access controls for pipeline tools and registries to reduce the possibility of insider abuse.
- Admission controls to prevent deployment of vulnerable or unauthorized images, or prevent them from moving further in the pipeline.
- Enforcement of other corporate software management policies such as license controls for open source components and code-scanning tools.

Preparing Kubernetes Nodes for Production

Before deploying application containers, host systems for Kubernetes worker nodes should be locked down. The following section describes the most effective ways to lock down the hosts.

Recommended pre-deployment security steps

- Use namespaces
- Restrict Linux capabilities
- Enable SELinux
- Utilize Seccomp
- Configure Cgroups
- Use R/O Mounts
- Use a minimal Host OS
- Update system patches
- Run CIS Benchmarks security tests

Kubernetes hosts should be continually audited and scanned in staging and production environments to ensure that security configurations are not inadvertently misconfigured during updates and scaling events.

Kubernetes Run-Time Container Security

Once containers are running in production, the three critical security vectors for protecting them are network filtering, container inspection, and host security.

Inspect and secure the network

A container firewall is a new type of network security product which applies traditional network security techniques to the new cloud-native Kubernetes environment. There are different approaches to securing a container network with a firewall, including:

- Layer 3/4 filtering, based on IP addresses and ports. This approach includes Kubernetes network policy to update rules in a dynamic manner, protecting deployments as they change and scale. Simple network segmentation rules are not designed to provide the robust monitoring, logging, and threat detection required for business-critical container deployments, but can provide some protection against unauthorized connections.
- Web application firewall (WAF) attack detection can protect web-facing containers (typically HTTP or HTTPS-based applications) using methods that detect common attacks, similar to the functionality of web application firewalls. However, the protection is limited to external attacks over HTTP and lacks the multi-protocol filtering often needed for internal traffic.

- Layer 7 container firewall. A container firewall with Layer 7 filtering and deep packet inspection of inter-pod traffic secures containers using network application protocols. Container firewalls are also integrated with orchestration tools such as Kubernetes, and utilize behavioral learning for automated policy creation. Protection is based on application protocol whitelists as well as built- in detection of common network-based application attacks such as DDoS, DNS, and SQL injection. Container Firewalls are also in a unique position to incorporate container process monitoring and host security into the threat vectors monitored.

Deep packet inspection (DPI) techniques are essential for in-depth network security in a container firewall. Exploits typically use predictable attack vectors: malicious HTTP requests with a malformed header, or inclusion of an executable shell command within the extensible markup language (XML) object. Layer 7 DPI-based inspection can look for and recognize these methods. Container firewalls using these techniques can determine whether each pod connection should be allowed to go through, or if they are a possible attack which should be blocked.

Given the dynamic nature of containers and the Kubernetes networking model, traditional tools for network visibility, forensics, and analysis can't be used. Simple tasks such as packet captures for debugging applications or investigating security events are not simple any more. New Kubernetes and container-aware tools are needed to perform network security, inspection, and forensic tasks.

Container inspection

Cyber assaults frequently utilize privilege escalations and malicious processes to initiate an attack or spread it. Exploits of vulnerabilities in the Linux kernel (such as Dirty Cow), packages, libraries or applications themselves can result in suspicious activity within a container.

Inspecting container processes and file system activity and detecting suspicious behavior is a critical element of container security. Suspicious processes such as port scanning and reverse shells, or privilege escalations should all be detected. There should be a combination of built-in detection as well as a baseline behavioral learning process which can identify unusual processes based on previous activity.

If containerized applications are designed with microservices principles in mind, where each application in a container has a limited set of functions and the container is built with only the required packages and libraries, detecting suspicious processes and file system activity is much easier and accurate.

Host security

If the host (e.g. Kubernetes worker node) on which containers run is compromised, many types of negative outcomes can result. These include:

- Privilege escalations to root
- Theft of secrets used for secure application or infrastructure access
- Alteration of cluster admin privileges
- Host resource damage or hijacking (e.g. crypto mining software)
- Shutdown of critical orchestration tool infrastructure such as the API Server or the Docker daemon
- Initiation of suspicious processes discussed in the previous section on container inspection

As with containers, the host system needs to be monitored for these suspicious activities. Because containers can run operating systems and applications like the host, monitoring container processes and file systems activity requires the same security functions as monitoring hosts. Together, the combination of network inspection, container inspection, and host security offers the best way to detect a kill chain from multiple vectors.

Securing Kubernetes System and Resources

If not protected, orchestration tools such as Kubernetes and the management platforms built on top of them can be vulnerable to attacks. These expose potentially new attack surfaces for container deployments which previously did not exist, and thus are vulnerable to hacker incursion. The Tesla hack and Kubelet exploit are among the first of what is expected to be a continuing cycle of exploits and patching for new technologies.

To protect Kubernetes and management platforms from attacks, it's critical to properly configure the RBACs for system resources. Following are areas to review and configure for proper access controls:

1. **Protect the api server.** Configure RBAC for the API Server or manually create firewall rules to prevent unauthorized access.
2. **Restrict kubelet permissions.** Configure RBAC for Kubelets and manage certificate rotation to secure the Kubelet.
3. **Require authentication for all external ports.** Review all ports externally accessible and remove unnecessary ports. Require authentication for those external ports needed. For non-authenticated services, restrict access to a whitelist source.
4. **Limit or remove console access.** Don't allow console/proxy access unless properly configured for user login with strong passwords or two-factor authentication.

Generally, all role-based access controls should be carefully reviewed. For example, service accounts with a cluster admin role should be reviewed and restricted to only those requiring it.

When combined with robust host security as discussed previously for locking down worker nodes, the Kubernetes deployment infrastructure can be protected from attacks. However, it is also recommended to use monitoring tools to track access to infrastructure services to detect unauthorized connection attempts and potential attacks.

For example, in the Tesla Kubernetes console exploit, once access to worker nodes was compromised, hackers created an external connection to China to control crypto mining software. Real-time, policy-based monitoring of the containers, hosts, network, and system resources would have detected suspicious processes as well as unauthorized external connections.

Auditing and Compliance for Kubernetes Environments – Security Posture

With the rapid evolution of container technology and tools such as Kubernetes, enterprises will be constantly updating, upgrading, and migrating container environments. Running a set of security tests designed for Kubernetes environments will ensure that security does not regress with each change. In this way the security posture of the infrastructure can be assessed for exploit risks. As more enterprises migrate to containers, changes in infrastructure, tools, and topology may also require recertification for compliance standards such as PCI.

Fortunately, there already are a comprehensive set of security posture checks for Kubernetes and Docker environments through the CIS Benchmarks for Kubernetes and the Docker Bench tests. Regularly running these tests and confirming expected results should be automated.

Some of the areas that these benchmarks test

- Host security
- Kubernetes security
- Docker daemon security
- Container security
- Properly configured RBACs
- Securing data at rest and in transit

In addition, image scanning should include CIS Benchmarks tests which are relevant for image security. Additional image compliance tests also can inspect images for embedded secrets and file access (setuid/setgid) violations.

Vulnerability scanning of images and containers in registries and in production is also a core component for preventing known exploits and achieving compliance. Scanning can be incorporated into the build process and CI/CD pipeline to ensure that all images moving into production have been examined. In production, running containers and hosts should be regularly scanned for vulnerabilities. However, vulnerability scanning is not enough to provide the multiple vectors of security needed to protect container run-time deployments.

Run-time Security Applied NeuVector Container Security Platform

Orchestration and container management tools are not designed to be security tools, even though they provide basic RBACs and infrastructure security features. For business-critical deployments, specialized Kubernetes security tools are needed. Specifically, a security solution that addresses security concerns across the three primary security vectors (network, container and host) is required.

NeuVector is a highly integrated and automated security solution for kubernetes, with the following features:

- Pipeline vulnerability and compliance scanning of images in the build phase and in registries.
- Admission controls to prevent deployment of vulnerable or unauthorized images.
- Multi-vector container security addressing the network, container, and host.
- Layer 7 container firewall to protect east-west and ingress/egress traffic.
- Container protection against unauthorized process and file activity.
- Host security for detecting system exploits.
- Automated policy creation with behavioral learning, and adaptive enforcement to enable auto-scaling.
- Run-time vulnerability scanning for any container or host in the Kubernetes cluster.
- Compliance and auditing through CIS security benchmarks.

The NeuVector solution is a container itself which is deployed and updated with Kubernetes or any orchestration system such as OpenShift, Rancher, Docker EE, IBM Cloud, SUSE CaaS, EKS, etc.
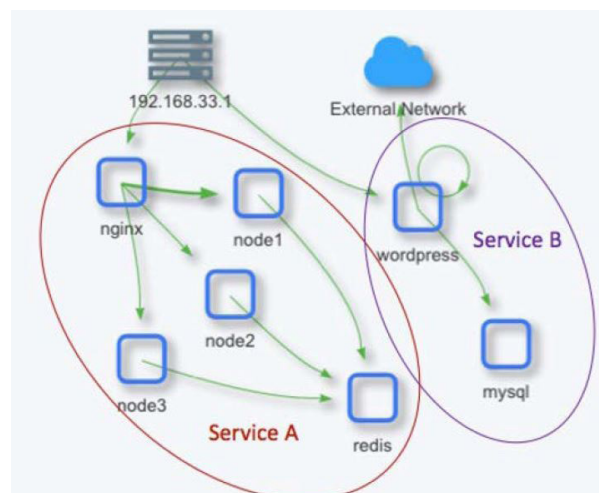
End-to-end vulnerability and compliance management

NeuVector enables shift-left security starting with the build phase in the CI/CD pipeline. Container image builds can trigger a vulnerability scan and fail builds with critical vulnerabilities. Developers can be required to remediate these vulnerabilities before their builds are allowed to pass the build phase and be stored in approved registries. NeuVector supports all of the popular pipeline tools such as Jenkins, CircleCI, Azure DevOps, and Gitlab. A rest API is also available for any other build tool being used.

NeuVector then continuously scans images in approved registries for new vulnerabilities. During image scanning in the build phase or in registries, not only is a layered scan result presented, but additional compliance checks are run for CIS Benchmarks, secrets detected, and file access permission violations.

A vulnerability and compliance explorer provides a powerful tool to analyze results, create compliance reports (for PCI, HIPAA, GDPR, NIST, etc.), and report progress on vulnerability remediation.
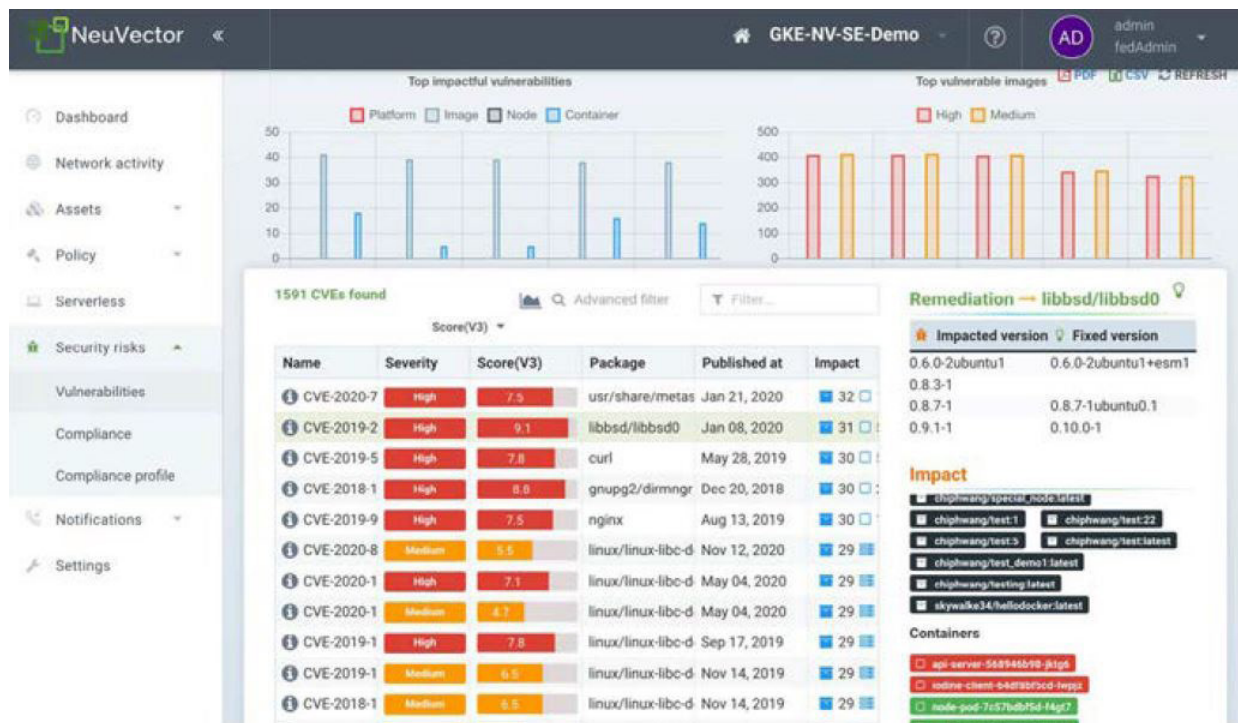
Image scanning results also can be tied to admission control policies to prevent deployment of vulnerable or unauthorized images into the production environment.

Run-time security

After NeuVector is deployed to each worker node, container network connections and service dependencies are easily visualized. The security policies to isolate and protect Kubernetes deployments are automatically created.

In real time, the NeuVector container starts inspecting network traffic and monitoring containers and hosts for suspicious activity. Following are a few examples of how NeuVector provides protection against multiple attack vectors in a Kubernetes deployment.

Network isolation, segmentation and threat detection

Running pods, their network connections and security policy to protect them are automatically discovered and visualized. Each application stack is isolated through whitelist rules which are automatically deployed.
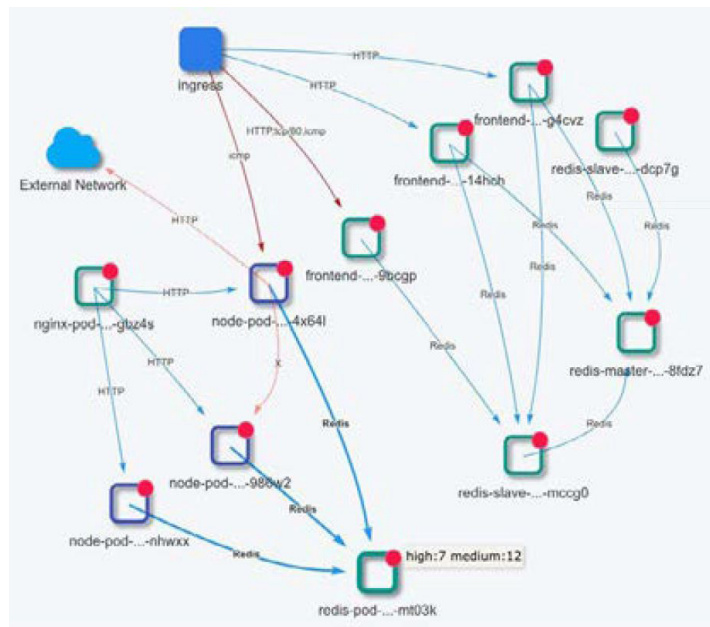
Attacks against containers, whether they originate externally or internally, are detected and can be blocked. The NeuVector Layer 7 firewall can run in a monitor (network tap) mode or a protect (inline) mode where attacks or unauthorized connections can be blocked while keeping the container active for valid traffic. Any security incidents are summarized in the network activity console.

Packet capture is automated and simplified for Kubernetes pods, enabling forensics, logging, and application debugging.
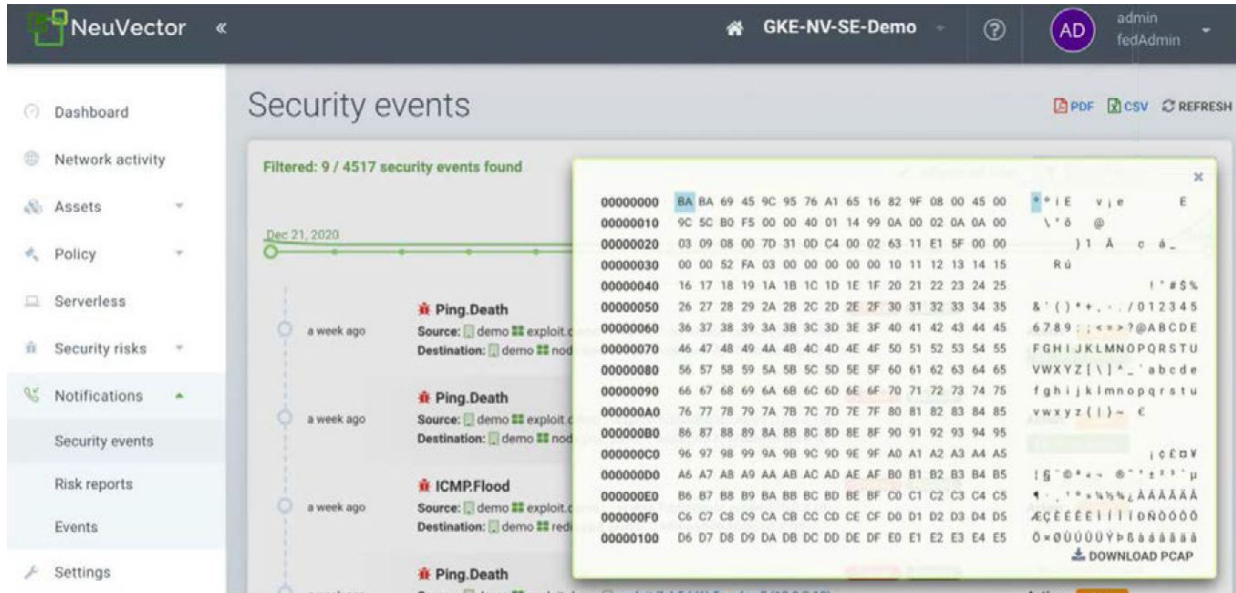
Container compromise detection

Unusual activity is detected in any container, with built-in detection for suspicious processes such as port scanning and reverse shells. In addition, running processes in each container are baselined to aid in the detection of unauthorized or malicious processes.

The container file system also is monitored for suspicious activity. For example, if a package or library is installed or updated, a vulnerability scan is automatically triggered and an alert is generated.

Host compromise detection

Host systems are monitored for exploits such as privilege escalations. Suspicious processes detected in containers are also detected running on hosts. For example, if port scanning or reverse shell processes start running, NeuVector will detect and alert. In addition, NeuVector can learn and whitelist allowed processes to run on the host, and block any unauthorized host processes which attempt to start.

Monitoring of system containers
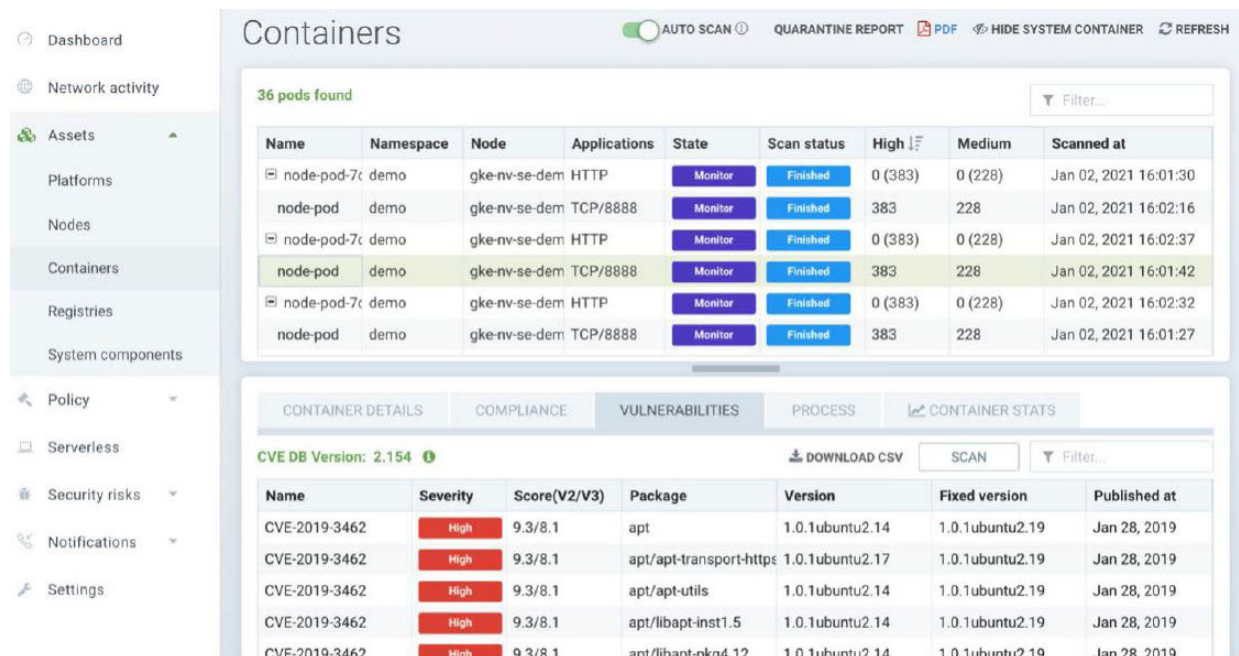
NeuVector also monitors system containers and network activity for each container. In the diagram below, the Kubernetes and OpenShift containers are shown with their network connections.
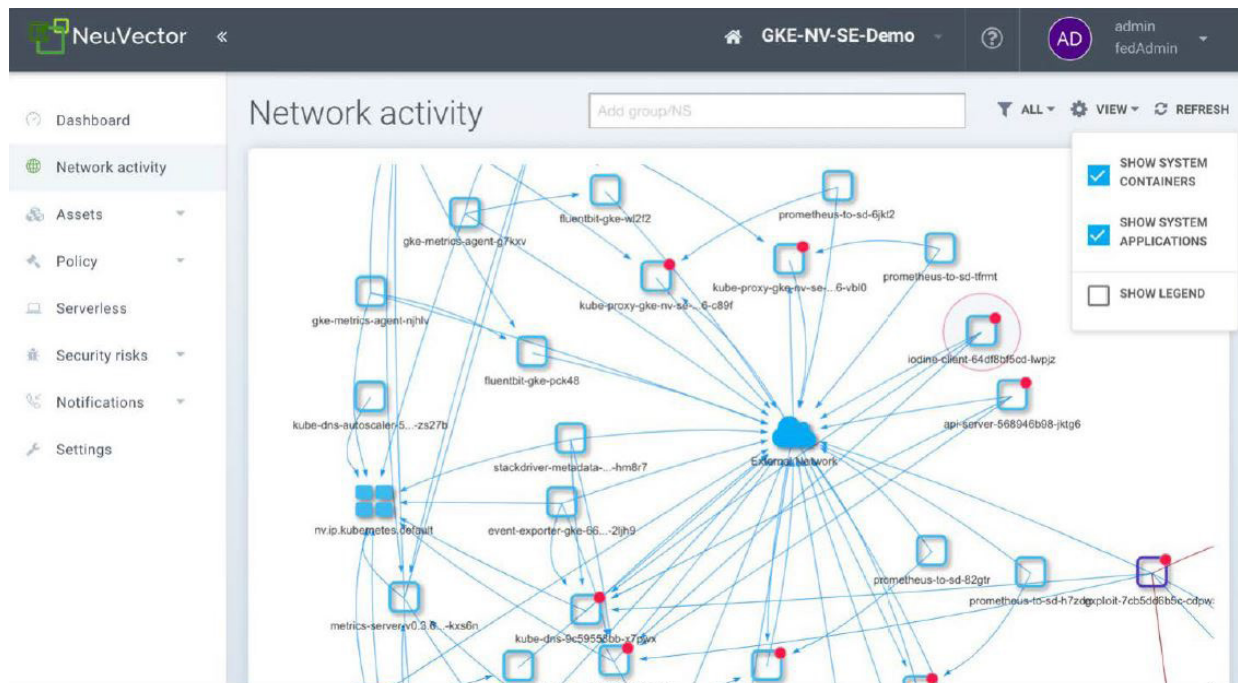
Suspicious activity to or from system containers can be easily detected.

Run-time scanning - compliance and auditing

NeuVector automatically scans running pods, containers and worker nodes for vulnerabilities, and runs the Kubernetes CIS Benchmarks tests on every node. System containers and the orchestration platform (e.g. Kubernetes 1.19) are also scanned for vulnerabilities.

Vulnerability scanning also can be performed by NeuVector in the build and ship phases to inspect image registries or during the CI/CD automated pipeline. Jenkins integration is provided to enable scanning during the image build process.

Kubernetes Security Automation – Is It Possible?

With DevOps teams rushing to automate application deployment with containers and Kubernetes, security automation is critical. Gone are the days when security teams can stop or slow deployments and updates of applications, infrastructures, or even new clouds.

Security automation starts with creating secure infrastructures and platforms for running containers, followed by automated run-time security. A secure infrastructure with repeatable secure configurations can be ensured by using infrastructure as code concepts and tools such as Terraform.

The good news is that most run-time security can be automated using a combination of behavioral learning and Kubernetes integration with custom resource definitions (CRDs) to implement security as code. There may always be some initial manual setup or customization required, but when the production switch is turned on and Kubernetes starts managing pods, your security should automate, adapt, and scale with the deployment.

New security tools such as NeuVector can provide multi-vector run-time security by fitting into the Kubernetes deployment model. By combining a Kubernetes container firewall with container inspection and host security, the activities in a kill chain for a damaging attack can be detected and prevented. The modern cloud-native architecture of NeuVector means it easily deploys as its own container, sitting next to your application containers, providing always-on, always running security.

Because of the declarative nature of container-based applications and the extensive integration available for tools like Kubernetes, advanced security controls can be enforced even in the hyper-dynamic container environment. By integrating with Kubernetes and incorporating behavioral learning and multi-vector security features such as those from NeuVector, security automation is now possible throughout the pipeline, and is a strong requirement for business-critical Kubernetes deployments.
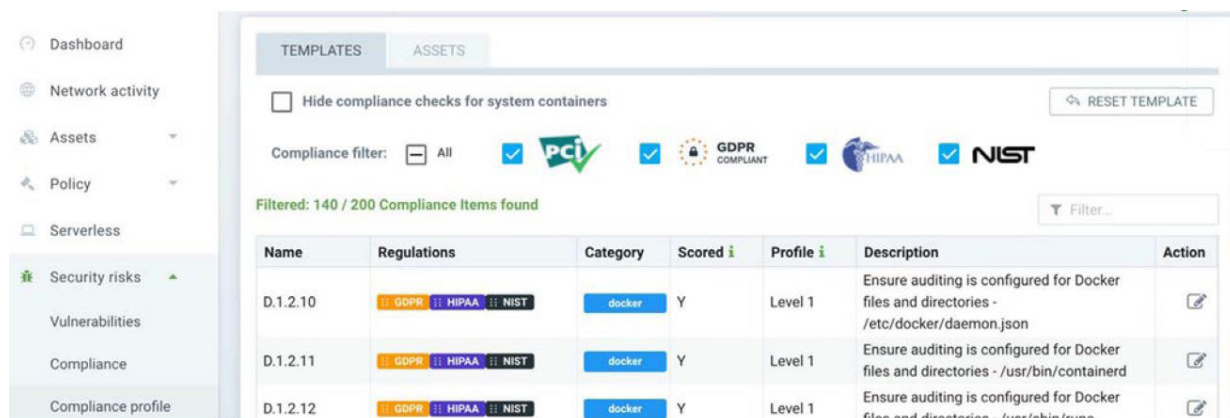
Achieving PCI, GDPR, SOC 2, HIPAA, NIST Compliance

Achieving compliance with industry standards can be difficult for container infrastructures. The new virtualized layers have not been scrutinized by auditors and compliance consultants and a consensus developed for compliance requirements.

NeuVector can help achieve compliance with standards such as PCI, GDPR, SOC 2, HIPAA, and NIST which require specific security practices and capabilities. These standards specify security practices which can include:

- **Network Segmentation and Firewalling.** With the Layer 7 container firewall designed for container and Kubernetes network filtering and protection, NeuVector is in a unique position to fill this requirement.
- **Vulnerability Scanning and Remediation.** End-to-end vulnerability management enables NeuVector to enforce vulnerability management policies from the build phase all the way into production.
- **Auditing Configuration Tests.** Reviewing and enforcing proper configuration of systems (hosts), orchestrators, and containers through compliance checks such as the Kubernetes CIS Benchmarks reduces the risk of misconfiguration leading to a breach.
- **Restricted Access Controls.** Evaluating and granting the least required user access privileges limits the likelihood of RBAC-based exploits and insider attacks.
- **Encryption and Sensitive Data Protection.** NeuVector can ensure that connections are encrypted for data in motion and can even use DLP technology to monitor for sensitive data leaks such as social security numbers, credit cards, and other PII.

NeuVector provides pre-configured, customizable reports for PCI, GDPR, HIPAA, and NIST compliance reports for container deployments.

With the combination of customizable compliance reports, end-to-end vulnerability management, firewalling and network segmentation, and compliance tests, NeuVector has helped companies achieve compliance for new cloud-native infrastructures and workloads. This extends beyond PCI and GDPR to standards such as service organizational control (SOC) 2.

SOC 2 is an auditing procedure that ensures service providers (applications) securely manage data to protect the interests of an organization and the privacy of its clients. For security-conscious businesses, SOC 2 compliance is a minimal requirement when considering a SaaS provider. SOC 2 compliance is mandatory for all engaged, technology-based service organizations that store client information in the cloud. Such businesses include those that provide SaaS and other cloud services while also using the cloud to store each respective, engaged client's information.

NeuVector addresses all key requirements of SOC 2 to enable organizations to exceed the SOC 2 standard. To maintain security and achieve compliance, you need the extra layer of protection that NeuVector provides. Financial services companies, and those in other highly regulated industries, turn to NeuVector for the Kubernetes data loss prevention capabilities required to keep customer information secure and businesses audit-ready at all times.

Open Source Kubernetes Security Tools

While commercial tools such as the NeuVector container security platform offer end-to-end protection and visibility, there are open source projects which continue to evolve to add security features. Following are several tools to be considered for projects which are not as business critical in production.

- **Network policy.** Kubernetes network security policy provides automated L3/ L4 (IP address/port based) segmentation. A network plug-in is required which supports enforcement of network policy such as Calico.
- **Service meshes/ISTIO.** Istio is an example of a service mesh for managing service-to-service communication, including routing, authentication, authorization, and encryption. Istio provides a solid framework for managing service routing, but is not designed to be a security tool to detect attacks, threats, and suspicious container events.
- **Grafeas.** Grafeas provides a tool to define a uniform way for auditing and governing the modern software supply chain. Policies can be tracked and enforced with integration to third-party tools. Grafeas can be useful in governing the CI/CD pipeline, but is not targeted to managing run-time security policies.
- **Clair.** Clair is a simple tool for vulnerability scanning of images, but lacks registry integration and workflow support.

- **Kubernetes CIS benchmark.** CIS Benchmarks for Kubernetes security offer more than 100 compliance and auditing checks. The NeuVector implementation of these tests is available [here](#).
- **Open policy agent (OPA).** [OPA](#) provides a framework for managing and enforcing security policies. A special query language is supported to manage disparate security policies across the enterprise, with limited enforcement through Kubernetes admission control.

NeuVector is a commercial container security solution which is compatible with these open source projects, and offers advanced security features designed to protect financial, compliance regulated, and other business-critical container deployments.

Summary Checklist for Run-Time Kubernetes Security

Following is a convenient checklist summary of the security protections to review for securing Kubernetes deployments in the CI/CD pipeline and during run-time.

CI/CD pipeline

- Scan images in build phase and fail/reject those with critical vulnerabilities with fixes available, or compliance violations.
- Continuously scan approved images in registries to alert if new vulnerabilities have been discovered.
- Scan images for compliance violations of CIS Benchmarks as well as embedded secrets, running as root, file permissions and other security issues.
- Implement declarative security as code practices to involve developers and/or DevOps teams to create or review allowed (whitelisted) application behavior for their application workloads.

Pre-production checklist for devops and security teams

- Use namespaces
- Restrict Linux capabilities
- Enable SELinux
- Utilize Seccomp
- Configure Cgroups
- Use R/O mounts

- Use a minimal host OS
- Update system patches
- Conduct security auditing and compliance checks with CIS Benchmarks tests

Run-time checklist for operations and security teams

- Prevent unauthorized and vulnerable deployments by using admission control policies
- Enforce isolation by application/service using network segmentation and namespaces
- Inspect network connections for application attacks
- Monitor containers for suspicious process or file system activity
- Protect worker nodes from host privilege escalations, suspicious processes or file system activity
- Capture packets for security events
- Quarantine or remediate compromised containers
- Scan containers and hosts for vulnerabilities
- Alert, log, and respond in real-time to security incidents
- Conduct security auditing and compliance checks with CIS Benchmarks

Kubernetes system protections

- Review all RBACs
- Protect the API Server
- Restrict Kubelet permissions
- Secure external ports
- Whitelist non-authenticated services
- Limit/restrict console access
- Monitor system container connections and processes in production
- Run the CIS Benchmarks for Kubernetes to audit configurations
- Keep the orchestrator version updated to remediate critical vulnerabilities

Next Steps
Want to learn more?

Visit NeuVector.com for additional container security articles on our blog or to schedule a demo of the NeuVector Kubernetes security platform.

NeuVector
BY SUSE

SUSE Software Solutions
Germany GmbH

Frankenstraße 146
90461 Nürnberg
Germany

www.suse.com

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

+49 (0)911-740 53-0 (Worldwide)

Innovate
Everywhere