The Ultimate Guide to Kubernetes Security

How to Deploy Kubernetes with **Confidence**

NeuVector

Table of Contents

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. Attacks for ransomware, crypto mining, data stealing and service disruption will continue to be launched against new container based virtualized environments in both private and public clouds.

To make matters worse, new tools and technologies like Docker and Kubernetes will themselves be under attack in order to find ways into an enterprise's prized assets. The recent Kubernetes **exploit at Tesla** is just the first of many container technology based exploits we will see in the coming months and years.

THE HYPER-DYNAMIC NATURE OF CONTAINERS CREATES THE FOLLOWING SECURITY CHALLENGES:

- **EXPLOSION OF EAST-WEST TRAFFIC**. While monolithic applications could be secured by traditional firewalls and host security tools, containers can be dynamically deployed across hosts or even clouds, dramatically increasing the **east-west**, or internal, traffic that must be monitored for attacks.
- **INCREASED ATTACK SURFACE.** Each container may have a different attack surface and vulnerabilities which can be exploited. In addition, the additional

attack surface introduced by container orchestration tools such as Kubernetes and Docker must be considered.

- **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. New application behaviors which can include new network connections must be instantly factored into enforced security policies. A new generation of 'automated' security tools are needed to secure containers.

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 attackers 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 practicality this is not possible, and even if it were, you'd still want to do real-time monitoring for attacks. As time has shown over and over, the sophistication of attackers always matches or out-preforms new infrastructure approaches. Bad actors are constantly developing new and novel ways to attack containers.

HERE ARE SECURITY RELATED QUESTIONS TO ASK YOUR KUBERNETES TEAM:

- Do you have visibility of Kubernetes pods being deployed? For example, how the application pods or clusters are communicating with each-others?
- Do you have a way to detect bad behavior in east/west traffic between containers?
- How can we tell if every individual pod behaving 'normally'?
- How are you being noticed or alerted when some internal service pods or containers start to scan ports internally or try to connect to the external network randomly?
- Are you familiar with the potential attack vectors in a Kubernetes based deployment?

- How would 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, and container hosts?
- When you have compliance policies, how do you enforce the compliance at run-time? For example, to ensure the encryption for your internal pod communication? How do you know when there's a pod is not following the encryption channel?
- If your containers are digital signed and scanned at registry, how do you ensure the same image is locked down at run-time? Make sure nobody opened it up or patch it or modify its running instances.
- When troubleshooting the application communication or record forensic data, how do you locate the problem pod and capture its logs and maybe even capture the raw traffic to analysis quickly before it's disappear?

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 intro into the key concepts and terms.

Kubernetes is a container orchestration tool which automates the deployment, update, 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. Here are some of the key things 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 a 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 attacks. In fact, the recent **Tesla exploit** attacked an unprotected Kubernetes console access to install crypto mining software.

KUBERNETES ROLE-BASED ACCESS CONTROLS

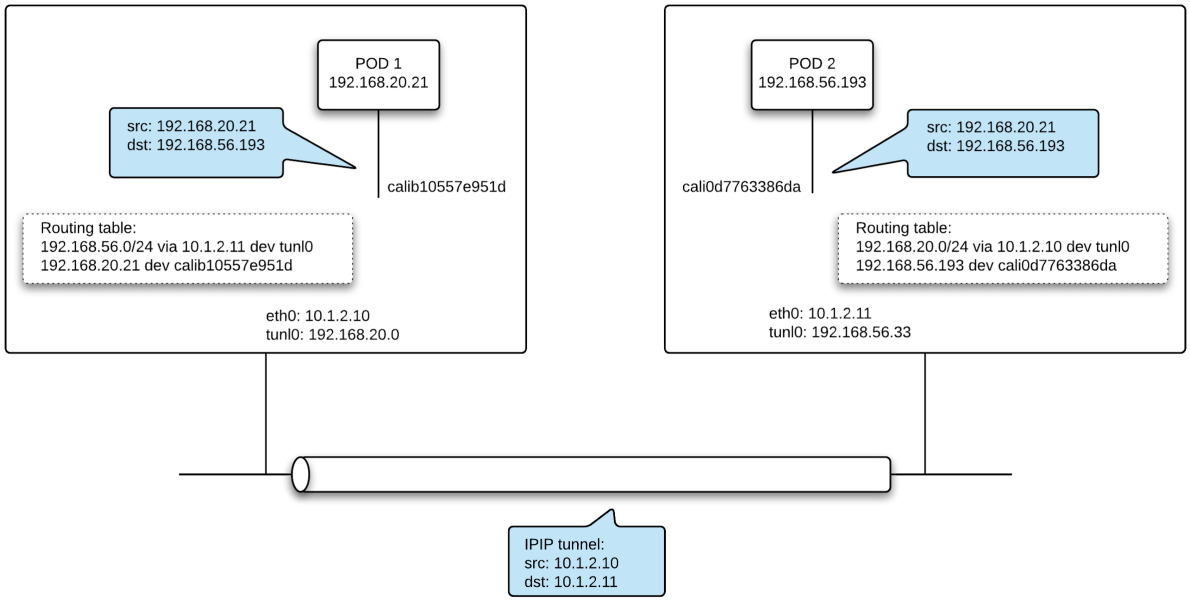
Kubernetes **role-based access controls** (RBACs) provide granular control of resources. These can control access to application workloads as well as Kubernetes system resources. Management tools such as OpenShift may add additional capabilities but rely on or use Kubernetes basic controls underneath.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

KUBERNETES NETWORKING BASICS

The key networking concept in Kubernetes is that every pod has 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. Here 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 plugin is used. Every network plugin 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 plugin 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, 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 plugin.

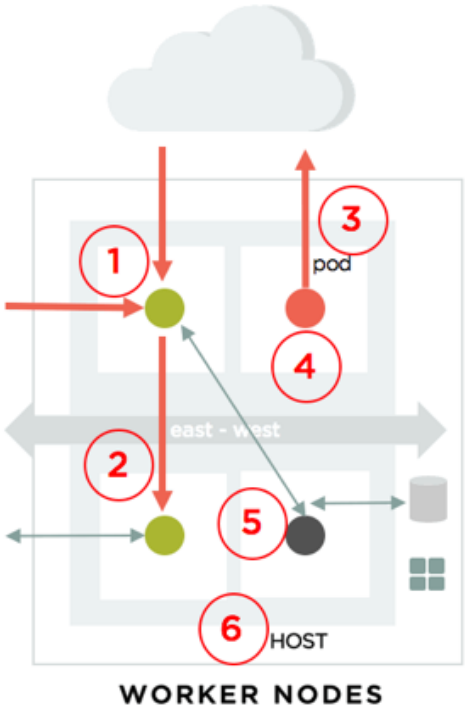
    c. In the above diagram, the packet is sent to the IPIP tunnel interface and encapsulated with an IPIP tunnel header.
5. When the packet reaches the destination node, the encapsulation is removed.
6. The routing table on the remote node routes the packets to the destination Pod, Pod2.

With all 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/control server and network tunneling to hide confidential data.
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 like network port scanning. Inject a binary (process exploit) that has not been seen before.

5. **CONTAINER FILE SYSTEM COMPROMISED.** An attacker can install vulnerable libraries/packages in order to exploit the container. Sensitive files can also be changed. Once exploited, a privilege escalation to root or other break out can be attempted.
6. **COMPROMISED WORKER NODE.** The host itself can be compromised, just as any container running on it can be. 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 attackers goal. These events can occur rapidly, within a span of seconds, or can be spread out 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 in order 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 have started or attempts are being made to escalate privileges and break out of the container.

- **HOST MONITORING.** Here 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 these must maintain the security settings and tools which Kubernetes manages.

In addition to the vectors above, attackers can also 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

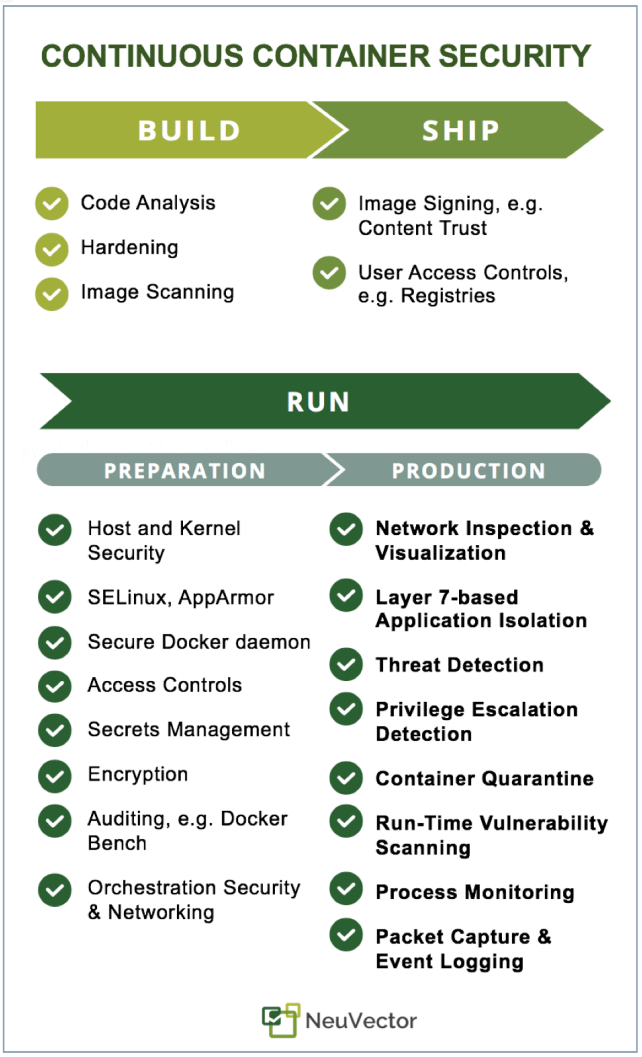
In order to disable or disrupt applications or gain access to secrets, resources, or containers, hackers can also attempt to compromise Kubernetes resources such as the API Server or Kubelets. The recent Tesla hack exploited an unprotected console to gain access to the underlying infrastructure and run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

One example is when the API Server token is stolen/hacked, or identity is stolen to be able to access the database by impersonating the authorized user can deploy malicious containers or stop critical applications from running.

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 have been some 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.

Securing the Entire Pipeline

Before we take a look at run-time security for Kubernetes, 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 before applications are deployed.

In the Ship phase, ensuring appropriate access controls and restricting deployment of images is critical to make sure 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 required hygiene, but real-time monitoring of the container environment is critical in order to catch 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 for the Build, Ship, and Pre-Deployment phases, while independent security vendors provide run-

time security specializing in network and host-based exploits. There are also a few open source container security projects which are summarized later in this document.

PREPARING KUBERNETES WORKER NODES FOR PRODUCTION

Before deploying any application containers, the host systems for the Kubernetes worker nodes should be locked down. Here are 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 Benchmark security tests

Kubernetes Real-Time, 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 based applications) using methods that detect common attacks, similar to the functionality 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 also are 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

Attacks frequently utilize privilege escalations and malicious processes to carry out 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, all kinds of bad things can happen. These include:

- Privilege escalations to root
- Stealing of secrets used for secure application or infrastructure access
- Changing of cluster admin privileges
- Host resource damage or hijacking (e.g. crypto mining software)
- Stopping of critical orchestration tool infrastructure such as the API Server or the Docker daemon
- Starting of suspicious processes mentioned in the Container Inspection section above

Like containers, the host system needs to be monitored for these suspicious activities. Because containers can run operating systems and applications just 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 offer the best way to detect a kill chain from multiple vectors.

Securing Kubernetes System and Resources

Orchestration tools such as Kubernetes and the management platforms built on top of it can be vulnerable to attacks if not protected. These expose potentially new attack surfaces for container deployments which previously did not exist, and thus will be attempted to be exploited by hackers. The recent **Tesla hack** and **Kubelet exploit** are just the start of the continuing cycle of exploit/patch that can be expected for new technologies.

In order to protect Kubernetes and management platforms themselves from attacks it's critical to properly configure the RBACs for system resources. Here are the 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 before for locking down the worker nodes, the Kubernetes deployment infrastructure can be protected from attacks. However, it is also recommended that monitoring tools should be used 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

With the rapid evolution of container technology and tools such as Kubernetes, enterprises will be constantly updating, upgrading, and migrating the container environment. Running a set of security tests designed for Kubernetes environments will ensure that security does not regress with each change. As more enterprises migrate to containers, the changes in the infrastructure, tools, and topology may also require re-certification for compliance standards like PCI.

Fortunately, there are already a comprehensive set of security 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.

HERE IS A LIST OF 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

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 built into the build process and CI/CD pipeline to ensure that all images moving into production have been scanned. In production, running containers and hosts should be regularly scanned for vulnerabilities. But, vulnerability scanning is not enough to provide the multiple vectors of security needed to protect container runtime deployments.

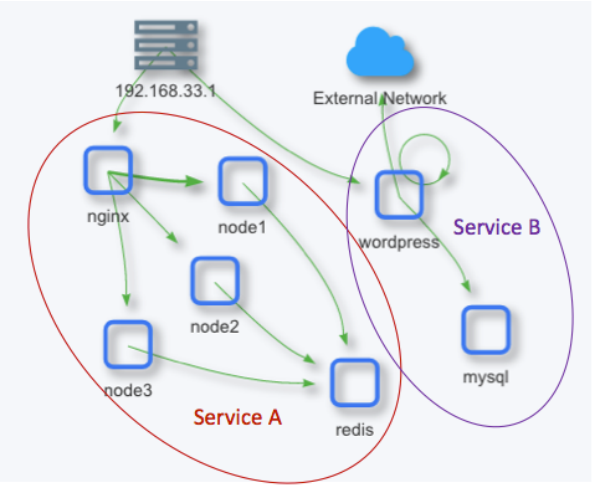
Run-Time Security Applied

NEUVECTOR MULTI-VECTOR CONTAINER SECURITY

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; and 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, AUTOMATED SECURITY SOLUTION FOR KUBERNETES, WITH THE FOLLOWING FEATURES:

- Multi-vector container security addressing the network, container, and host.
- Layer 7 container firewall to protect east-west and ingress/egress traffic.
- Container inspection for suspicious activity.
- Host security for detecting system exploits.
- Automated policy and adaptive enforcement to auto-protect and auto-scale.
- Run-time vulnerability scanning for any container or host in the Kubernetes cluster.
- Compliance and auditing through CIS security benchmarks.



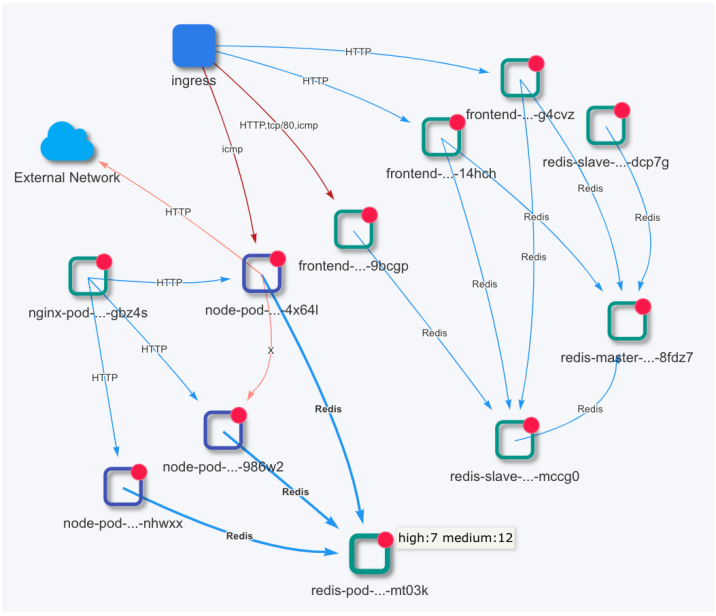
The NeuVector solution is a container itself which is deployed and updated with Kubernetes or any orchestration system you use such as OpenShift,

Rancher, Docker EE, IBM Cloud, SUSE CaaS, EKS etc.

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. Here are a few examples of how NeuVector provides protection against multiple attack vectors in a Kubernetes deployment.

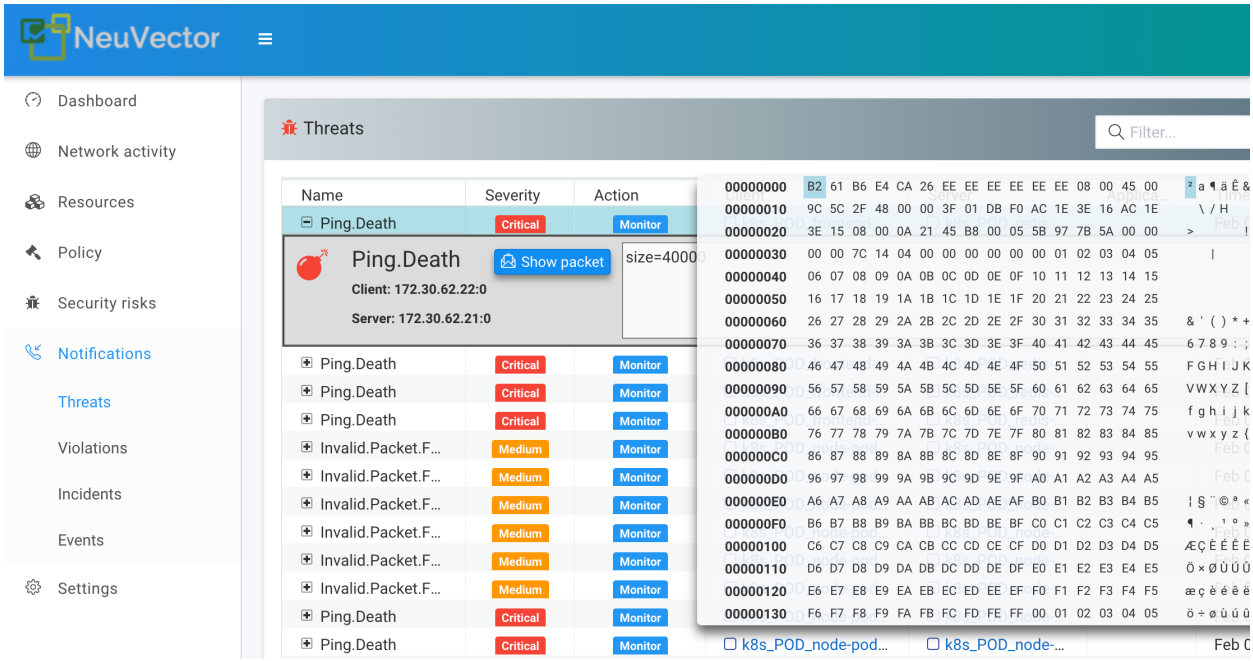
NETWORK ISOLATION, SEGMENTATION AND THREAT DETECTION



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.

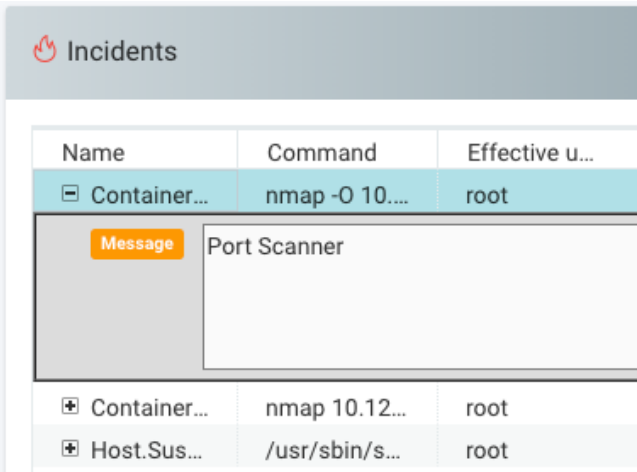
Attacks against containers, whether they originate externally or internally, are detected and can be blocked. The NeuVector Layer 7 firewall can run in a monitor (network tap) mode or a protect (inline) mode where attacks or unauthorized connections can be blocked while keeping the container up for valid 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

Suspicious 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 unusual or malicious processes.



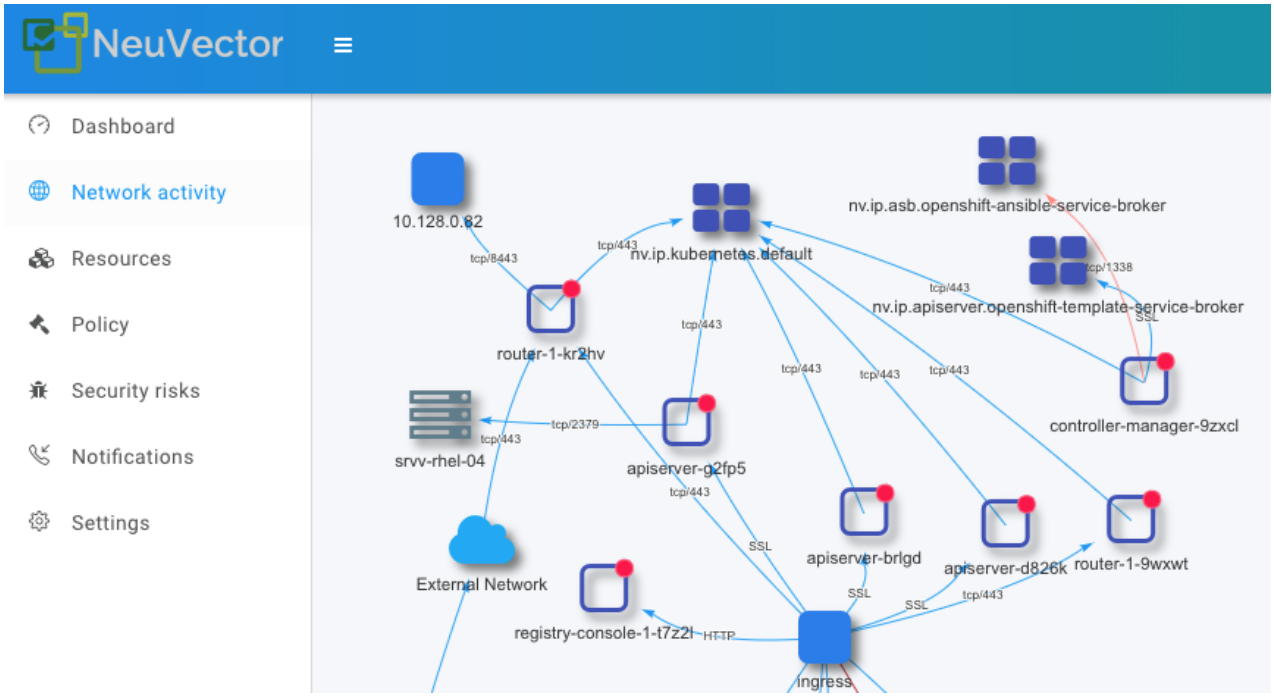
The container file system is also monitored for suspicious activity. For example, if a package or library is installed or updated, a vulnerability scan is automatically triggered and an alert is generated.

HOST COMPROMISE DETECTION

Host systems are also 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.

MONITORING OF SYSTEM CONTAINERS

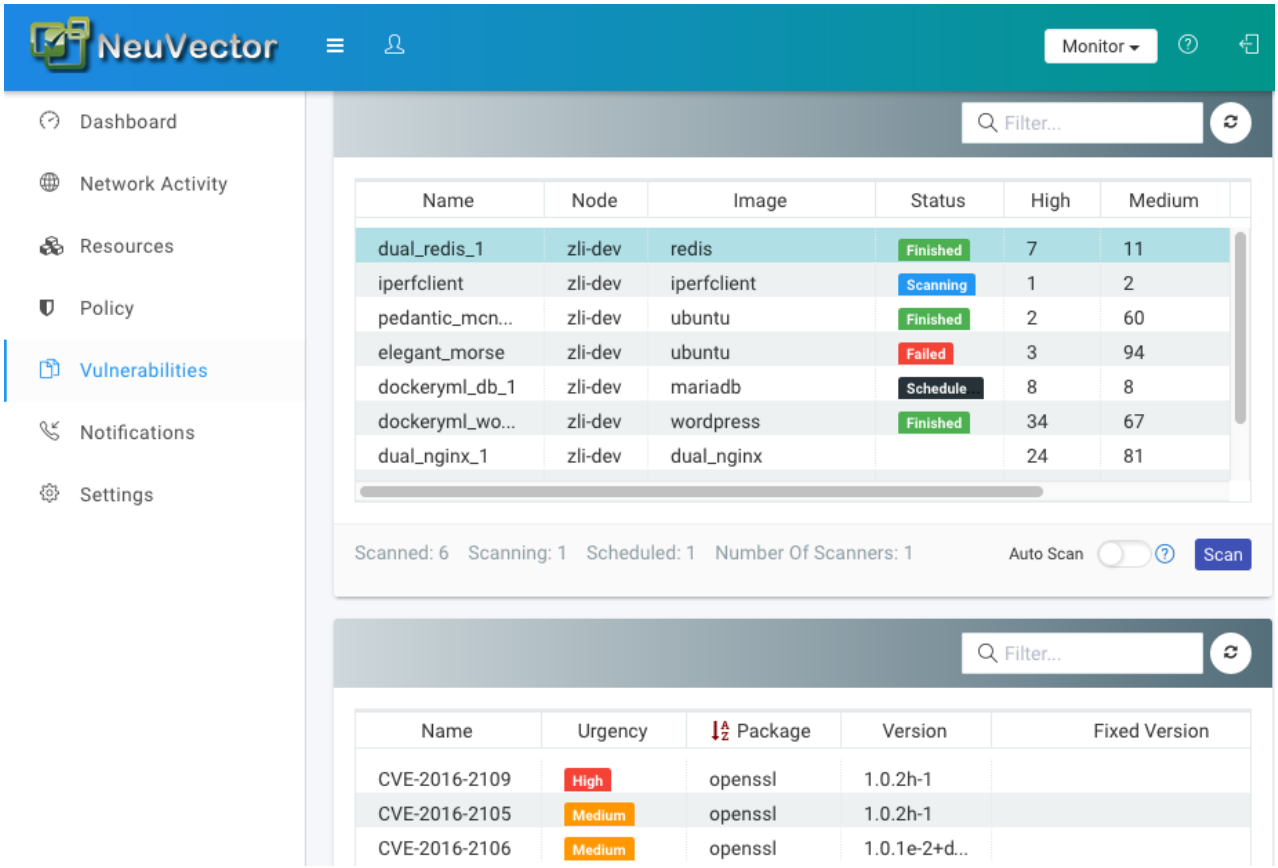
NeuVector also monitors system containers and the network activity for each. In the diagram below, the Kubernetes and OpenShift containers are shown with their network connections.



Suspicious activity to or from system containers can be easily detected.

SCANNING - COMPLIANCE AND AUDIT

NeuVector automatically scans running pods/containers and the worker nodes for vulnerabilities, and runs the Kubernetes CIS benchmark tests on every node. System containers are also scanned for vulnerabilities.



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

For a private demo of NeuVector or to arrange for a **free trial**, please contact NeuVector at **info@neuvector.com** or **https://neuvector.com**.

Kubernetes Security Automation - *Is It Possible?*

With DevOps teams rushing to automate application deployment with containers and Kubernetes, it's critical that security also be as automated as possible. Gone are the days when security teams can stop or slow deployments and updates of applications, infrastructures, or even new clouds. The good news is that much of run-time security can be automated using a combination of behavioral learning and Kubernetes integration. 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 like 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 are able to 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, and is a strong requirement, for business-critical Kubernetes deployments.

Open Source Kubernetes Security Tools

While commercial tools like the NeuVector container firewall offer multi-vector protection and visibility, there are open source projects which continue to evolve to add security features. Here are some of them to be considered for projects which are not as business critical in production.

- **NETWORK POLICY.** Kubernetes Network 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.
- **ISTIO.** Istio creates 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.
- **GRAFAES**. 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**. The compliance and auditing checks from the CIS Benchmark for Kubernetes Security are available to use. The NeuVector implementation of these 100+ tests is available **here**.

NeuVector is a commercial run-time 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

Here is a convenient checklist summary of the security protections to review for securing Kubernetes deployments during run-time. This list does not cover the build phase vulnerability scanning and registry protection requirements.

PRE-PRODUCTION

- ❑ 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 benchmark tests

RUN-TIME

- ❑ Enforce isolation by application / service
- ❑ 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 & hosts for vulnerabilities
- ❑ Alert, log, and respond in real-time to security incidents
- ❑ Conduct security auditing and compliance checks with CIS benchmark tests

KUBERNETES SYSTEM

- ❑ 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

Next Steps

WANT TO LEARN MORE?

Contact NeuVector at https://neuvector.com for more container security articles on our blog and to schedule a demo of the NeuVector Multi-Vector Container Firewall.