Guide

10 Steps to Automate Container Security Into the CI/CD Pipeline

10 Steps to Automate Container Security Into the CI/CD Pipeline

How to integrate security into the kubernetes pipeline

Introduction

Enforcing security and compliance requirements in modern cloud-native pipelines can be a challenge. The increased attack surface of container infrastructures makes security even more important, but security and DevOps teams can't afford to slow the pipeline with manual processes.

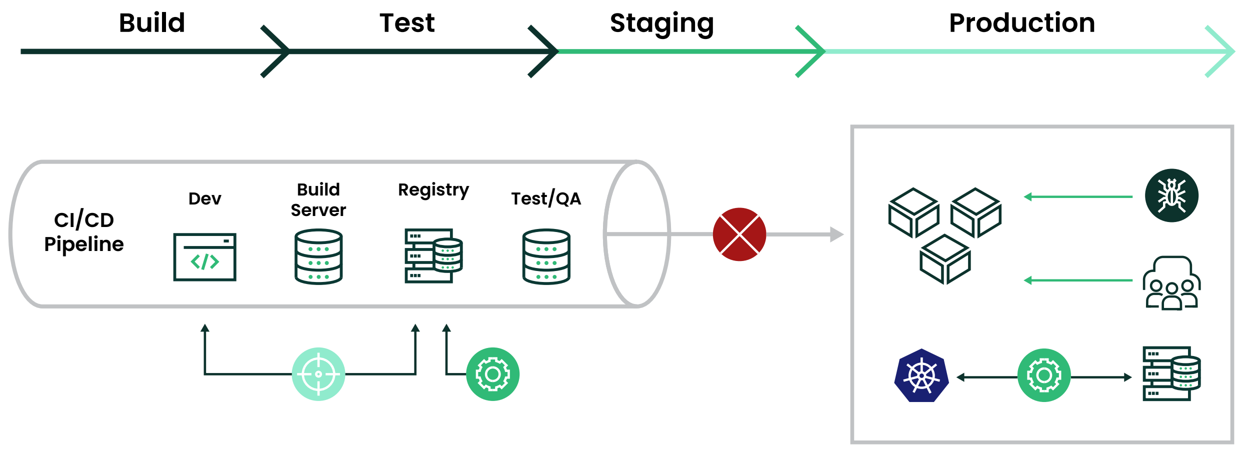
It takes a broad array of cloud-native security capabilities to successfully integrate security into the CI/ CD pipeline, including into the production environment. To implement a layered security approach for containers, start from the 'left-side,' of development with vulnerability and compliance scanning. Then progress to the right with real-time network, container, and host protections.

Vulnerability scanning without virtual patching will overwhelm teams with non-actionable data, and run- time security without vulnerability and compliance management in the pipeline will increase the risk of damaging attacks penetrating the production environment.

This guide discusses security issues facing DevOps and security teams and identifies ten integration points where container security can be automated into the pipeline.

The CI/CD Pipeline Attack Surface

Before we look at automating container security, let's understand why security is so critical to container and Kubernetes pipelines. From the beginning of the CI/CD pipeline into production, there are attack points where security can be comprised. The diagram below provides a summary of the most important critical security issues which keep DevOps and security teams awake at night.



Crical vulnerabilities
Can enter Pipeline at Any Stage

Misconfiguration
Registries, Kubernetes, Container Hosts

Inadequate protection in production
Container Exploits, Zero day, Insiders

Security breaks pipeline
Manual Configuration, Lack of Integration

New attack surfaces
Kubernetes, Docker, Tool Vilnerabilities

Properly addressing these security concerns should be the primary focus for integrating security into the pipeline

Critical vulnerabilities

Vulnerabilities can be introduced in the build phase, registry, and even into the production environment. Not all vulnerabilities have fixes available, and vulnerabilities can be discovered even after passing build phase scanning, in images in registries or containers already deployed.

Security breaks pipeline

Adding manual configuration steps for security reasons or introducing process checkpoints for security reviews can slow or even stop the continuous integration and release of new or updated applications. This will reduce the business benefits realized from a CI/CD pipeline. Security teams can't afford to be holding the business back from achieving their goals.

Misconfiguration

Cloud-native tools including development, registries, and Kubernetes are both new and complex, leading to misconfigurations which compromise security. While education and training are a solution, continuous auditing of configurations should be conducted to be sure a securely configured environment one day is not open to attacks after the next update.

New attack surfaces

The critical infrastructure such as the container orchestrator Kubernetes, run-time CRI-O or containerd, and supporting infrastructure such as registries are all attack surfaces. Hackers are only now seeing the explosion of containers in production as ripe targets to launch zero-day attacks. The rapid evolution and lack of widespread operational experience of the cloud-native tools means they will not become battle- tested and attack resistant for many years.

Inadequate protection in production

Good security hygiene includes rigorous vulnerability scanning, compliance auditing, and hardening of production infrastructures. But no amount of scanning and preparation will eliminate the risk of zero-day, insider, and phishing attacks. If you have sensitive data or valuable assets to protect, run-time security with deep network and endpoint (containers, host) protection is a must.

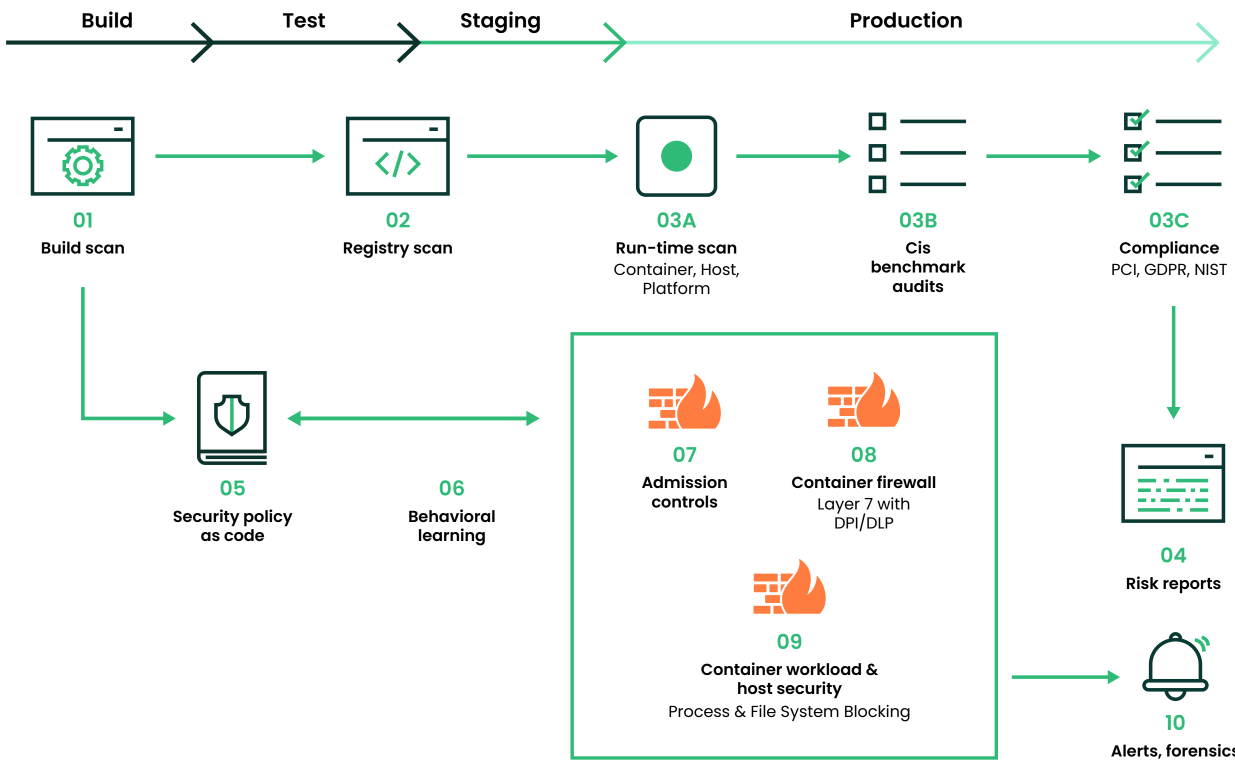
10 Steps to Container Security Automation - A Summary

In this guide we'll take a detailed look at ten integration points which can serve as a guide for 10 steps for container security automation. While there are many more than ten, these are highlighted as first steps because they are simple and impactful for improving security. Here's a summary of these, followed by a pipeline reference diagram.

1. **Build-phase scanning**
   Container images should be scanned for vulnerabilities and compliance violations during the build phase so the build step can be stopped (failed) in order to force correction or remediation. Integration is made easy through plug-ins and extensions for popular tools such as Jenkins, CircleCI, Azure DevOps, Gitlab, Bamboo etc.
2. **Registry scanning**
   After images pass build-phase scanning, they are staged into registries and should also be scanned there. New vulnerabilities can be discovered or introduced after images are pushed to registries. Registry scan results can also be linked to Admission Controls (see #7 below) to prevent unauthorized or vulnerable images from being deployed.
3. **Production scanning, cis benchmarks & compliance auditing**
   Scanning and auditing should extend into the staging and production environments with run-time vulnerability scans and running of CIS benchmarks for Kubernetes and Docker, as well as any custom compliance checks.
4. **Risk reporting and vulnerability management**
   Although addressing high risk environments and interpreting vulnerability management reports typically involve some manual review, alerting and correlation can be done automatically to speed and ease assessment and remediation.
5. **Security policy as code**
   Defining and deploying security rules for new applications can be automated as code in the same way that Kubernetes supports deployment manifests in standard yaml files. In this way, new or updated workloads are automatically protected as they are deployed to production.
6. **Application behavioral learning**
   Behavioral learning is an important technique for automatically characterizing application behavior such as network connections, protocols used, processes required and file access activity allowed. We will see later how this can work together with Security Policy as Code (#5) to automate run-time security from dev to production.
7. **Admission controls**
   Admission controls are an important bridge between the CI/CD pipeline and the production environment. Once rules are established, vulnerable or unauthorized deployments can be automatically prevented.

8. **Container network firewall**
   A layer 7 (application layer) container firewall will automatically enforce network segmentation by blocking network attacks and unauthorized connections. The creation and maintenance of these whitelist network rules can be automated as code (#5) utilizing behavioral learning (#6).
9. **Container workload & host (endpoint) security**
   Containers and hosts should be continuously monitored for suspicious and unauthorized behavior such as processes and file activity. These activities can be automatically blocked. The creation and maintenance of these whitelist rules can be automated as code (#5) utilizing behavioral learning (#6).
10. **Alerting, response, and forensics**
   Finally, automated alerting, response, and forensic capture can be initiated for suspicious activity. These should include quarantine of containers, packet captures, and custom notifications to case management systems.

10 steps to full lifecycle security



Who Should Implement These Steps?

The 10 steps summarized above provide a great start to security automation. But who should be responsible for each of these steps? The answer is, of course, it depends. While the many roles involved in the pipeline can include developers, DevOps, SREs (site reliability engineers)/operations, compliance, and security, every organization will have a different approach and process for managing security.

Let's take a simplified view of two important roles: DevOps (which includes DevSecOps), and Security. These considerations can help determine who is responsible for each step:

Security teams are ultimately chartered with protecting critical assets, especially in a production environment. However, the 'Shift-Left' movement to build security into the pipeline earlier means that DevOps needs to understand security concepts and own some aspects of security automation.

While security teams are often trained on concepts such as network segmentations, firewalls, endpoint security, IDS/IPS and incident response, they are often not as familiar with modern cloud orchestration tools such as containers and Kubernetes. They will need to learn which of these traditional security concepts should be applied and enforced in cloud-native deployments, and how they should be implemented.

While DevOps teams are rapidly becoming experts on automated CI/CD pipelines and how to manage orchestration tools such as Kubernetes, they often lack the understanding and experience of security technologies and best practices in production environments required to fend off the constant attacks on the infrastructure and applications.

New cloud-native tools such as registries, orchestrators (Kubernetes), service meshes and other open source tools introduce new attack surfaces that are not known to either security or DevOps teams.

New concepts such as 'infrastructure as code' and 'security policy as code' will require collaboration between DevOps and security teams in order to enable automation of pipelines in a secure environment.

In general, the earlier steps (1-4) are typically the responsibility of DevOps and Compliance teams, with the later steps (8-10) being the responsibility of Operations and Security teams. The middle steps (5-7) are the bridge between the CI/CD pipeline and the production environment with Security Policy as Code and Admission Controls being managed by the DevOps teams with oversight from the Security team.

How Should the Integration Be Done?

There are tools and techniques available for automating all the 10 steps and more, from plug-ins to simple scripting. Many automations are already built into tools like SUSE Neu-Vector to simplify security enforcement.

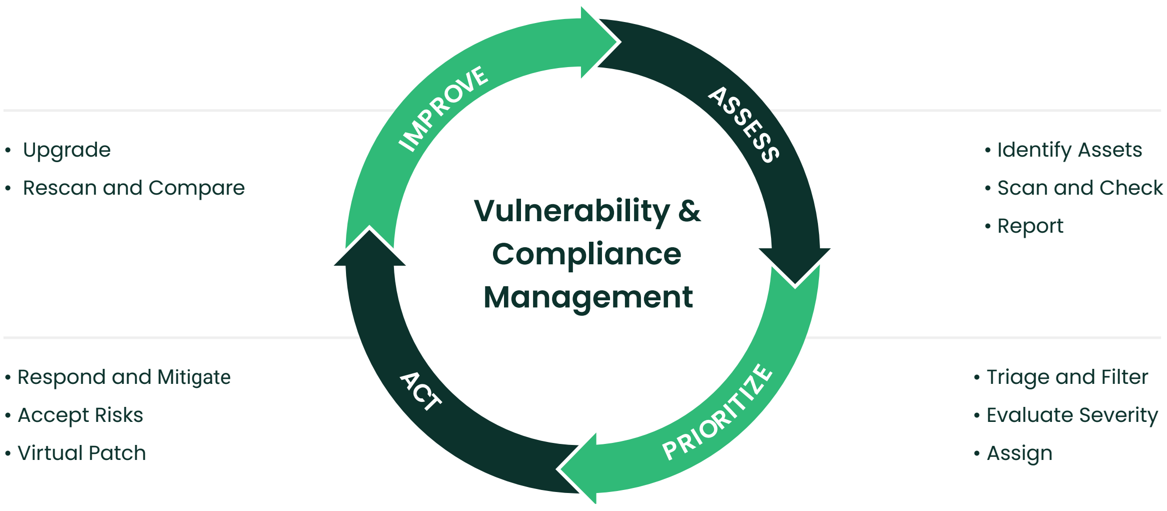
Plug-ins and extension	Built-inintegrations	Rest API's	Notifications
The most frequently used security automations have plug-in or extensions, such as a Jenkins plug-in for build-phase scanning.	Modern cloud-native tools such as Kubernetes offer resources to enable security integrations such as admission controls, CRDs or RBAC enforcement. These can be leveraged by security tools like SUSE NeuVector to provide a rich set of security features automatically integrated with the orchestrator.	For any integration that needs to be highly customized or is not supported by a plug-in, a fully featured REST API provides a way to script security automations. For example, all scanning, policy rules, user management and every configurable function in SUSE NeuVector can be controlled through the REST API.	Alerting and notifications can be automated through traditional methods such as SYSLOG events or modern triggers such as webhooks. These can not only trigger alerts and notifications but can open cases and trigger responses such as quarantines and packet captures.

There are tools and techniques available for automating all the 10 steps and more, from plug-ins to simple scripting. Many automations are already built into tools like SUSE Neu-Vector to simplify security enforcement.

In most cases, some simple configuration for customizing the security policy is required initially, and even this can be automated through REST APIs. Once configured, scanning, monitoring, enforcement, updating, and alerting can all be automated.

Steps 1-4: End-to-End Vulnerability and Compliance Management

End-to-end vulnerability management should follow the Assess, Prioritize, Act, and Improve workflow as shown below.



Key Automation Steps in the CI/CD pipeline include

1. **Build-phase vulnerability scan triggers**
   A. Use plug-ins, extensions, or REST API's to enforce scanning during image builds to catch vulnerable images early in the pipeline. For example, SUSE NeuVector provides a Jenkins plug-in, CircleCI ORB, Azure DevOps extensions, REST API and other ways to force build-phase scans. Images can also be annotated with the developer or team responsible to make alerting or reporting easier.
   B. Alerts for critical vulnerabilities discovered should go to the compliance management team and potentially the developer responsible for the image. These can be filtered to only alert and fail the build if there is a fix available for the vulnerability. If the image is annotated with the developer name alerting becomes easier and more accurate. See the section below on Virtual Patching for an overview of how to handle vulnerabilities with no fix available that must be allowed to exist in production.

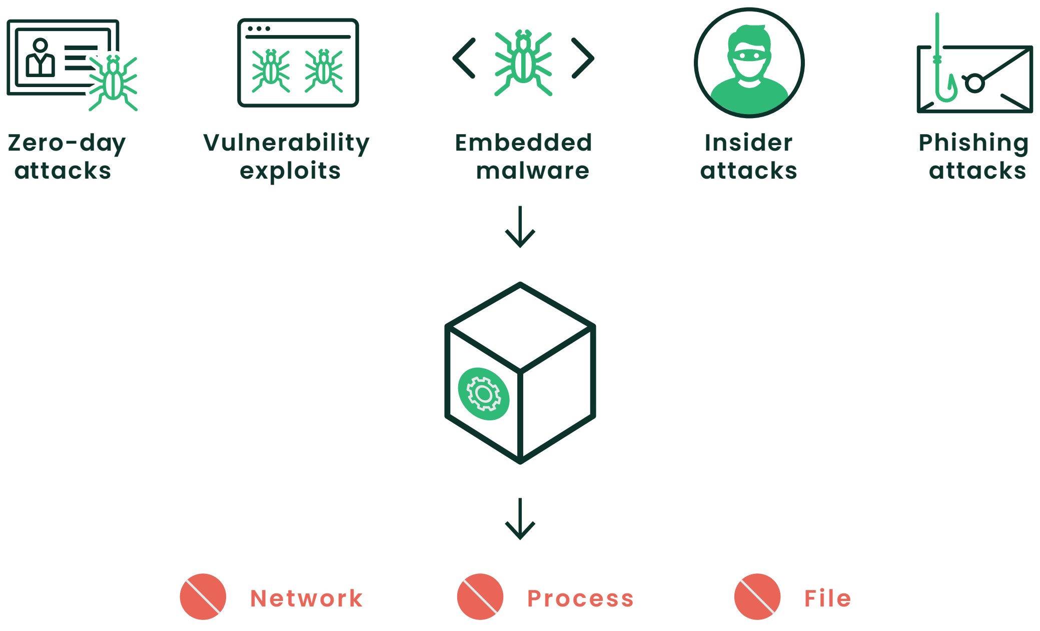
2. **Automated registry scans**
    A. Continuously monitor images in each registry being used for staging and production environments. Layered scan results can make it easier for developers to find the vulnerable package or library by examining the build commands for each layer. Images can also be annotated with the developer or team responsible to make alerting or reporting easier.
    B. Alerts can be handled similarly to build-phase scans where only critical vulnerabilities with fixes available are sent to developers for remediation. If the image is annotated with the developer name alerting becomes easier and more accurate. While analyzing and alerting should be automated, some manual processes may be required initially until integration is tested and completed.
3. **Run-time (production) scanning & auditing**
    A. Continuously monitor images in each registry being used for staging and production environments. Layered scan results can make it easier for developers to find the vulnerable package or library by examining the build commands for each layer. Images can also be annotated with the developer or team responsible to make alerting or reporting easier.
    B. Run CIS-benchmarks and custom compliance checks continuously.
    C. Enforce industry compliance requirements such as firewall and segmentation for PCI and other privacy standards. See the sections later in this document on Security Policy as Code, Behavioral Learning, and Network Segmentation for ways to automate compliance for network firewall enforcement.
    D. Alerts for critical run-time vulnerabilities with fixes available as well as certain CIS benchmarks such as containers running as root should be sent to the DevOps team for investigation.
4. **Analyze, triage, and correlate the 'impact' of vulnerabilities in production**
    A. Correlate vulnerabilities to images, containers, hosts. Assess which production assets are affected by critical vulnerability and compliance issues.
    B. Implement 'Virtual Patching' to mitigate production risks.
    C. Alerts on unprotected assets with critical vulnerabilities should be sent to DevOps and security teams. Assets which have virtual patching applied can be downgraded or lowered in priority because they present a lower exploit risk.

What is Virtual Patching?

Virtual Patching enables security teams to 'virtually patch' a vulnerability in a running container or host, without needing to update or replace the running asset with a patched version. It essentially protects the running asset (container, host etc.) against an attempted exploit of the vulnerability.

The best way to do this is to automatically characterize and whitelist all application container behavior such as network connections, processes, and file activity, then lock it down. The workload or host are then 'virtually patched,' and any attempted exploit will create an unauthorized network connection, process, or file access which can be blocked.

Virtual patching protects against vulnerability exploits, embedded malware, zero-day attacks, and insider & phishing attacks.



Additional Reading | See this article for more information about End-to-End Vulnerability Management and Virtual Patching

Steps 5-6: Security Policy as Code and Behavioral Learning

Automating vulnerability scanning is a great start for securing containers. What is even more challenging and important is automating the creation of security policies to protect application workloads in production. Run-time security policies, especially firewall rules, have up to now largely required manual configuration in legacy data center-based infrastructures.

However, in modern cloud deployments, the use of Kubernetes custom resources to declare an application security policy at any stage in the pipeline provides a solution to this problem.

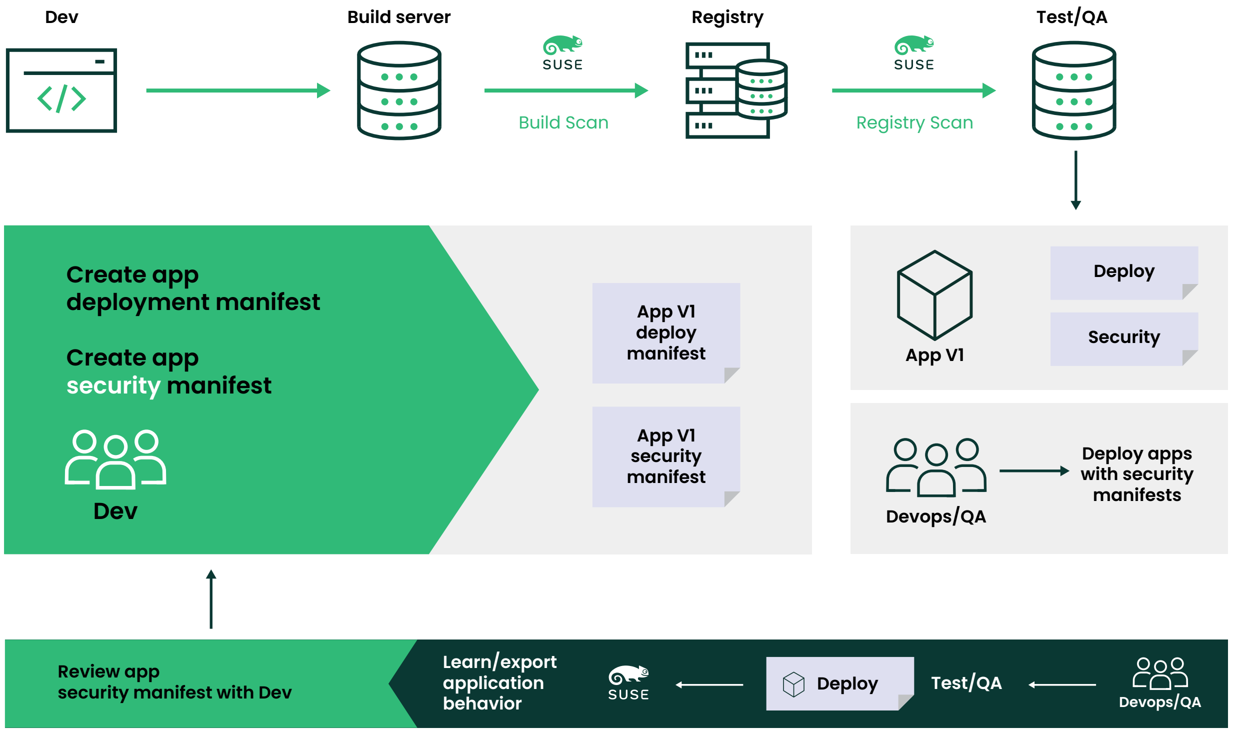
Developers, DevOps, and Security teams can use custom resource definitions (CRDs) to automate and maintain run-time security policies. CRDs can also be used to enforce global security policies across multiple Kubernetes clusters.

Behavioral Learning is a useful technique to learn and characterize an application's behavior to automatically draft the security policy CRD. This CRD can then be reviewed by Dev, DevOps and Security teams and edited as needed, then signed off as the security policy for the application. This CRD 'code' can then be checked into the change management system before it is deployed into production.

Key Automation Steps in the CI/CD pipeline include

5. **Security policy as code and behavioral learning**
   A. Capture the 'allowed' application behavior in standard CRD yaml files. Behavior should include allowed network connections and protocols including ingress/egress, processes, and file access activity. Security policies that are not tied to specific applications such as global security rules can also be expressed in separate CRDs.
   B. Build review, approval, and check-in steps into the pipeline so the security policy as code represents the ultimate authority for production security rules.
   C. Deploy the security policy CRDs into the production environment before the new or updated applications are deployed, thereby ensuring the new workloads are secured from the moment they start running.
6. **Use behavioral learning to automate security policy**
   A. In the test, QA, and/or staging environments, run the applications and their associated test suites and use behavioral learning to capture allowed application behavior in the form of network, process, file and other security rules.
   B. Export the rules as a CRD for review and use in Step 5 above.

The diagram below shows the workflow using behavioral learning to help generate the security policy CRD for a new application.



The top of the diagram shows the workflow where developers create the app, app deployment manifest, and build the image, which goes through automated scanning process in Steps 1 and 2. At the same time, the DevOps/QA team runs the app (bottom of the diagram, going right to left), and uses SUSE NeuVector behavioral learning to create the security manifest (CRD) for review with the Dev team. After being approved and tested, the CRD is checked-in to the change management system and deployed into the production environment to protect the new application from run-time attacks.

What is a SUSE NeuVector Custom Resource Definition (CRD)?

The SUSE NeuVector CRD enables creation of security policy as code with Kubernetes-native yaml files. The CRD defines allowed application behavior, such as:

Network connections	Container process	Container file activity
These include Layer 7 rules based on application protocols. The rules can enforce east-west application segmentation as well as ingress and egress rules.	Processes in the application container are whitelisted so unauthorized processes can be detected.	File activity with specific applications which are whitelisted can be defined.

In addition, other SUSE NeuVector security related configurations can be declared:

Groups	Application (container) protection state	Other configurations
Logical groupings of containers or other objects can be created in SUSE NeuVector, and rules can be applied to these Groups.	The protection state of applications, which is Discover, Monitor, or Protect.	Additional settings can be configured, and this format can be extended to add more in future versions. These are easily applied and updated, for example using 'kubectl apply -f new_policy.yaml' commands.

CRDs are RBAC controlled by Kubernetes automatically, so only users with appropriate access (namespaced or cluster admin) will be allowed to create or modify the security policy as code resources.

Additional Reading | See this article for more information about implementing Security Policy As Code

Steps 7-9: Complete Run-Time Protection

Once they are running in production, containers, their hosts, and the orchestrator need to be protected against attacks. Complete run-time protection should include container process and file access controls, host monitoring, and most critically, network security with deep packet inspection. In the past, network firewall rules and endpoint security policies required heavy manual customization, but this can't be required for modern automated pipelines.

Run-time security starts with admission controls as a gatekeeper for the production environment, and progress through run-time scanning and compliance checks to real-time attack prevention.

Key Automation Steps in the CI/CD pipeline include

7. **Use admission controls to prevent vulnerable or unauthorized deployments**
   A. Use criteria such as vulnerability scan results, namespaces, registries, and container properties (e.g. running as root) to control deployments.
   B. While creation of these rules can be automated or manual, enforcement and alerting should be automated.
8. **Deploy a layer7 container firewall for automated segmentation and threat detection**
   A. Automate blocking, quarantine, packet capture, and alerting for network attacks and segmentation violations.
   B. Use Security Policy as Code to automate rule creation and updates.
9. **Use endpoint security controls for containers and hosts**
   A. Automate blocking and alerting for suspicious process and file activity in containers and hosts.
   B. Use Security Policy as Code to automate rule creation and updates.

What is Container Network Segmentation?

SUSE NeuVector provides a true cloud-native Layer 7 container firewall which does network segmentation automatically. By using behavioral learning, connections and the application protocols used between services are discovered and whitelist rules to isolate them are automatically created. This means that container segmentation is easy and automated, without requiring knowledge of connections beforehand or the manual creation and maintenance of segmentation rules.

With a cloud-native, Layer 7 container segmentation solution, workloads can be segmented even if they are running on the same host or in the same cluster. This is especially valuable to fulfill industry standard compliance requirements such as PCI DSS.

The Ultimate Cloud Security Pattern - Container Segmentation by Workload

Ultimately, to give the business the most flexibility for rapid release and optimal resource utilization, container segmentation must be enforced on each pod and follow application workloads as they scale and move dynamically. In this micro-perimeter vision article, SUSE NeuVector CTO Gary Duan outlines a vision for cloud security where the protection perimeter surrounds the workload even as it moves across hybrid clouds.

Additional Reading | See this article for more information about using Admission Controls to prevent unauthorized and vulnerable container deployments.

Additional Reading | See this article for more information about Container Segmentation Patterns, including for PCI compliance.

Steps 10: Alerting, Response, & Forensics

Many automations in this step are fairly well understood and implemented for other infra-structures. For example, event reporting to SIEM systems for alerting and forensic capture. In a container-based environment, some things are more difficult, such as initiating pack-et captures only on suspicious containers. Or correctly identifying the pod, namespace or application service by its Kubernetes deployment name, in order to investigate where an attack is occurring.

Key Automation Steps in the CI/CD pipeline include

10. **Automate alerting and real-time responses**
    A. Generate special alerts for critical security events.
    B. Quarantine suspicious containers.
    C. Initiate packet captures for investigation and forensics.
    D. Integrate with case management tools to address security and compliance violations.

SUSE NeuVector provides several mechanisms for automation and integration in Step 10. These include SYSLOG event output, customized webhook alerts, automated packet capture & quarantine, and a REST API for customized, scripted automations. A Prometheus exporter and Grafana dashboard is also supported.

Summary

Take the steps now to automate security

The road to a fully automated pipeline with integrated security controls all the way into production starts with a few easy first steps. Which of the ten steps in this document you start with depends on where you are in the process of automating your pipeline. What is most important is to get started with what can be done easily and what is most critical at this time, then create a roadmap to fully automated security.

For example, build-phase and registry scanning is fairly easy to automate and delivers good value in return. Automated run-time security responses such as alerting, blocking, quarantine and packet capture are also easy to enable. In contrast, requiring Security Policy as Code can be more complicated, from both a technical creation and workflow testing perspective as well as a human process perspective. To be successful, DevOps and Security teams need to agree on roles, responsibilities, pipeline processes, and the role that developers play in deploying security policy as code.

The good news is that implementing even a small subset of the steps outlined in this guide will improve pipeline automation and move organizations towards the goal of pipeline automation with security integration.

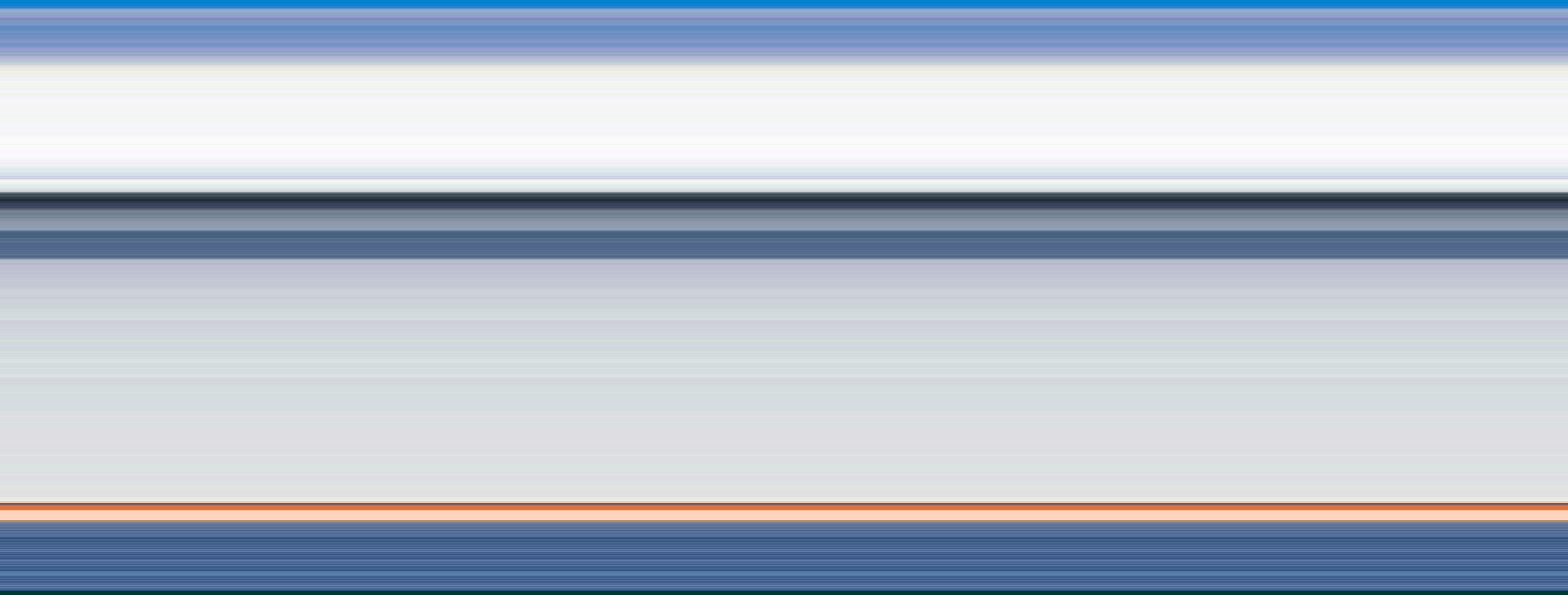
Next Steps

Want to learn more?

Contact SUSE NeuVector at https://NeuVector.com for more container security articles on our blog and to schedule a demo of the SUSE NeuVector Container Security Platform, including the Layer 7 Container Firewall.

SUSE

2022



SUSE
Maxfeldstrasse 5
90409 Nuremberg
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