StackRox

Container and Kubernetes Security: An Evaluation Guide

Your Definitive Resource for Protecting Containers, Kubernetes Orchestrator, and Infrastructure

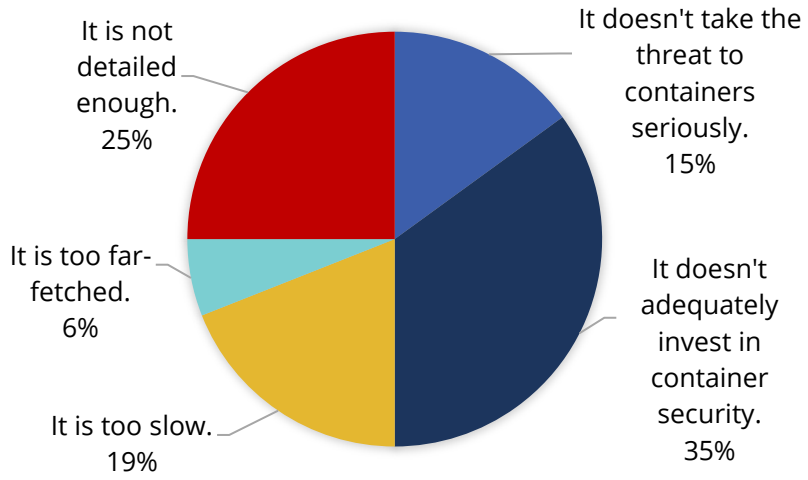
Table of Contents

Overview

More and more organizations are transforming their businesses by embracing DevOps principles, microservice design patterns, and container technologies such as Docker and Kubernetes. While security teams have the same mission regardless of the technology stack in use – keep the bad guys out and find and stop them if they do break in – the tools and tactics security staff employ must change to accommodate this infrastructure shift.

Security remains the primary concern around enterprise container strategy. In a recent survey[1] of IT and security practitioners and decision makers, 35% of respondents identified a lack of adequate investment in container security as their biggest concern relating to their organization's container strategy, with another 15% lamenting that their container strategy doesn't take security threats seriously.

Q. What is your biggest concern about your company's container strategy?



The right container security solution can address these concerns by leveraging the operational advantages of containers. Container runtimes and orchestrators have brought new standard interfaces to the way applications are built, deployed, and operated. Security, operations, and development teams can speak a common "language" using these interfaces to gain greater insight into — and control over — security-critical configurations. And, when something goes wrong, the same design patterns that allow containerized applications to be resilient to infrastructure failure can be used to automate security response.

[1] The State of Container Security 2018 Report, StackRox with observations and analysis from CyberEdge Group

A successful container security program requires integrating security into each phase of the container lifecycle: Build, Deploy, and Run.

- **Build**
  The build phase centers on what ends up inside of the container images developers create. In the build phase, security efforts are typically focused on reducing business risk later in the container lifecycle by applying best practices and identifying and eliminating known vulnerabilities early.

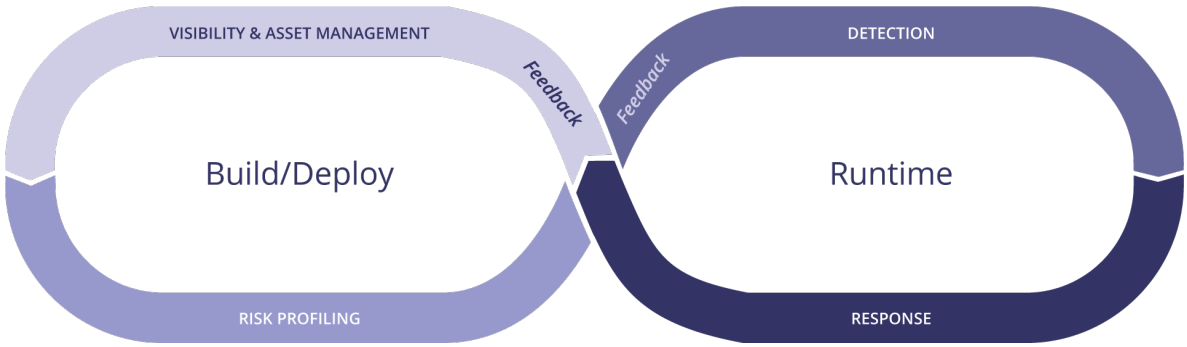
- **Deploy**
  Containerized applications are configured in the deploy phase. In this phase, context about images can be combined with the rich variety of configuration options available for orchestrated services. Security efforts in this phase often center around compliance with operational best practices, least-privilege principles, and identifying misconfigurations to reduce the likelihood and impact of potential compromises.

- **Run**
  The runtime phase is when containers go into production with live data, live users, and exposure to networks, internal or the public Internet. The primary purpose of security during the runtime phase is protecting both running applications and the container infrastructure by finding and stopping malicious actors in real time.

As critical as protecting containers across their life cycle is the need to ensure the underlying infrastructure is properly configured. Containers can help organizations implement finer-grained workload-level security, but they also introduce new infrastructure components and unfamiliar attack surfaces. The right container security solution must help secure the cluster infrastructure and orchestrator as well as the containerized applications they run.

Finally, the right solution must integrate seamlessly into an organization's architectural plans, tools, and business workflows. Otherwise, the solution may negatively impact business operations, lock the organization into specific technologies, or increase the workload on already busy teams.

This guide outlines the most important security controls and capabilities that a solution must provide to secure each phase of the container lifecycle, your Kubernetes orchestrator, including clusters, nodes, container engines, and other infrastructure components, to achieve effective overall security.



Build Phase – Building Secure Images

Container images specify the programs, tools, and components that will be available when an application is deployed and run, as well as some important configurations. The makeup of container images sets a security baseline as applications progress through the container lifecycle.

Time and effort invested to reduce the attack surface in the build phase is well spent: the earlier in the container lifecycle that organizations catch vulnerabilities, configuration errors, and security policy violations, the lower the cost to the organization. It's far more efficient to mitigate a problem in a container image in the build stage than to respond to the same issue in production. Similarly, fixing a problem in a widely used base image can address security problems across multiple teams and applications.

An effective container security program seeks to remediate vulnerabilities and reduce the attack surface before images ever leave the development sandbox. Because container images are immutable and identified by their contents, security-relevant changes usually need to be applied by the same development team and built using the same build process that produced image. So, it's best for security to "shift left" as far as possible rather than instituting controls in later stages of deployment.

Vulnerability scanning is a key element of image security. Container security solutions should assist teams in identifying fixable vulnerabilities and enable automated enforcement to block new issues from being introduced. But too many solutions focus primarily on vulnerability scanning – solutions must go far beyond this capability to protect at the build phase. Container security solutions should asses compliance with other security practices during the build phase. For instance, a security team might decide to minimize the attack surface by requiring tools—like apt-get or curl—that are useful to attackers not be included in final images.

Images also contain certain elements of runtime configuration, such as the program that will execute, the user identity with which it will run, and a default set of environment variables. Container security solutions should also be able to check these configurations for compliance with organizational security policies.

Required Features and Functions: *Building Secure Images*

Feature / Function	Why You Need This
IMAGE ASSESSMENT	
Analyze image configuration settings to detect issues, including Dockerfile instructions used in the build, user identity configurations, and environment variables	Vulnerability scanners identify known vulnerabilities in packages but not risky practices. Analyzing image configuration settings can detect additional security-relevant weaknesses that a scanner will miss.
Identify vulnerabilities in image components	Shipping images with vulnerable versions of dependencies is an avoidable risk. Reporting known vulnerabilities early in the container lifecycle helps avoid moving this risk into production. Your security solution should support scanning images for vulnerabilities to give you this information.
Gather data from existing scanners, including those built into registries	If your organization has purchased a scanner, including a registry-integrated scanner, your security solution should maximize the value of that existing investment by using data from it to gather enhanced security context.
Attribute vulnerabilities and configuration problems to specific components and specific image layers	Container images often are based on open-source or corporate-standard base images. Vulnerabilities and other issues in base image layers are often the responsibility of a different team, so it's important to be able to attribute issues to those layers.
Specifically identify fixable vulnerabilities as well as those that are not fixable to avoid future alerts	In some cases, operating system distributors or open-source maintainers may assess that a given vulnerability does not apply or does not warrant a fix. Interrupting a development team's workflow over such an issue is not productive if a fix is not readily available.
RESPONSE	
Fail Continuous Integration (CI) builds for severe issues	To effectively contain risk in the build phase, you need to inject feedback during the development process as soon as a problem is introduced. Failing a build informs developers as quickly as possible.
Allow customized control over the conditions that will fail a build	Overly aggressive blocking may cause teams to disable security controls if violations are too costly or unrealistic to remediate.

Deploy Phase – Hardening the Environment

Once you've built your images, the next step is to deploy them. To achieve security in the deployment phase, you need to understand and manage risk factors including those posed by:

- The contents of the images being deployed
- The configuration of individual containerized workloads

While default orchestrator and container engine settings provide a reasonable level of security in some areas, other settings require specific configurations to lock them down. The right container security solution should not only display violations of standard policies but also provide the DevOps team with recommended changes to quickly correct unnecessary risks or misconfigurations.

Individual workloads have rich configuration options that affect the security posture of the application. To fully understand the risk posed by an application, you need to know:

- What it is (including information about the image, such as components or vulnerabilities)
- Where it came from (including the registry and tag)
- How it's deployed (including user identity, privilege level, and other configurations)
- What it can access (including secrets, volumes, and other infrastructure components such as the host or orchestrator API)
- Whether it complies with your policies and security requirements

Only with all of this data can you get a full understanding of your risk posture, so you know where to target remediation and hardening efforts. And only with checks and enforcement on all of these attributes can you ensure that workloads are deployed securely.

Within a cluster, sensitive workloads should also be separated logically and physically. Different groups of applications, and services with different security needs, should be deployed in separate namespaces as a first level of isolation. The most sensitive workloads should run on dedicated nodes so that less trusted workloads are not able to affect their operation.

Required Features and Functions: *Hardening the Environment*

Feature/Function	Why You Need This
WORKLOAD VISIBILITY	
Deliver multi-factor risk rankings based on how images are built and how deployments are configured	The container image is an important starting point, but the full picture of a service's risk must take into account key aspects of deployment configuration as well. These deployment attributes should be synthesized into a risk ranking that accounts for: • Violations of organizational security policies and practices • Image vulnerabilities • Access to secrets, storage, and other important resources • Privileges and capabilities • Network exposure settings • Variety and number of components useful to adversaries
Provide visibility into namespace isolation practices	Namespaces are a key boundary for network policies, orchestrator access control restrictions, and other important security controls. Separating workloads into namespaces can help contain attacks and limit the impact of mistakes or destructive actions by authorized users.
Discover vulnerabilities across the entire container environment	To respond efficiently to vulnerabilities and exposures, you need to be able to quickly view data across all of your clusters.
Visualize network segmentation rules	Orchestrators typically provide relatively open networking between components — for instance, Kubernetes by default allows every pod to contact every other pod. Network segmentation policies are a key security control that can limit the ability of an attacker to move laterally through a container environment.

Show external network exposures	External network exposure opens services to a broader pool of potential adversaries. Removing unnecessary exposures can prevent exploits, especially when new vulnerabilities emerge.
Provide visibility into use of secrets	The container infrastructure can enforce access controls to prevent secrets from being read by unrelated deployments, but that step helps only if deployments mount just the secrets they actually need. Visibility into secrets helps you find this type of unnecessary exposure and can expose other weaknesses, such as short key lengths.
Provide visibility into the use and configuration of persistent storage	Persistent storage is a key persistence vector in otherwise ephemeral container environments. Persistent storage may also store valuable, sensitive data. For these reasons, a container security solution should identify the use and configuration of persistent storage.
Provide visibility into the use and configuration of host mounts	Host mounts are another vector that adversaries use to establish persistence or interfere with other containers. Host mounts are sometimes unintentionally writable or overbroad. A container security solution should inform you about the use and configuration of host mounts.
POLICY ASSESSMENTS	
Assess privileges used by containers	The capabilities, user identity, and privileges granted to container processes can allow or block many security risk vectors, and a container security solution should examine these settings to minimize risk.
Assess image provenance, including registries and other attributes	Your container security solution should help you understand where the code in your environment was deployed from and should provide controls that prevent deployment from untrusted sources or registries.
Ensure old—and potentially vulnerable—images are not being used	Your security solution should include the age of images in their risk profiling, since older images can contain more vulnerabilities than fresher ones. Your solution should be able to alert you when very old images are deployed.

Assess image scan status before, during, and after deployment	It's important to enforce distinct policies based on last vulnerability scan date because images that haven't been scanned recently may contain vulnerabilities that are discoverable but not yet reported for the image.
Assess resource requests and limits for deployments	Deployments without resource limits can cause availability issues for themselves or for neighboring applications, especially if they are compromised and used to run cryptocurrency miners or similar payloads.
Assess operational best practices, such as labels and annotations or the use of the "latest" tag	Workloads that don't specify standard annotations or labels are harder to understand, making it more difficult to remediate security issues. Using tags like "latest" increases the risk of new code being accidentally deployed.
Deliver out-of-the-box policies detecting common configuration problems, including: • High-severity vulnerabilities • Image scan age and other best practices • Abnormal privilege levels • Security-relevant image components, including package managers and download tools • Key vectors for adversary persistence attempts, such as sensitive host directory mounts • Common bad practices, such as distributing secrets in environment variables	Configuration options evolve over time and may require significant research to develop appropriately specific and understandable policies. Look for out-of-the-box policies in your security platform to help your team initiate its security efforts. A solid base of pre-built policies allows your team to instead focus on building custom policies in areas they know best: your applications and practices.
Enforce granular policies uniquely designed to specific environments and their security requirements	Different organizational policies may apply in different environments; for example, production deployments may be held to a higher standard than testing. Policies may also need to be more restrictive for sensitive deployments, such as those that process regulated data. Your security solution must support this type of scoping.

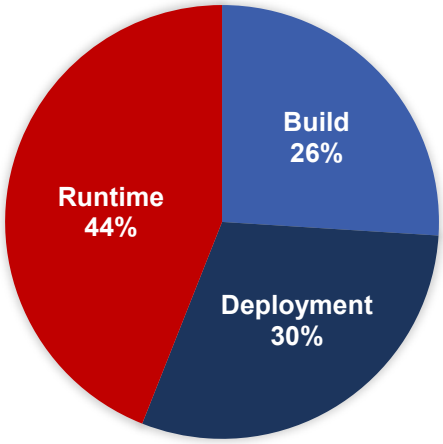
PROACTIVE HARDENING	
Simulate or preview new network segmentation policies and compare them to runtime traffic	It can be difficult to understand the effects of network policies, especially when multiple policies apply to deployments in different ways. This difficulty can lead to overbroad rules. YAML text files can also be hard to decipher. Simulation and preview features make it easier to write fine-grained segmentation rules and visualize the network policy changes.
Automatically generate new network policies based on observed traffic in runtime	It can be tedious to create adequate network policies, especially when applications have long relied on an open-by-default configuration. A container security solution should generate new network policies based on observed traffic in runtime.
RESPONSE	
Automatically notify appropriate teams based on deployment metadata	Many organizations annotate deployments with the name, email alias, or Slack channel of the team responsible for an application. The security solution should be able to use this metadata effectively to prevent security teams from having to manually triage and forward alerts and findings.
Block deployments that violate policies	The best way to ensure ongoing compliance with policies is to prevent noncompliant workloads from being deployed.

Runtime Phase – Securing Running Containers

Once container images are built and deployed into production, they are exposed to new security challenges and a larger spectrum of adversaries. The goal of the security controls instituted in the Build and Deploy phases is to minimize risk and prevent attacks, but your security solution must also detect and respond to issues as containers run.

According to our survey report, a near majority of respondents indicate that securing the runtime phase of the container life cycle is their greatest concern.

Q. Which life cycle phase are you more worried about from a security perspective?



In runtime, security efforts should focus on protecting both applications and infrastructure and providing both visibility and detection features. A security solution must monitor the most security-relevant container activities, including:

- Process activity
- Network communications among containerized services
- Network communications between containerized services and external clients and servers

Container image and application deployment formats allow developers to better specify their intent and enable easier introspection into what's deployed. Effective container security solutions must take into account the context accumulated during the Build and Deploy phases of the container lifecycle to deliver more effective and accurate runtime detection.

In addition, effective runtime security requires behavioral analysis of container runtime activity using sophisticated techniques that establish risk, detect and respond to active attacks, and track attack progression. Basic approaches including whitelisting, blocking, and static log analysis are insufficient to detect and respond to advanced attacks.

Key Elements of Securing Containers in Runtime

- **Visibility**
  Security practitioners have the upper hand when intruders can't hide. An effective security solution should identify the riskiest places to investigate and should give security professionals effortless access to the data they need to understand the container environment and respond to alerts.
- **Continuous Monitoring**
  New vulnerabilities and exploit techniques appear every day, and security teams are often tasked with finding these problems in running applications and coordinating remediation efforts.

  Your security solution must monitor for and discover new vulnerabilities in running services and enforce custom policies specific to those services. If your security solution evaluates policies only when images are first built and applications are first deployed, you won't be protected from these new vulnerabilities, exposures, and attack vectors.
- **Detection**
  Relying solely on blacklists for detection leaves you vulnerable to unknown attack techniques. Meanwhile, overly broad whitelists can suppress true alerts. Relying solely on this type of technique can leave a large amount of suspicious or malicious behaviors undetected. Machine learning and automated correlation should be paired with static rules to optimize detection efficacy.

  Expert users should also be able to use visibility features to conduct investigations and hunt for undiscovered threats.
- **Response**
  Runtime detection events should be part of a feedback loop, prompting changes to the way applications are built and deployed. It's important to focus on responding within the orchestrator and recommending corrective actions that cooperate with the orchestrator and with immutable infrastructure principles. Security solutions

that make disruptive changes at runtime can interfere with DevOps teams' intent, leading to outages, downtime, or unexpected application behavior.

For instance, quarantining a container in runtime could leave the container only partially operational. Containers should instead be replaced if they become compromised – a hobbled container is hard for developers to troubleshoot. In addition, the detection result should spur changes in the earlier phases of the container lifecycle to prevent similarly susceptible containers from launching. Dev and Ops should ultimately be responsible for fixing vulnerabilities; the security product should gather the data that those teams need, contain the damage, and recommend solutions — not interfere.

Required Features and Functions: *Securing Containers in Runtime*

Feature/Function	**Why You Need This**
CONTINUOUS MONITORING	
Continuously assess all security policies, even after applications are first deployed	Security and DevOps teams learn more about their exposures over time. Checking policies once isn't enough — running deployments need to be continually reassessed.
Monitor running deployments for newly discovered vulnerabilities	Vulnerability data on running containers must be continually updated rather than relying on a single image vulnerability scan at a particular time. New vulnerabilities may be disclosed publicly after deployment, so images need to be periodically rescanned, and any new results need to be included in the risk score and context for the deployment.
VISIBILITY	
Visualize active network traffic between orchestrated microservices—organized by cluster, namespace, and deployment	Microservices and containerized applications typically make extensive use of cluster networking to cooperate with other services. Active network traffic is a key way to understand how applications are interacting and spot unexpected contact.

Display all processes executed in containerized deployments	Processes are an important indicator of security and operations-relevant container activity. In containers, ad-hoc process launches are less frequent and more suspicious than they would be in other types of infrastructure.
Compare and analyze different runtime activity in pods of the same deployments	Containerized applications are replicated for high availability, fault tolerance, or scale reasons. Replicas should behave nearly identically; replicas with significant deviations from the others could indicate they've been compromised.
Enable searching of runtime activities and deliver related deployments, violations, and other resources	The same malicious or unwanted activity might affect multiple deployments across different applications or environments. Staff investigating a potential incident need to quickly find those exposures.
Deliver deployment-focused views with context from build, deploy, and runtime	Deployments are the most natural unit of organization and are understood by teams responsible for different parts of the container lifecycle. To easily understand a runtime detection event or decide a hardening improvement, security staff must have all of the context about the workload's activity, its configuration, and its component images.
Provide support for threat hunting	The container security system should conveniently provide necessary actionable data for threat hunting. Related capabilities include security alert details and supporting information such as suspicious runtime activity.
DETECTION	
Define custom and granular policies based on process name and arguments at runtime	Processes are an important indicator of security- and operations-relevant container activity. Process names and their arguments provide important visibility into a container's activity. And, even if an image includes non-default aliases or renamed binaries, attackers will still attempt to use well-known names.
Optionally restrict where runtime policies apply	Different organizational policies may apply in different environments; for example, production deployments may be held to a higher standard than those built for testing. Policies may also need to be more restrictive for sensitive deployments, such as those that process regulated data. Your security solution must support this type of scoping.

Define custom policies that combine runtime policies with attributes from build- and deploy-time	Build- and deploy-time context, such as orchestrator annotations or image details from scanners and registries, can help security teams build more effective and targeted policies. For example, images from public registries might be less trusted, and interactive debugging activities might be more restricted in sensitive applications.
Deliver out-of-the-box policies for attempted or successful: • Privilege escalation • Network reconnaissance • Package installation • Cryptocurrency mining • Modification of host configuration • Execution of reverse shells or other remote access tools • Data exfiltration	Certain container-specific attack techniques are not obvious, and methods to detect them may require research and validation. Look for out-of-the-box policies in your security platform to help your team initiate its security efforts. A solid base of pre-built policies allows your team to instead focus on building custom policies in areas they know best: your applications and practices.
Detect unexpected activities that indicate an adversary may be gaining a foothold, establishing persistence, escalating privilege, moving laterally, or achieving their objectives	Adversary behaviors may not match a specific blacklist item or known vector but should still be identified using more sophisticated techniques that compare activity to each application's baseline.
RESPONSE	
Automatically alert external systems (e.g., email, PagerDuty, Slack, Google Cloud SCC, and SIEM systems) when a policy violation is detected	The container security solution must be able to immediately alert the right responders once an attack is detected.
Provide custom tiered responses based on severity of policy violations	Teams can be overwhelmed by, and begin to ignore, noisy notifications. An effective security solution must be able to tune which problems lead to which responses to avoid alert fatigue.
Send notifications to the most relevant teams based on orchestrator annotations or other metadata	Many organizations annotate deployments with the name, email alias, or Slack channel of the team responsible for an application. The security solution should be able to use this metadata to automate feedback to the relevant development team so security teams don't have to manually triage and forward alerts and findings.

Optionally contain attacks by automatically instructing the orchestrator to replace suspicious instances of running applications	Until the responsible team develops a countermeasure to the underlying security exposure, any action that slows the adversary is valuable. Responses that leave individual containers partially operational can interfere with application uptime and are hard for developers to troubleshoot. Orchestrator-native responses, such as killing and restarting a container, avoid this pitfall and are more understandable to operators.

Infrastructure Security

Security must extend beyond containers themselves to effectively protect container environments. To achieve overall security in the deployment and runtime phases, you also need to secure the cluster infrastructure. Your security solution should assess the security posture of the underlying infrastructure at the cluster, node, and container engine level.

Some of these recommendations are included in community best practices, such as the CIS Kubernetes Benchmark. Your security tool should be able to assess compliance with those standards and bring additional unique security insights and recommendations.

Required Features and Functions: *Infrastructure Security*

Feature/Function	**Why You Need This**
CLUSTER SECURITY	
Ensure that important security features like orchestrator API access controls are activated and used	Leveraging the native security capabilities of the cluster infrastructure helps ensure default workload configurations are more secure.
Assess configurations of key system components, such as the availability of network segmentation.	Network segmentation rules isolate applications in a cluster from each other and from external services to reduce the blast radius of attacks.

Check that authentication and authorization are configured securely	Properly configuring authentication and authorization in a cluster lowers risk and ensures malicious users or compromised applications have limited impact.
NODE SECURITY	
Assess important host-level configurations, including: • The way that key system services are operated • Implementation of host audit logging • Track these settings to enforce corporate security policies and flag any nodes that do not comply.	The physical or virtual nodes running your containers are a fundamental component of the infrastructure. Your security solution should give your containers a secure base on which to run.
CONTAINER ENGINE SECURITY	
Continuously evaluate the container engine's configuration, including important attributes such as: • The engine's security patch level • The unencrypted or insecure registries allowed, if any • The access restrictions on the container engine API • The default configurations for launched containers	Each node runs a container engine, such as Docker or containerd, that operates your applications. The configuration of this container engine has significant impact on the security of the containers it creates. Many out-of-the-box settings are built to make implementation easy, often defaulting to more privileges and rights than necessary. It's critical to continuously evaluate the container engine's configurations and remediate instances of misconfigurations.
Ensure compliance with standards such as the CIS Docker Benchmark to ensure that containerized workloads operate on top of a secure foundation.	Standards such as the CIS Docker Benchmark are created by the security community to identify consensus-based best practices and standards that help organizations improve their container security. A container security solution must be able to assess an organization's compliance with these industry benchmarks.

Container Security Platform Requirements

The ideal container security product will integrate seamlessly into an organization's deployment environments and future architectural plans, maximize the value of existing tools, and enable key workflows.

Many organizations value the portability of Docker and Kubernetes and plan to deploy containers in different environments. Some plan to use a variety of public cloud providers, private data centers, and other infrastructure — including environments that are disconnected from the Internet for security reasons. Even cloud-native companies may plan to mix workloads between cloud providers or want to preserve the option to switch later. The ideal container security solution should support all such architectures.

The container security solution must seamlessly integrate into the organization's DevOps practices and tools, including integration with CI/CD systems, deployment tools, container registries, and other security products. Ideally, the container security solution will provide a tight feedback loop to *shift left*, continuously providing guidance to developers and operations teams. This feedback loop continually improves the overall container security posture.

Finally, the security solution must enable the key workflows that security and DevOps teams execute. Whether the organization needs to get a handle on what's running in containers, assure that container workloads are compliant with regulatory or industry standards, identify hardening opportunities, or find and mitigate security issues, the security solution must be ready to help.

Required Features and Functions: *Container Security Architecture*

Feature/Function	**Why You Need This**
DEPLOYMENT REQUIREMENTS	
Deploy using standard formats such as Kubernetes YAML files and Helm charts	Solutions that use standard orchestrator deployment formats are more portable and more easily fit into existing processes for cluster creation and maintenance.

Support data collection in necessary operating environments	The security solution must be able to deploy, collect data, and execute key functions throughout the secured infrastructure. Key aspects of compatibility include the orchestrator and version, the host operating system, and the container engine in use.
Support self-contained operation without Internet connectivity	Some environments may limit Internet connectivity for security reasons or other purposes. The security solution must be able to operate in such environments. Any functions dependent on remote services may also suffer if connectivity is interrupted or the remote service suffers downtime.
IMPORTANT INTEGRATIONS	
Prefer native orchestrator capabilities to ad-hoc solutions	The container ecosystem is constantly evolving, and many important functions are being implemented through standard interfaces such as the Container Network Interface used in Kubernetes. The security solution will work reliably in more environments and interoperate better with existing tooling if it uses standard, portable interfaces.
Integrate with necessary Continuous Integration (CI) build tools	It is significantly more efficient to solve an issue earlier in the development process. Images created through the CI process set a security baseline for the rest of the container lifecycle.
Integrate with necessary container image registries and scanners	Container image registries and scanners contain useful security context that should be leveraged to improve visibility, detection, and response.
Integrate with necessary enterprise security tools (e.g., SIEM)	Enterprise security workflows often involve data retention or alerting in a central system. The container security solution should support this mode of operation.
Integrate with necessary workflow tools	Remediation work often requires changes from development or operations teams. Notifications and tasking through workflow tools such as Jira, Slack, or email can eliminate delays, frustration, and mistakes from manual triage.

KEY WORKFLOWS	
Automatically discover assets, configurations, and activities	The security solution should discover all relevant data without manual intervention after it is deployed. Any manual steps may lead to oversights and gaps in awareness and protection.
Assess compliance with necessary internal security policies, industry standards, and regulatory requirements	Container infrastructure brings a new set of security options and new components to secure. The security solution must provide data and explanation to help compliance and risk professionals understand and successfully audit these new environments. Even if an organization is not subject to regulatory requirements, compliance with internal security policies using custom policy rules, and industry best practices and standards can make a big difference in security.
Identify opportunities to harden the container infrastructure and workloads	Containers offer an array of security options, and organizations can use them to implement very fine-grained security controls. However, many of these options are not secure by default, and some are difficult to understand. The security solution must proactively identify hardening opportunities and explain them so that teams can implement more secure configurations.
Provide extensive search capabilities across container infrastructure (e.g., cluster, namespace, labels, CVE, image contents, image name, image registry, deployment name, configurations, secret metadata, alert policy name)	To quickly identify vulnerable assets or investigate exposures or attacks, you need powerful, fast search that spans images, deployments, clusters, and all related data. Solutions that leave Build, Deploy, and Run data separated leave defenders at a disadvantage.

Conclusion

As more businesses adopt container technologies, security teams have new adversary models to combat and new infrastructure components to secure. Legacy security solutions leave blind spots and conflict with the technologies and workflows DevOps teams use. Security teams must adapt to this DevOps model to retain their edge in the container world.

A successful organizational container security program must include the implementation of key controls throughout the phases of the container lifecycle – Build, Deploy, and Run – as well as in the underlying container infrastructure. The right container security solution must provide all the critical features in these areas and must integrate seamlessly with the organization's infrastructure, tools, and workflows to deliver fine-grained security throughout the cloud-native stack.