McAfee Guidance on Container Security Strategy Aligned to NIST SP 800-190

Holistic view of container risks, threat scenarios, and countermeasures to help formulate container security strategy
Table of Contents

6  MVISION CNAPP Mapping to NIST SP 800-190
8  Review of Risk Elements and Countermeasures
  8  Image Risks and Countermeasures
  10 Registry Risks and Countermeasures
  12 Orchestrator Risks and Countermeasures
  20 Container Risks and Countermeasures
  24 Host Risks and Countermeasures
26  Key Takeaways
The declarative nature of these systems enables numerous advantages in application development and deployment, like faster development and deployment cycles, quicker bug fixes and patches, and consistent build and monitoring workflows. These streamlined and well controlled design principles in automation pipelines lead to faster feature delivery and drive competitive differentiation.

Legacy application architecture stacks continue to make way for loosely coupled microservices, significantly reducing the cost, allowing parallel development, simplifying complexity, and shortening the time needed to package and deploy applications. Containers (especially Docker) are intended to run single applications which makes them ideal for microservices based application patterns.

However, the unique methods by which application containers are created, deployed, networked, and operated present unique challenges when designing, implementing, and operating security systems for these environments. They are ephemeral, often too numerous to count, talk to each other across nodes and clusters more than they communicate with the outside endpoints, and they are typically part of fast-moving continuous integration/continuous deployment (CI/CD) pipelines. Additionally, development toolchains and operations ecosystems continue to present new ways to develop and package code, secrets, and environment variables. Unfortunately, this also compounds supply chain risks and presents an ever-increasing attack surface.
Figure 1. Example Architecture of Deployment to Containers in AWS

Figure 2. Example Architecture of Deployment to Containers in Azure
As more enterprises adapt to cloud-native architectures and embark on multi-cloud strategies, demands are changing usage patterns, processes, and organizational structures. Despite having a greater percentage of containers in production, these enterprises have only modestly reduced their security concerns, primarily because of the lack of a comprehensive container security strategy or often not knowing where to start. There are legitimate concerns that persist about misconfigurations and runtime risks in cloud native applications, and still too few organizations have a robust security plan in place.

As organizations continue to recognize the need to evolve their security toolchains and processes and somehow embrace complexity via automation, there are common themes that have emerged, such as the emphasis on cloud teams to integrate specific security and compliance checks into their respective DevOps processes to better understand and manage risks over time.

These complex problem definitions mentioned above have led to the development of a special publication from National Institute of Standards and Technology (NIST) – NIST SP 800-190. It outlines a set of guidelines for securing container applications and infrastructure components. While centered in container technologies, it comprises seven main sections:

- Section 1: Introduction
- Section 2: Introduces containers, including their technical capabilities, technology architectures, and uses
- Section 3: Explains the major risks for the core components of application container technologies
- Section 4: Recommends countermeasures for the risks identified in Section 3
- Section 5: Defines threat scenario examples for containers
- Section 6: Presents actionable information for planning, implementing, operating, and maintaining container technologies
- Section 7: Conclusion

This whitepaper covers the mapping of each of the key risk components from Section 3 of NIST SP 800-190 to individual capabilities within MVISION Cloud Native Application Protection Platform (CNAPP). These capabilities are then aligned with a significant majority of countermeasures that have been proposed in Section 4 of NIST SP 800-190, which should be reviewed as reference implementation guidance to automate compliance and implement security for containerized application workloads.
MVISION CNAPP Mapping to NIST SP 800-190

MVISION Cloud Native Application Protection Platform (CNAPP) is a comprehensive device-to-cloud security platform for visibility and control across SaaS, PaaS, and IaaS platforms. It provides deep coverage on cloud native security controls that can be implemented throughout the entire application lifecycle. The chart in this section outlines how CNAPP’s capabilities map to support individual risk elements in NIST SP 800-190 Application Container Security Guide.

MVISION CNAPP first discovers all the cloud-native components mapped to an application, including hosts, IaaS/PaaS services, containers, and the orchestration context that a container operates within. With the use of native tagging and network flow log analysis, customers can visualize cloud infrastructure interactions including across compute, network, and storage components. Additionally, the platform scans cloud native object and file stores to assess presence of any sensitive data or malware. Depending on the configuration compliance of the underlying resources and data sensitivity, an aggregate risk score is computed per application which provides detailed context for an application owner to understand risks and prioritize mitigation efforts.

As a cloud security posture management platform, MVISION CNAPP provides a set of capabilities that ensure that assets comply with industry regulations, best practices, and security policies. This includes proactive scanning for vulnerabilities in container images and VMs and ensuring secure container runtime configurations to prevent non-compliant builds from being pushed to production. The same principles apply to orchestrator configurations to help secure how containers get deployed using CI/CD tools. These baseline checks can be augmented with other policy types to ensure file integrity monitoring and configuration hardening of hosts (e.g., no insecure ports or unnecessary services), which help apply defense-in-depth by minimizing the overall attack surface.

Finally, the platform enforces policy-based immutability on running container instances (and hosts) to help identify process-, service-, and application-level whitelists. By leveraging the declarative nature of containerized workloads, threats can be detected during the runtime phase, including any exposure created as a result of misconfigurations, application package vulnerabilities, and runtime anomalies such as execution of reverse shell or other remote access tools. While segmentation of workloads can be achieved in the build and deploy phases of a workload using posture checks for constructs like namespaces, network policies, and container runtime configurations to limit system calls, the same should also be enforced in the runtime phase to detect and respond to malicious activity in an automated and scalable way.

The platform defines baselines and behavioral models that can specially be effective to investigate attempts at network reconnaissance, remote code execution due to zero-day application library and package vulnerabilities, and malware callbacks. Additionally, by mapping these threats and incidents to the MITRE ATT&CK tactics and techniques, it provides a common taxonomy to cloud security teams regardless of the underlying cloud
application or an individual component. This helps them extend their processes and security incident runbooks to the cloud, including their ability to remediate security misconfigurations and preemptively address all the container risk categories outlined in NIST 800-190.

### MVISION CNAPP Platform Capabilities

<table>
<thead>
<tr>
<th>Risk Element Type</th>
<th>Risks</th>
<th>Configuration Posture Checks</th>
<th>Vulnerability Assessment</th>
<th>Host/VM Protection</th>
<th>Cloud Data Loss Prevention</th>
<th>Workload Runtime Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Image Vulnerabilities</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Image</td>
<td>Image Configuration Defects</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Image</td>
<td>Embedded Malware</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Image</td>
<td>Embedded Clear Text Secrets</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Image</td>
<td>Use of Untrusted Images</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Registry</td>
<td>Insecure Connection to Registry</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Registry</td>
<td>Stale Images</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Registry</td>
<td>Insufficient Auth and Az restrictions</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Unbounded Administrative Access</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Unauthorized Access</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Poorly separated network traffic</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Mixing workload sensitivity levels</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>Orchestrator Node Trust</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Container</td>
<td>Vulnerabilities in Runtime Software</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Container</td>
<td>Insecure Runtime Configurations</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Container</td>
<td>Unbounded Network Access from Containers</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Container</td>
<td>App Vulnerabilities</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Container</td>
<td>Rogue Containers</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Host OS</td>
<td>Large Attack Surface</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Host OS</td>
<td>Shared Kernel</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Host OS</td>
<td>Host OS Component Vulnerabilities</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Host OS</td>
<td>Improper User Access Rights</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Host OS</td>
<td>Host OS File System Tampering</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Figure 3. Mapping NIST 800-190 risks to MVISION Cloud Capabilities
Review of Risk Elements and Countermeasures

The following section outlines a list of container security risks and countermeasures that can be implemented per risk element type, as provided in Sections 3 and 4 respectively of the NIST SP 800-190 Guidance. These recommendations can also be fully implemented via capabilities offered within MVISION CNAPP, which show specific application to each of the risk elements described in the NIST special publication guidance.

Image Risks and Countermeasures

Risks in this section are as follows:
- Image vulnerabilities – 4.1.1
- Image Configuration Defects – 4.1.2
- Embedded malware – 4.1.3
- Embedded clear text secrets – 4.1.4
- Use of untrusted images – 4.1.5

Countermeasures in this section are as follows:
Organizations should use container specific technology for vulnerability and compliance outcomes that span across the entire lifecycle of images — from beginning of the build process to any applicable registries that the organization is using. Use of such pipeline-based methodologies will yield support to the immutable nature of containerized workloads and provide more actionable and reliable results. It is to be noted that performing vulnerability scanning directly on containers deployed to production is not recommended, unless it is performed via the use of independent security sensors deployed alongside production-deployed containers.

Organizations should be able to create “quality gates” at each stage of the build and deployment process, thereby implementing policy-driven enforcement. This can be achieved via integration with CI/CD pipelines to scan and detect vulnerabilities in images at testing and acceptance stages of the development lifecycle.

Vulnerability scan policies and incidents should be centralized across the organization and should provide flexible reporting and monitoring views in alignment with account jurisdictions, controlled access to development teams, and organizations’ business processes. The management aspect of policy design should allow for ongoing monitoring and maintenance of container repositories to ensure images within them are maintained and updated as vulnerabilities and configuration requirements change.

While checking for vulnerabilities the scanner technologies need to be able to scan the image, pull apart the layers and build a bill of materials to compare the content of the image manifest against the list of known vulnerabilities and report any matches. Selection of base layers must be from trusted sources along with an emphasis placed on use of container-specific host operating systems (OS) and packages, like CoreOS Container Linux, Snappy Ubuntu Core, Google Container-Optimized OS, Alpine Linux, and Windows Nano Server.
Ensure that configuration audit checks are in place to further limit management access to an image that is either stored and managed in a container registry or instantiated as a running container instance. Additionally, mitigate supply chain risks associated with malware and backdoors with use of trusted images that are sourced from sanctioned (enterprise controlled) container registries and ensure policies are in place to ensure consistent application of image tags.
Registry Risks and Countermeasures

Risks in this section are as follows:
- Insecure connections to registries – 4.2.1
- Stale images in registries – 4.2.2
- Insufficient authentication and authorization restrictions – 4.2.3

Countermeasures in this section are as follows:
Organizations should use a secure/encrypted connection when pushing/pulling from a registry, including the use of role-based access model and network restricted access to registries from trusted locations and service principals. Policy definitions should be implemented across all AWS accounts/Azure subscriptions/GCP projects that control exactly what images and registries are deemed trusted in the environment, so the images can be promoted from dev to test to production.

Generally, organizations should configure development tools, orchestrators, runtimes to enable mutual authentication, and encryption when connecting to container registries.

The security policy engine should have built-in capabilities that identify the use of SSH within containers, including policies that alert on the exposure of port 22 and processes that appear to be SSH.
daemons. Do not allow remote shell access through SSH or similar tools inside containers but use container orchestration tools (Swarm, Kubernetes, etc.) and/or container APIs to manage containers. Remote shell access can be hijacked by attackers and violates the principle of treating containers as immutable.

Operational practices should emphasize that image tag immutability should be a set property for every sanctioned container repository. This ensures that images are accessed using immutable names and that specify discrete versions of images to be used. This can be used to ensure that control mechanisms are in place to vet image content with an audit trail on each build and deployment artifact. This can also help prioritize identification of stale and potentially risky images within repositories.

Organizations should enable the appropriate cloud security posture management checks to test for authentication and authorization profiles mapped to container registries. Particularly, registries that are used to source sensitive application workload images should have stricter check-in and checkout guardrails that require images to be signed by the authorized personnel and promoted to a repository only after they have passed a thorough vulnerability scan and compliance assessment.

Organizations should integrate these automated scans into their processes to prevent the promotion and deployment of vulnerable or misconfigured images.
Orchestrator Risks and Countermeasures

Risks in this section are as follows:
- Unbounded administrative access – 4.3.1
- Unauthorized access – 4.3.2
- Poorly separated inter-container network traffic – 4.3.3
- Mixing of workload sensitivity – 4.3.4
- Orchestrator node trust – 4.3.5

Countermeasures in this section are as follows:
Organizations must leverage automated container orchestration tools to build, test, and deploy containers to production. Key functionality for container deployment is provided at the orchestration and scheduling layers. The orchestration layer interfaces with the application, keeping the containers running in the desired state and maintaining SLAs. Scheduling places the containers on the most optimal hosts in a cluster, as prescribed by the requirements of the orchestration layer.

Organizations should properly configure cluster RBAC roles and bindings to help minimize the impact of application compromises, user account takeovers, application bugs, or simple human mistakes. A given cluster’s RBAC configuration controls which subjects can execute which verbs on which resource types in which namespaces. By implementing principles, like least privilege access, one can significantly reduce the attack surface of Kubernetes native components, like the API server, etcd instance, and Kubelet.

Roles and Cluster Roles should be created and granted only to specific users and service accounts that need broad level of permissions. Access to cluster-wide administrative accounts should be tightly controlled as these accounts provide ability to affect all resources in the environment.

Additionally, specific cloud posture management checks should be put in place to restrict administrative access that can be used to manipulate orchestrator control plane components referenced earlier, such as enforced use of certificate-based authentication between the API server and the Kubelet’s HTTPS endpoints or detect misconfigurations that allow anonymous requests to the Kubernetes cluster API.
To prevent against the risk of unauthorized access, it is highly recommended to build a comprehensive activity audit trail of user and service principal activities in a cloud environment, regardless of container or orchestrator usage. These activity patterns should then be correlated by a user entity behavioral analytics (UEBA) engine. It either detects threats propagating across multiple cloud services at the same time or gets visibility into access related or service usage related anomalies, including attempted exfiltration of sensitive data to external cloud accounts and services. With the democratization of Security Operations functions across multiple product teams, mapping these anomalies, threats, and incident categories to a common framework, like the MITRE ATT&CK, provides a common taxonomy to streamline threat mitigation while being completely agnostic to the underlying architecture used by a cloud native workload.
### McAfee Guidance on Container Security Strategy Aligned to NIST SP 800-190

#### MITRE ATT&CK Matrix: Cloud

McAfee’s evaluation of Cloud Services against the ATT&CK matrix, Last 3 Calendar Months: Jan 1 - Aug 31 UTC

![MITRE ATT&CK Matrix](image)
Organizations should implement single sign-on to existing directory systems where applicable. Single sign-on simplifies the orchestrator authentication experience, makes it easier for users to use strong authentication credentials, and centralizes auditing of access, making anomaly detection more effective.

As an effective security practice, network segmentation should be managed and implemented across all container orchestrator deployments to decrease the overall risk posture, especially to apply container protection during runtime. From a configuration policy perspective, appropriate posture checks should enforce strict controls on inter-container communication. Inter-container network firewalls can be enabled where necessary to prevent cross-host and cross-container attacks within the same host.
At runtime, orchestrators should be configured to separate network traffic into discrete virtual networks by sensitivity level in a way that segregates apps open to internet access from other internal facing apps. This allows any communication between two different sensitivity contexts to occur through well-defined interfaces. Such protection can be applied at the application, process, and service levels to ensure running containers remain immutable to external threats while also providing a visual simulation of East-West and North-South communication flows and app segmentation policies.
Network traffic should be restricted within containers in AWS EKS cluster hosts.

By default, all network traffic is allowed between containers on the same host or the default network bridge. If not desired, restrict all inter-container communication. Link specific containers together that require communications. Alternatively, you can create custom networks and only join containers that need to communicate to that custom network.

- **Schedule**
  - Runs every 24 hours

- **Permissions**
  - Required Permissions:
    - eksClusters
Additionally, containers isolate applications at runtime primarily through OS kernel namespaces which provide a separate view of different system resources (IPC, network, mount, and PID). However, it is important to realize that container breakouts are real (in the sense that isolation is not as strong as VM isolation) and configuration mistakes can often allow processes in containers to interact with other containers and with the host. That is why it is recommended to ensure application containers of different tiers, threat posture, and sensitivity levels run on completely different VMs and physical hosts to provide additional defense in depth.

Finally, to build node level resilience in orchestrator platforms, policies should be enabled to enforce mutually authenticated connections between cluster members and end-to-end encryption of intra-cluster traffic. By implementing additional configuration audit checks on managed Kubernetes configurations as an example, validation can be performed across several Admission Controller and Scheduler best practices to ensure nodes and pods are securely introduced to the cluster. Because of the portability of containers, many deployments may occur across networks that organizations do not directly control, so a secure-by-default posture is particularly important to support orchestrator node trust.
Container Risks and Countermeasures

Risks in this section are as follows:
- Vulnerabilities within the runtime software – 4.4.1
- Unbounded network access from containers – 4.4.2
- Insecure runtime configurations – 4.4.3
- App Vulnerabilities – 4.4.4
- Rogue containers – 4.4.5

Countermeasures in this section are as follows:
To account for container runtime and application vulnerabilities, monitoring should be configured to alert or block malicious/unexpected actions based on the profile of a container, service, or process. By deploying runtime defense tools, organizations can use auto-discovery and fingerprinting techniques to identify and profile containers as they start up and tools to visualize container activity. By correlating with additional vulnerability scan data, known vulnerabilities can be proactively mitigated with stricter runtime policy checks, preventing any zero-day attacks from getting executed, such as any attempts to change container configuration at runtime or execution of any invalid or unexpected processes.

By applying proactive compliance checks, organizations can audit container engine configurations to prevent compromised container instances from accessing other containers on the same host. Docker implements several namespaces to effect isolation in Linux, such as PID process namespace, IPC access namespace, and MNT filesystem mount points, that should not be shared between the host and the hosted containers.

Additionally, design principles, like app or service level segmentation, can provide visibility into inter-container traffic patterns to detect any port scanning operations or unbounded network access attempts over both ‘on the wire’ traffic and encapsulated virtual network overlay traffic. This should also help support automated determination of proper container networking surfaces, including both inbound ports and process-port bindings.
**McAfee Guidance on Container Security Strategy Aligned to NIST SP 800-190**

---

### Security Configuration Policy

**Review And Activate Docker Host’s IPC namespace should not be shared**

**Description**

- **Name**: Docker Host’s IPC namespace should not be shared
- **Description**: IPC namespace debilitating applications of named shared memory segments, semaphore and message queues. IPC namespace on the interaction should not be shared with the containers and should remain isolated.

**Policy Category**: Container Security

**Rules & Exceptions**

- **Detect a high severity incident**
  - **IPC Mode**: Host

**Responses**

- **When violation of this policy occurs**
  - Generate an incident

---

### Security Configuration Audit

- **Policy Status**

<table>
<thead>
<tr>
<th>Policy Status</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td></td>
</tr>
</tbody>
</table>

- **Service Name**

  - Amazon Web Services
  - Microsoft Azure
  - Google Cloud Platform

- **Policy Type**

---

1. User should not start a container with the `-ipcs` option argument. For example, do not start a container as follows: `docker run --interactive --tty host:container bash`
To mitigate the risks associated with insecure runtime configurations, organizations should use tools or processes to continuously assess configurations settings across their containerized environments versus running compliance as point-in-time checks. In addition to hardening host configurations by using a stripped down, minimalistic base OS, organizations should consider taking advantage of Linux security extensions in production environments, including the use of Mandatory Access Control models supported by SELinux or via the use of AppArmor profiles, to every launched container to provide runtime protection coverage. These restrict the set of allowed system calls to ensure safe operation of containers including their ability to mount sensitive directories on the host file system (e.g., /boot, /sys/ or /etc access for Linux containers, C:\Windows for Windows containers).
Secure computing (Seccomp) profiles are another strong mechanism that can be used to constrain the system-level capabilities containers are allocated at runtime. Common container runtimes, like Docker, include default seccomp profiles that define what system calls can be made and the allowed arguments necessary for container operation. Custom profiles can also be created and passed to container runtimes to further expand such capabilities.
Host Risks and Countermeasures

Risks in this section are as follows:
- Large attack surface – 4.5.1
- Shared Kernel – 4.5.2
- Host OS component vulnerabilities – 4.5.3
- Improper User Access Rights – 4.5.4
- Host OS file system tampering – 4.5.5

Countermeasures in this section are as follows:
Organizations are highly recommended to use base host operating system distributions that are purpose-built to run container-based applications. These container-focused OSes are skinned down in terms of functionality and services supported natively, simple to update (can be patched on the fly), with read-only root file systems, and are integrated with Docker to minimize the system attack surface and simplify deployment and operations. This often requires a concerted effort to encourage development teams to work from thin base images, which requires them to work out their runtime dependencies.

By leveraging automated host vulnerability management and configuration management processes, security system teams can detect host level component vulnerabilities that can result in container breakout scenarios, such as host OS permissions to control access to control socket and daemon configuration files. Only trusted users should be allowed to control Docker daemon files and TLS authentication should be enabled to ensure clients are authenticated and authorized to interact with container components. System configurations should be maintained as templates within infrastructure as code systems, like Chef/Ansible/Puppet, to ensure consistency and auditability for every configuration change or component modification made to the underlying host OS. This is where additional server protection can also be implemented in the form of file integrity monitoring to lock down any changes to the host OS binaries/kernel or any of the file system and registry configurations.

In alignment with the NIST guidance, organizations must run containerized workloads only on designated hosts that do not support any non-containerized functions. These base OS layers need to be tagged appropriately to ensure they are safely used to standardize composition of other application container images.

By limiting management access to containers or hosts, user access rights can be fully streamlined or if necessary be provisioned in a Just-in-Time capacity. At all times teams must ensure that containers are run with minimal set of file system permissions required. Container jobs should be distributed across hosts using the native orchestrator components, like scheduler and admission controllers. These clusters should be appropriately configured to prevent privileged container operations and have runtime activity monitoring capabilities in place to detect anomalous access patterns.

Containers should also be run with their root filesystems in read-only mode. Very rarely should containers mount local file systems on a host. Instead, any file changes that containers need to persist to disk should be made within storage volumes specifically allocated for this purpose.
This approach isolates writes to specifically defined directories, which can then be more easily monitored by monitoring contain behaviors at runtime. By layering cloud configuration posture checks across all container deployments on top of runtime protection capabilities that get automatically instantiated at scale, organizations can enforce the principle of immutable infrastructure across all the stages an application container lifecycle. Running containers with a minimal set of file system permissions significantly reduces the attack surface.
Key Takeaways

Security has long been an afterthought — widely perceived as the last gate before deploying new application workloads. Changes in the infrastructure of the cloud-native development stack, including containers and orchestrators, are transforming the security landscape. As a result, they are driving the need to employ security best practices and standards, such as NIST SP 800-190.

The advent of declarative and immutable container-based workloads has paved the way for an entirely new operational model for security use cases that apply across all the stages of a container lifecycle. This requires organizations to adopt new tools and processes that augment and elevate traditional security implementation models.

By incorporating technologies, such as MVISION CNAPP, security teams have the opportunity and responsibility to help developers build good security principles into their assets early into their development lifecycle. MVISION CNAPP protects containerized applications across the entire container life cycle by enabling discovery and security posture management of all the cloud-native components mapped to an application including hosts, IaaS/PaaS services, containers, and the orchestration context that a container operates within. By layering additional runtime and host-level controls, visibility, best practices, and security policies can be enforced across the entire stack.
About McAfee

McAfee is the device-to-cloud cybersecurity company. Inspired by the power of working together, McAfee creates business and consumer solutions that make our world a safer place. By building solutions that work with other companies’ products, McAfee helps businesses orchestrate cyber environments that are truly integrated, where protection, detection, and correction of threats happen simultaneously and collaboratively. By protecting consumers across all their devices, McAfee secures their digital lifestyle at home and away. By working with other security players, McAfee is leading the effort to unite against cybercriminals for the benefit of all.