cloud security alliance®

Best Practices for Implementing a Secure Application Container Architecture

Integrating Application Container Security Considerations into the Engineering of Trustworthy Secure Systems

ABSTRACT

Application containers and a microservices architecture are being used to design, develop, and deploy applications leveraging agile software development approaches such as DevOps. Security needs to be embedded into these software development approaches. This document identifies recommendations and best practices to address the challenges in securing application containers in the engineering of trustworthy secure systems, through the lenses of Developers, Operators, and Architects.

ACKNOWLEDGEMENTS

Application Containers and Microservices Working Group Co-chairs:

Anil Karmel
Andrew Wild

Lead Authors:

John Kinsella
Cem Gurkok
Frank Geck

Contributors:

Jeff Barnes
Randall Brooks
Ramaswamy Chandramouli
Madhav Chablani
Atul Chaturvedi
Aradhna Chetal
Joshua Cuellar
Joshua Daniel
Shyamkant Dhamke
Michele Drgon
Michaela Iorga
Frank Geck
Michael Green
Cem Gurkok
Amir Jerbi
John Kinsella
Juanita Koilpillai
Yin Lee
Aaron Lippold
Vishwas Manral
James McCloskey
Ki-Hong Min
Lloyd Osafo
Mark Potter
Alex Rebo
Michael Roza
Ed Santiago
Kina Shah
Shankar
Ken Stavinoha
Shanthi Thomas
David Wayland
Shawn Wells
John Wrobel
Mark Yanalitis
John Osborne
Ashish Kurmi
James Yaple
Vrettos Moulos

CSA Staff:

Hillary Baron
Marina Bregu

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Application containers and a microservices architecture, as defined in NIST SP 800-180, are being used to design, develop, and deploy applications leveraging agile software development approaches such as DevOps. The security of application components needs to be considered throughout the software development life cycle (SDLC). NIST 800-160, Systems Security Engineering, defines the need for trustworthy secure systems based on a wide variety of stakeholder needs.

This document is intended to be a companion document to Challenges in Securing Application Containers and Microservices, which identifies challenges in securing application containers and microservices through the lenses of the Developer, Operator, and Architect. This document provides recommendations and best practices to address those challenges.

Recommendations and best practices were developed through extensive collaboration among a diverse group with strong knowledge and practical experience in information security, operations, application containers, and microservices. The recommendations and best practices contained herein are intended for Developer, Operator and Architect audiences.

1 INTRODUCTION

1.1 Purpose and Scope

The purpose of this document is to present best practices and recommendations for implementing a secure application container architecture. The guidance is intended to apply to a variety of roles, including the Developer, Operator, and Architect. Secure DevOps necessitates that security, Developers, and Operators work together to engineer trustworthy secure systems.

The scope of this document is limited to application containers.

1.2 Document Structure

The remainder of this document is organized into the following sections and appendices:

- Section 2 introduces application containers
- Section 3 provides a summary of recommendations and best practices to address the challenges of securing application containers.
- Appendix A provides common acronyms and abbreviations used throughout this document.
- Appendix B provides a glossary of selected terms from this document.
- Appendix C provides a list of references cited in the document.

1.3 Audience

The intended audience for this document is application Developers, application Architects, system and security administrators, security program managers, information system security officers, and others who have responsibilities for or are otherwise interested in the security of application containers in a software development life cycle (SDLC).

This document assumes that readers have some operating system, networking, and security expertise, as well as expertise with application containers, microservices and agile application development approaches such as DevOps. Because of the constantly changing nature of application container technologies, readers are encouraged to take advantage of other resources, including those listed in this document, for more current and detailed information.

2 APPLICATION CONTAINERS

As detailed in *Challenges in Securing Application Containers and Microservices*, application containers and microservices have unique characteristics with distinct security ramifications based on the stakeholder responsible to secure the same. To that end, challenges for application container and microservices features have been identified for Developers, Operators, and Architects (for microservices). Additional challenges for which these best practices are applicable are listed in the additional challenges column.

Table 1 – Application Container Best Practices

Best Practice	Viewpoint
3.1.1 Code Promotion Across Environments (Development, Quality Assurance, Test, Production)	
Developers should establish a root of trust.	Developer
Operators should leverage a container management platform to securely move images across environments.	Operator
3.1.2 Securing the Host	
Multi-tenancy or data sensitivity requirements must be documented through container technologies (e.g., labels) to allow automation of placement and audit of containers.	Operator
Container runtime must only engage in encrypted communication with remote services (registry, orchestration).	Operator
Unless absolutely necessary, containers must not run as root or in privileged mode.	Operator
Container runtimes should only mount data volumes inside containers unless absolutely necessary (e.g., do not mount /proc, /etc, runtime sockets, etc.).	Operator
Users should remove all capabilities except those explicitly required for their processes. Use a whitelist instead of blacklist approach.	Operator
3.1.3 Container Continuous Monitoring from the Platform/Host	
Developers need to use clear versioning schemes to identify application versions running in containers.	Developer
For the Operator, container runtimes should be configured to emit logs to centralized logging systems.	Operator
For the Operator, monitoring service containers must not run in privileged mode or access the host.	Operator

3.1.4 Container Networking - Communications Between Host and Container	
It is recommended that applications use secure network communication protocols.	Developer/ Operator
The Operator should provide authentication and authorization infrastructure.	Operator
The Operator should also provide network monitoring infrastructure.	Operator
3.1.5 Container Networking - Communications Between Containers	
Applications must use secure network communication protocols.	Developer/Operator
The Operator must provide authentication and authorization infrastructure.	Operator
The Operator must provide network monitoring infrastructure.	Operator
The Operator must use containerization solutions that support granular access control for exposed services.	Operator
3.1.6 Validate Integrity and Security Quality of the Image	
The Developer should sign images as part of the image build process and validate images before use.	Developer
The Developer should use vulnerability scanning tools as part of the development process.	Developer/ Operator
The Developer should include only necessary components inside the image.	Developer
The Operator should design into infrastructure the ability to only allow the use of signed and approved images.	Operator
The Operator should design the infrastructure to continuously identify newly published vulnerabilities.	Operator
3.1.7 Container Forensics	
For new applications, the Developer should create and follow coding policies to include logging capabilities at the planning and design stages.	Developer
For existing applications, the Developer should implement plans to capture application logs, starting with authentication logs.	Developer

The Operator should include forensics capability planning for container environments.	Operator
For new infrastructure, the Operator should implement planned forensics capabilities.	Operator
For existing infrastructure, the Operator should manage a gradual roll-out of capabilities, starting with capturing data (i.e., network traffic, disk and memory artifacts).	Operator
3.1.8 Trust Chain Through Containers	
The Developer should ensure image signing as part of the image build process and should validate images before use.	Developer/ Operator
Developers should use vulnerability scanning tools as part of the development process.	Developer/ Operator
Developers should make sure only necessary components are included inside the image.	Developer
Leverage 3.1.6 (Validate Integrity and Security Quality of the Image) for Operator recommendations on designing infrastructure for image integrity validation, vulnerability scanning, and patching.	Operator
Operators should use trusted hosts that are securely hardened and, where container engine is securely configured, to store images and run containers.	Operator
Operators should use authentication and authorization mechanisms (i.e., TLS-based mutual authentication) to ensure trust between the containers themselves and between the management platform and containers.	Operator
Operators should limit access to containers and images based on user roles.	Operator
Operators should monitor container configuration parameters and runtime integrity.	Operator
3.1.9 Container Volume Management	
Developers should receive sufficient training to ensure development of applications that minimize the need to use shared container volumes and that do not require any access to host directories.	Developer

The Operator should provide infrastructure that supports resource management, access control, and monitoring of container volumes.	Operator
The Operator should provide infrastructure that isolates network traffic of different users and applications.	Operator
The Operator should provide an interface to allow network traffic to communicate between containers owned by an entity (e.g., workload).	Operator
3.1.10 Container Secret Management	
Provide training and best practices guidance to Developers.	Developer
Create common libraries for Developers to handle sensitive data and secrets in the backend application code.	Developer
Operators should design the secret management architecture by integrating with deployment tools or scripts and container orchestration tools.	Operator
3.1.11 Platform Management - Notification of Lifecycle Events	
The container's lifecycle (start/stop/scaled) is managed by CMP. From a Developer's perspective, the application could be left in an insecure state if a containerized application is unaware of the container's transitions.	Developer
3.1.12 Platform Management - Resource Request	
The Developer should help ensure sustainable system performance by balancing system resources.	Developer
In a multi-tenant environment, the Operator should ensure balanced resource consumption.	Developer
In a multi-tenant environment, the Operator should ensure that the CMP optimizes cluster resource consumption.	Operator
3.1.13 Platform Management - Container Resource Management	
The Operator should find and achieve an acceptable balance between the CMP and individual application resource management objectives.	Operator

3.1.14 Container Management - Scaling Container Resources	
When creating container deployment configurations, the Developer should utilize resource control features of their CMP to orchestrate intra-container resource utilization, prioritization and allocation thresholds.	Developer
Operators need to work with Developers to understand the resource needs of containerized applications, set constraints as appropriate, and monitor for performance challenges.	Operator
3.1.15 Container Management - Data Backups and Replication	
The Operator must ensure data backup systems can back up any new data within a container before it is destroyed.	Operator
3.1.16 Container Management - Container Rehosting Between CMPs	
For an Operator of legacy applications that use sensitive data, some work will need to be done to create and manage utilities that automate the exporting and importing of sensitive data related to a container that is being rehosted.	Operator
3.1.17 Container Encryption	
Developers and Operators should agree to use standard, commonly available authentication systems.	Developer/ Operator
Developers need to identify what data at rest (DAR) solution they will utilize, taking into account if the solution has been validated by an impartial third party and if the solution is viable in the intended operational environment.	Developer
For sensitive applications, Developers should encrypt the sensitive application, then create a container entrypoint application that decrypts and executes the main application.	Developer

3 APPLICATION CONTAINER MITIGATIONS FOR CHALLENGES

3.1.1 Code Promotion Across Environments

Proposed mitigations for challenges inherent in code promotion across environments (development, quality assurance, test, production) comprise the following recommendations:

1. Developers should establish a root of trust.
A public key infrastructure (PKI) allows creation and management of a hierarchy of digital certificates for encryption and digital signatures. A PKI may already exist in an organization—certificates from this new or existing system can be used to sign container images and later verify their provenance and integrity.

To ensure the integrity of the build chain, the Developer must have the ability to digitally sign and provide a digital signature for the code and/or binary artifacts that will be promoted between environments or to the next promotion phase.

By signing a container image with a certificate that is part of a PKI, container management platform (CMP) users can easily verify the source of an image. This is important, as once the source of the image is determined, users can ensure that the image has been through appropriate scrutiny before being promoted for production use.

2. Operators should leverage a container management platform to securely move images across environments.
In a production container environment, there are many components to manage in addition to management of application code and container images. A CMP that can support the organization's scale across one or more environments and/or locations should have capabilities to securely synchronize container images.

To ensure the integrity of the build chain, the Operator must have the ability to validate and verify the code and/or binary artifacts between environments and promotion phases utilizing digital certificates and/or digital signatures provided by the Developer or the orchestration engine.

To ensure the consistency of an application deployed by the build chain, the Operator must also have the ability to synchronize the application code across all locations where the application is running or being consumed.

By using a CMP to manage container images, users of the platform can confidently expect appropriate versions of a container image to be available for execution across their environments. This minimizes confusion around versions of an application that is running in different environments. When coupled with the best practice of using a root of trust, users of the platform can ensure provenance and integrity of the container images across their environments.

3.1.2 Securing the Host

Proposed mitigations for challenges present in securing the host comprise the following Operator viewpoints:

1. Multi-tenancy or data sensitivity requirements must be documented through container technologies (e.g., labels) to allow automation of placement and audit of containers.
Labels are important to classify the type of data stored within or processed by a container or set of containers. These labels can be used by a CMP to automate placement of containers into environments that match the security profile of the data stored therein. The labels also serve the function of allowing an auditing engine to validate data within a container.

By applying labels to containers, a CMP can ensure a container is placed on a cluster of hosts that has the appropriate security to protect the data stored in said container.

Labels can provide context to an auditing engine and inform the interrogation of a host's security posture to ensure protection of the data processed by a container hosted on the same.

2. Container runtime must only engage in encrypted communication with remote services (registry, orchestration).
Any container ecosystem that consists of more than a single host will require communication between nodes to manage containers and resources. Encryption of network traffic ensures confidentiality of this traffic. Some container runtimes now have encryption enabled by default; others do not but still provide support for encrypted traffic.

While the security of the application's network traffic is left as a challenge to the Developer, providing a secure container hosting platform requires the Operator to ensure communication within the CMP is not subject to manipulation by a malicious entity. Encrypting management communications addresses this challenge.

3. Unless absolutely necessary, containers must not run in privileged mode.
Granting administrator-level privileges to a container essentially grants the container administrator-level access to the container host. This requires the privileged container to be ultimately trusted by the host and all other containers that may run on that host. Such grants must be requested to the container runtime at container start; some CMPs are now disallowing privileged containers by default.

By disallowing such privilege grants the administrator can mitigate the risk of a privileged container being compromised and thus the rest of the host and its guest containers. By disallowing by default, an organization is forced to consider the risk/reward of running a particular container in privileged mode.

4. Container runtimes should only mount data volumes inside containers when absolutely necessary (e.g., do not mount /proc, /etc, runtime sockets, etc.).
As a partial mitigation to some containers that would otherwise require privileged access, some application containers are designed to instead require read and/or read/write access to files or directories mounted in the container from the host system. While this is somewhat better than granting the container administrative privileges, this method has been shown to have loopholes that would allow a malicious user to still gain unintended access to the host system.

By disallowing access to components of the host filesystem and services, the security of the host is ensured through separation of host/container resources.

5. Users should remove all capabilities except those explicitly required for their processes. Use a whitelist instead of blacklist approach.
By default, containers start with a restricted set of Linux kernel capabilities. Using these capabilities, the processes do not have to run as root for almost all the specific areas where root privileges are usually needed. Containers support overriding the defaults by adding or removing capabilities, which can make containers less secure through addition of capabilities.

Restricting Linux kernel capabilities (which does not require process to be run as root) where root privileges are usually needed reduces the attack surface and keeps the host safe.

In "legacy" host hardening guides, the Operator understands which applications are running on the host as well as the parameters to be reconfigured to reach a hardened status. For an application container host, the Operator does not necessarily know ahead of time the application running on the host, so the hardening process must consider application containers that could run on the host and the resources they would legitimately consume.

3.1.3 Container Continuous Monitoring from the Platform/Host

In container continuous monitoring from the platform to the host, these proposed mitigations for the challenges are recommended:

1. Developers need to use clear versioning schemes to identify application versions running in containers.
By defining and using an accepted versioning scheme, a development team can have version numbers automatically implemented as part of a build process. By applying this version either to a container name, tag, or label, the Developer is then able to easily identify application versions running in containers.

2. For the Operator, container runtimes should be configured to emit logs to centralized logging systems.
The temporary nature of containers heightens the need for centralized logging; when a container is terminated (either due to a fault or purpose), CMPs quickly clean up container artifacts from the host. In order to view logs both while the container is running and after its exit, logs need to be shipped and stored in a secure centralized location.

Both audit and application logs from a container can provide troubleshooting and forensic value while the container is running as well as after it has terminated. There exists a need to trust these logs, as they may be the only indicators remaining after container termination. Because of this need, these logs need to be stored in a secure, trustable manner.

As applications are made up of more granular components, troubleshooting can become more difficult for the Operator or Developer. The Operator also needs to provide all logs in a consumable manner with a consistent timestamp.

By providing a centralized logging system with a secure means of storage, the Operator enables both secure log storage and a means to retrieve and/or review application or audit logs for monitoring, troubleshooting, or forensics purposes.

3. For the Operator, monitoring service containers must not run in privileged mode or access the host.
By default, container runtimes do not grant a running container full access to the host and other containers. In some cases, users or third parties will suggest running their containers in "privileged" mode so they can easily access system resources. The risk in doing so is that the container may gain unintended access to resources or, if the container is compromised, a malicious actor will gain access to the host and thus other containers and resources.

Two tenants on a CSP may be different departments within an organization or two different customers in a public cloud, for example. In any scenario, it is imperative that the tenants do not have access to each other's data or metadata. This provides a challenge for the Operator to ensure that privileged containers cannot run in a multi-tenant environment, as they would allow access across tenants. Additionally, containers running with root privileges may provide security concerns in single-tenant scenarios.

By disallowing privilege grants, the administrator can mitigate the risk of a privileged container making unauthorized access of resources or data. When disallowing by default, an organization is forced to consider the risk and/or reward of running a particular container in privileged mode.

3.1.4 Container Networking - Communications between Host and Container

Proposed mitigations for challenges in communications between host and container include the following recommendations:

1. It is recommended that applications use secure network communication protocols.
The network traffic generated by the containers should maintain confidentiality regarding the applications and management by utilizing proper encryption protocols (i.e., SSL/TLS, IPSec).

Selecting libraries that support secure communications to provide encryption and maintain confidentiality prevents information spillage and network traffic tampering. Tampering of network traffic may result in system compromise.

2. The Operator should provide authentication and authorization infrastructure.
The Operator should provide the means to not only encrypt traffic, but also ensure the identity of the actors and the actions that they can take.

Using a solution such as TLS with mutual authentication would ensure both the confidentiality of the traffic and the identity of the actors. The authorization mechanisms would be provided based on a role-based access control (RBAC) such as LDAP, which leverages the identity provided by the authentication mechanism.

3. The Operator should also provide network monitoring infrastructure.
Monitoring network traffic and configurations provides the means to detect tampering of network traffic and configurations besides potentially malicious traffic.

Providing a record of network traffic (i.e., netflows) and configuration changes ensures that only expected behaviors and actions take place within a given environment. This helps in maintaining incident response and detection capabilities.

3.1.5 Container Networking - Communications Between Containers

Proposed mitigations for challenges present in communications between containers comprise the following recommendations:

1. Applications must use secure network communication protocols.
Network traffic generated by the containers should maintain confidentiality regarding the applications and management by utilizing proper encryption protocols (i.e., SSL/TLS, IPSec).

Selecting libraries that support secure communications to provide encryption and maintain confidentiality prevents information spillage and network traffic tampering. Tampering of network traffic may result in system compromise.

2. The Operator must provide authentication and authorization infrastructure.
The Operator should provide the means to not only encrypt traffic, but at the same time ensure the identity of the containers and the actions that they can take.

Using a solution such as TLS with mutual authentication would ensure both the confidentiality of the traffic and the identity of the containers. Authorization mechanisms would be provided based on an RBAC such as LDAP, which leverages the identity provided by the authentication mechanism.

3. The Operator must provide network monitoring infrastructure.
Monitoring network traffic and configurations provides the means to detect the tampering of network traffic and configurations besides potentially malicious traffic.

Network monitoring challenges may be mitigated by defining standard semantics and syntax for network monitoring for containers, specifying “must report” events for compliance, and addressing the accuracy or relevance of network activity/events to network health for containers.

Providing a record of network traffic (i.e., netflows) and configuration changes ensures that only expected behaviors and actions take place within a given environment. This helps in maintaining incident response and detection capabilities.

4. The Operator must use containerization solutions that support granular access control for exposed services.

The Operator should provide the means to support granular access control for exposed services. Using a solution such as granular access control to containers would reduce security risks.

3.1.6 Validate Integrity and Security Quality of the Image

Proposed mitigations for challenges present in validating the integrity and security quality of the image comprise the following:

1. The Developer should sign images as part of the image build process and validate images before use.
Images should be signed as part of the build process and validated before used. Signing and validation can be achieved through GNU Privacy Guard (GPG) signatures on image contents or through similar methods.

Digital signing of image content at build time and validation of the signed data before usage ensures that image data cannot be tampered with between build and run time.

2. The Developer should use vulnerability scanning tools as part of the development process.
Developers should use vulnerability scanning tools as part of the development process and the CI pipeline and integrate a vulnerability assessment into the build process. They will also want to consider failing a build if the vulnerability assessment fails. Once a vulnerability is identified, apply security patches to the vulnerable component and rebuild the image.

A vulnerability scanner will identify and alert on the use of third-party components with known security vulnerabilities. Applying vulnerability scanning as part of the development cycle improves the security quality of the images, since well-known vulnerabilities are identified and patched before software gets into runtime environment.

3. The Developer should include only necessary components inside the image.
Developers should trim down images to include only necessary components. It is preferred to use a baseline image with minimal set of packages over a full OS distribution.

Removing unnecessary components from the image will reduce the number of potentially outdated or unpatched packages and therefore reduce the number of security vulnerabilities.

4. The Operator should design into infrastructure the ability to only allow the use of signed and approved images.

Operators should design an infrastructure that will accept only signed and approved images. The image approval can be tied to the organization's software acceptance policy (e.g., quality, security, stability, etc.), or to any acceptance criteria that is applicable. It should be possible for an Operator to prevent running an image even if it is properly signed but otherwise fails to meet acceptance criteria.

Checking for signature validation ensures that image data was not tampered with between build and the time image is running. Preventing the use of unauthorized or unacceptable images (even if signed) provides further validation of images before they run.

5. The Operator should design the infrastructure to continuously identify newly published vulnerabilities.

The Operator should provide an infrastructure that continuously identifies image vulnerabilities in the following cases: images that are stored in a registry, images that are stored locally on the host, and images that are running as containers. Once a new vulnerability is identified, mitigation actions should take place based on the vulnerability severity and its impact on the application. Instead of patching running containers, it is preferable to apply the patch on a new image version and deploy the new image to the container runtime environment.

There is a need to continuously rescan images since new vulnerabilities are found and published over time. The scan should include all the images in the image registry, images on local machines (waiting to run), and images that are already running as containers. Applying patches by creating a new image version instead of updating running containers will provide better visibility into the process and ensure consistency between containers and their images.

3.1.7 Container Forensics

Proposed mitigations for challenges present in container forensics comprise the following recommendations:

1. For new applications, the Developer should create and follow coding policies to include logging capabilities at the planning and design stages.
An application should provide logs regarding authentication, authorization, actions, and failures. The Developer should include this capability as part of planning and design phases.

Logging application authentication, authorization, actions, and failures provides a trail of evidence to follow when an investigation takes place and a root cause needs to be established.

2. For existing applications, the Developer should implement plans to capture application logs, starting with authentication logs.

An existing application should provide logs regarding authentication, authorization, actions, and failures. If any of these logging items have not been implemented, Developers should provide these capabilities as part of the maintenance phase.

Logging application authentication, authorization, actions, and failures provides a trail of evidence to follow when an investigation takes place and a root cause needs to be established.

3. The Operator should include forensics capability planning for container environments.
Forensics capabilities are the basis for any incident response and mitigation activity. Forensics can provide the necessary evidence to determine the root cause of an incident and increase the speed of mitigation. Forensics planning should include the instrumentation, human resources, sources of evidence (i.e., network, disk, memory), processes, and procedures. The volatile nature of the container environment necessitates a more agile framework to capture and analyze the evidence.

Integrating forensics capabilities into an incident response plan and procedures will provide the means to acquire and process evidence, decrease the time to determine root cause, and minimize exposure to a compromise.

4. For new infrastructure, the Operator should implement planned forensics capabilities.
The design and implementation of planned forensics capabilities should take place along with the rest of the infrastructure and should be integrated with corporate incident response, business continuity and disaster recovery plans.

The design and implementation of forensics capabilities will help to verify the plan and revise the designs if needed to accommodate changes. These capabilities will provide the means to acquire and process evidence, decrease the time to determine root cause, and minimize exposure to a compromise.

5. For existing infrastructure, the Operator should manage a gradual roll-out of capabilities, starting with capturing data (i.e., network traffic, disk and memory artifacts).

Integrating forensics capabilities into an existing infrastructure should be part of the maintenance and change management activities of the Operator. This should be done as part of due diligence and compliance, as well as to satisfy rules and regulations.

The integration of forensics capabilities will help the Operator meet due diligence requirements, compliance measures, rules, and regulations. These capabilities will provide the means to acquire and process evidence, decrease the time to determine root cause, and minimize exposure to a compromise.

3.1.8 Trust Chain Through Containers

The following are proposed mitigations for the securing the trust chain through containers:

1. The Developer should ensure image signing as part of the image build process and should validate images before use.
Images should be signed as part of the build process and validated before used. Signing and validation can be achieved by using GPG signatures on image contents or through similar methods. Developers should ensure that their images are signed as part of the build process. Operators should ensure the availability of the capability to validate image signatures before deployment.

Digital signing of image content at build time and validation of the signed data before usage ensures that image data cannot be tampered with between build and runtime.

2. Developers should use vulnerability scanning tools as part of the development process.
Developers should use vulnerability scanning tools as part of the development process and the CI pipeline, as well as integrate the vulnerability assessment into the build process. They may also consider failing a build if the vulnerability assessment fails. Once a vulnerability is identified, they should apply security patches to the vulnerable component and rebuild the image.

A vulnerability scanner will identify and alert on the use of third-party components with known security vulnerabilities. Applying vulnerability scanning as part of the development cycle improves the security quality of the images, since well-known vulnerabilities are identified and patched before software gets into runtime environment.

3. Developers should make sure only necessary components are included inside the image.
Developers should trim down images to include only necessary components. It is preferred to use a baseline image with minimal set of packages.

Removing unnecessary components from the image will reduce the number of potentially outdated or unpatched packages and therefore reduce the number of security vulnerabilities.

4. Leverage 3.1.6 (Validate Integrity and Security Quality of the Image) for Operator recommendations on designing infrastructure for image integrity validation, vulnerability scanning, and patching.

5. Operators should use trusted hosts that are securely hardened and, where container engine is securely configured, to store images and run containers.
Operators will need to harden the OS and the container engine before storing images and running containers on them. It is important to only deploy the necessary packages on host and to regularly apply security updates and kernel patches. Operators will want to harden the container engine according to vendor's security best practices.

Ensuring that host OS and container engine are properly patched and securely configured will reduce the container's attack surface and exposure to known vulnerabilities.

6. Operators should use authentication and authorization mechanisms (i.e., TLS-based mutual authentication) to ensure trust between the containers themselves and between the management platform and containers.
Configure the container management platform and container endpoints with mutual authentication between endpoints (e.g., mutual TLS authentication). In case mutual authentication is not supported, consider using the container ambassador pattern to create a secured proxy communication.

It is recommended to apply mutual authentication on all communications to prevent data tampering, man-in-the-middle attacks, and disclosure of sensitive information.

7. Operators should limit access to containers and images based on user roles.
Operators should apply user access control mechanisms to ensure that only authenticated and authorized users have access to containers and images. Applying user access control mechanisms will prevent unauthorized users from accessing images and containers.

8. Operators should monitor container configuration parameters and runtime integrity.
Operators should monitor containers configuration for non-secure settings and should check the integrity of configuration files and processes in running containers.

Running containers with non-secure settings can gain unlimited access to host and network resources. A change in container processes and configuration files can indicate that the container has been tampered.

3.1.9 Container Volume Management

Proposed mitigations for challenges present in container volume management comprise the following recommendations:

1. Developers should receive sufficient training to ensure development of applications that minimize the need to use shared container volumes and that do not require any access to host directories.

2. The Operator should provide infrastructure that supports resource management, access control, and monitoring of container volumes.
The management of volume resources includes following volume management service activities:

- Creating, provisioning, and updating volume resource
- Enrolling, accessing, and denying volume resource
- Decommissioning and terminating volume resource
- Troubleshooting volume resource
- Monitoring, reporting, auditing, inventorying, validating, and verifying volume resource
- Migrating volume resource
- Encrypting volume resource
- Recovering volume resource
- Performance tuning volume resource

The relationship between assets (i.e., people, groups, containers, network, host, and volumes) is explicitly known and mapped to volume management service activities:

- Container access mapping to data volume
  - Asset container (x)...
    - is granted access (x) to the data volume
    - Is granted access full if volume is on same VM host

- Role access mapping to data and operations volume:
  - Volume role:
    - Data volume(s) are always mapped to dedicated storage
    - Runtime operations volume
  - Group Role:
    - Developer role ...
      * is granted access (x) to the data volume
      * is granted access (x) to the operations volume
    - Operator role ...
      * is granted access (x) to the data volume
      * is granted access (x) to the operations volume
    - User role ...
      * is granted access (x) to the data volume
      * is granted access (x) to the operations volume

The granular breakdown of "resources management" is concretely identified as a potential standard practice for consistent management of volume resource management activities.

3. The Operator should provide infrastructure that isolates network traffic of different users and applications.
Network traffic for different users and applications needs to be segregated to limit lateral movement from potentially compromised users or applications. This network isolation limits the threat vector of lateral movement.

4. The Operator should provide an interface to allow network traffic to communicate between containers owned by an entity (e.g., workload).
An interface is required to ensure containers than comprise an entity (e.g., workload) can effectively communicate with each other. This interface can be monitored by an external entity to ensure it isn't tampered with.

An interface coupled with a monitoring platform ensures containers owned by an entity (e.g., workload) can communicate yet be isolated from other users and applications.

3.1.10 Container Secret Management

In container secret management, these proposed or adopted mitigations for the challenges are recommended:

1. Provide training and best practices guidance to Developers.
Backend Developers should be informed and trained on threats arising from multi-tenancy on public clouds. They also need to be trained and provided with templates for how to use the secret management feature within their apps so they avoid the practice of hard-coding static secrets within application code.

Backend Developers are accustomed to the practice of baking sensitive information into server code from the days when server-side was considered trusted environment. Raising the awareness of the nuances and threats present in current public cloud environments helps drive in the need to avoid such practices.

2. Create common libraries for Developers to handle sensitive data and secrets in the backend application code.
To enable consistent ways of addressing the handling of sensitive data and secrets within backend application code, a common set of libraries should be provided to Developers.

Application Developers are not necessarily security Developers and should devote time on developing application features, not security. Security that is required within applications should be easy to use and made clear, so minimum effort is spent and consistency across applications is achieved.

3. Operators should design the secret management architecture by integrating with deployment tools or scripts and container orchestration tools.
Secret management deployment and initial seeding utilizing deployment time and user inputs can be handled by closely integrating it with the specific container orchestration scripts, as well as the corresponding deployment scripts in scenarios where deployment is triggered by a human user. In the

case of a completely automated deployment, the use of a key management server (KMS) or a hardware secure module (HSM) should be utilized to bootstrap the secret management server with the initial bootstrapping secrets required. For instance, virtual machine (VM) metadata that has been bound to a specific role can be used to authenticate to an HSM to obtain secrets from an HSM during VM boot up.

Bootstrapping of secrets is always a chicken-and-egg issue and utilizing some auto-generated data, such as VM metadata, to obtain the first secret from the HSM/KMS is a way to overcome this issue.

Once the secret management system has been bootstrapped with deployment-time or boot-up time secrets, it is armed to handle dynamic generation of secrets. For instance, in order to dynamically generate database account credentials for applications it needs to use the database admin credentials obtained at deployment time to authorize the creation of a new user account and then pass these on to the application that needs to use it.

3.1.11 Platform Management - Notification of Lifecycle Events

Proposed mitigations for challenges present in platform management, as it applies to the notification of lifecycle events, comprise the following recommendations:

1. TThe container's lifecycle (start/stop/scaled) is managed by CMP. From a Developer's perspective, the application could be left in an unknown state if a containerized application is unaware of the container's transitions.
The CMP should provide the encapsulated application an opportunity to perform a graceful transition to a known secure state. The conventional solution is to allow an application to be notified of the container lifecycle events. Resources, freed after container's removal, should be released back to the pool in a well-known state and containers should be allowed to log any state changing events.

It is essential to inform the application of the container's lifecycle events, so that it may take appropriate action to ensure a secure startup and shutdown.

3.1.12 Platform Management - Resource Request

The following proposed mitigations to the challenges are outlined below:

1. The Developer should help ensure sustainable system performance by balancing system resources.
Working with the Operator and Architect, the Developer should ensure that the binaries and libraries are optimized to operate in a containerized infrastructure.

2. In a multi-tenant environment, the Operator should ensure balanced resource consumption.
Quality of service should always be upheld at the expense of multi-tenancy, which allows fulfillment of at least one service-level agreement (SLA). It can be beneficial to work with Developers and/or vendors to refine resource requests.

3. In a multi-tenant environment, the Operator should ensure that the CMP optimizes cluster resource consumption.
Balancing the resource consumption facilitates sustainable execution, which in turn minimizes the resource contention.

3.1.13 Platform Management - Container Resource Management

The following proposed mitigations for the challenges are recommended:

1. The Operator should find and achieve an acceptable balance between the CMP and individual application resource management objectives.

The CMP should uphold an application-formulated resource management policy. This affirmation results in a binding SLA.

The CMP should attempt to allocate predetermined resources during a set time interval. In turn, the application agrees to use these resources in a predefined fashion during said time interval. There are clauses that a party can take to address a set of predetermined actions in case of a violation (including unconditional application termination).

3.1.14 Container Management - Scaling Container Resources

Proposed mitigations for challenges present in container management as they apply to scaling container resources comprise the following recommendations:

1. When creating container deployment configurations, the Developer or Operator should utilize resource control features of their CMP to orchestrate intra-container resource utilization, prioritization and allocation thresholds.

The application Developer must accept that the application under development will not have on-hand the entirety of computing resources available in the production environment. Even in the presence of auto-scale, auto-scaling capability has limits and does not scale out infinitely. Developers should expect the presence of horizontal infrastructure scaling capability but should avoid the need for vertical scaling by ensuring that the application can function within the parameters established by the configuration management team.

The Developer must obtain the resource utilization and hardening parameters and incorporate those configurations into the software development environment and testing scripts. Without testing application functions within defined production parameters, the risk of poor application performance goes up, not down. Proper configuration of resource and security functionality as appropriate to the CMP in use ensures that a resource consumption fault or runaway container processes do not affect adjacent containers.

2. Operators need to work with Developers to understand the resource needs of containerized applications, set constraints as appropriate, and monitor for performance challenges.

By working together, Developers and Operators can gain a combined understanding of the resource requirements for an application. Resource consumption is a run-time event, but the controls in place to prevent container runaway and consuming all CPU, memory, and IO resources occur in the environment prior to container launch.

The Operator needs to communicate to the Developer an application's current resource usage, security parameters, and necessary logs so the Developer can make informed tests and QA decisions to ensure that containerized applications under development can operate within the limits placed on the container given the resources required in production environment.

3.1.15 Container Management - Data Backups and Replication

In container management as it applies to data backups and replication, the following proposed mitigations for the challenges are recommended:

1. The Operator must ensure data backup systems can back up any new data within a container before it is destroyed.
For new containerized applications, no persistent data should be stored within the container. Realizing that some legacy applications may be running within application containers, the Operator must work with Developers to understand where data is saved and ensure that this data is backed up both in regular intervals and before container destruction. Some CMPs provide notification before container shutdown/destruction; if available, this feature should be leveraged to ensure no data is destroyed without backup.

Developer's viewpoint:

1. Properly document the requirements for an application backup and restore of an application. This documentation must include the state for which an application can be backed up and restored. For some applications, a simple snapshot of the storage medium may suffice. For other applications, application-specific tools will need to be run. If an application needs to be taken offline or drained of active users/session in any capacity, then it must be documented.
2. If an application needs to be taken offline for a backup and/or restore, documenting what services flow upstream and downstream from that application is critical. By documenting this requirement, an Operator can ensure that other containers are run temporarily using techniques such as a blue-green deployment to ensure that a cascading failure does not occur.
3. If the Developer has influence over the upstream or downstream dependencies of the application, using cloud-native best practices is critical. For instance, implementing the services to be designed for recovery by having service dependency retries and idempotency built into the application is important. These best practices will ensure that if the application is unavailable during a backup/restore that upstream and downstream dependencies will be able to recover and further avoid a cascading system failure.
4. Developers need to adhere to key microservice practices of single-service responsibility, interface separation, and avoidance of transitive service dependency. When dealing with immutable infrastructure, Developers need to apply gateway service or service helper patterns to offload backend operational processes from compute occurring inside the container. Developers should refrain from placing reliance on web application archives (WAR package files) and numerous JAR files placed in the file system to augment or shape the client application runtime environment.

Developers also need to build applications tolerant of dynamic deployment patterns that are prone to variable latency and not knowing where the workload executes.

5. Developers should deliver software capability that manages data and metadata independent of the file system within which the workload executes. The Developer has to treat the container execution as a "unit of processing" whereby the data consumed exists outside the container but is accessible via logical volume reference. The temptation to use the Developer's code repository as the restore point is a misplaced assumption. A well-constructed and executed backup API will not only be able to restore data and metadata; it should also be able to do so independent of the compute stack and be able to restore to a point in time.

Operator's viewpoint:

1. The Operator should implement how an application will be backed up and restored on a system using the criteria set forth by the Developer and organizational policies. These policies may include requirements for the number or frequency of the backups.
2. For simple use cases in which the storage snapshots will be sufficient, the Operator should schedule backups in accordance with the application and organizational requirements.
3. Some environments may include the ability to create volumes and dynamically map them to storage devices; this is often called dynamic provisioning. If an Operator is using a snapshot policy, making this work with dynamic provisioning may not be possible since many storage mediums will not allow setting snapshot schedules until after the volume is created. If this cannot be done properly, an Operator should disable dynamic provisioning.
4. For non-simple use cases, the Operator will be responsible for creating the safe-state for which the application can be backed up and for running the application-specific tools required to back up the application. This may include standing up temporary application instances to handle the workload for application dependencies when the application is taken offline. This may be non-trivial. As an example, an Operator may need to:
    a. Rollout a blue-green deployment to offload traffic while the application is taken offline.
    b. Create the safe-state for backup/restore as listed by the Developer and/or vendor of the application. This may include draining sessions or closing out all transactions. Implementing this in a container scenario could involve using a sidecar pattern (an agent running a co-located container next to a database) or using other cron tools.
    c. Running the application-specific tool is required to back up the application. For web servers this may include replication of all events to a secondary cluster. For databases this may include running specific tools. If the tool creates an archive of data, then the data needs to be backed up safely to a storage medium. This may require custom scripting to move the data archive to a safe location.

3.1.16 Container Management - Container Rehosting Between CMPs

Proposed mitigations for the challenges present in container management, as it applies container rehosting between CMPs, comprise the following recommendations:

1. For an Operator of legacy applications that use sensitive data, some work will need to be done to create and manage utilities that automate the exporting and importing of sensitive data related to a container that is being rehosted.

There is currently no universal standard method to export and import a platform or container's security configurations between CMPs. This activity could take place at the CMP level or in the cloud broker role. In the meantime, the Operator will need to manage utilities to provide the capability to rehost containers and their related security components.

The mitigation relies on the manual process of creating the empty container with security configuration at the target CSP, migrating the container itself, and then deleting the container and its security configuration at the source CSP. Doing this manually risks incurring mistakes, so having a collection of utilities that can automate this process will be worth the time invested.

3.1.17 Container Encryption

Proposed mitigations for challenges present in container encryption comprise the following recommendations:

1. Developers and Operators should agree to use standard, commonly available authentication systems.
By using common authentication methods, this ensures a predictable ability to authenticate and authorize the use of encryption keys across CMPs. When either data or applications are encrypted within a container, having a reliable method to authorize use of encryption keys is required. Alternatively, applications must be written to support multiple authentication methods, introducing unnecessary complexity.

2. Developers need to identify what data at rest (DAR) solution they will utilize, taking into account if the solution has been validated by an impartial third party and if the solution is viable in the intended operational environment.

The Developer should utilize a DAR solution, if at all possible, that has been FIPS 140-2 validated. The Developer should also analyze the application programming interface (API) and a compatible key management system (KMS). With a FIPS-validated solution and a compatible KMS, there is some third-party validation that the solution is secure and thus a higher level of assurance.

The Developer should utilize the latest advanced encryption standard (AES) with the highest possible key length, which will ensure data confidentiality.

3. For sensitive applications, Developers should encrypt the sensitive application, then create a container entrypoint application that decrypts and executes the main application.
By encrypting the sensitive application, the risk of unintended exposure of the sensitive application can be mitigated. This allows the container image (and sensitive application) to be stored, transmitted, or shared without unauthorized exposure. Encryption and decryption of the application should be treated as any other data encryption/decryption process.

Appendix A–Acronyms

Selected acronyms and abbreviations used in this paper are defined below.

ACM	Application Container and Microservices
CMP	Container Management Platform
CSP	Cloud Service Provider
HSM	Hardware Secure Module
IAM	Identity and Access Management
KMS	Key Management System
OS	Operating System
RBAC	Role-Based Access Control
SLA	Service-Level Agreement
SSL	Secure Sockets Layer
VM	Virtual Machine

Appendix B –Glossary	
Application container [NIST SP800-180]	An application container is a construct designed to package and run an application or its components running on a shared operating system. Application containers are isolated from other application containers and share the resources of the underlying operating system, allowing for efficient restart, scale-up, or scale-out of applications across clouds. Application containers typically contain microservices
Container management platform	A container management platform is an application designed to manage containers and their various operations, including but not limited to deployment, configuration, scheduling, and destruction.
Container lifecycle events	The main events in the life cycle of a container are create container, run container, pause container, unpause container, start container, stop container, restart container, kill container, destroy container.
Developer [NIST SP 800-64 Rev 2/ ISO/IEC 17789:2014]	The cloud service Developer is a sub-role of cloud service partner which is responsible for designing, developing, testing, and maintaining the implementation of a cloud service. This can involve composing the service implementation from existing service implementations. The cloud service Developer's cloud computing activities include: • design, create and maintain service components (clause 8.4.2.1); • compose services (clause 8.4.2.2); • test services (clause 8.4.2.3). NOTE 1 – Cloud service integrator and cloud service component Developer describe sub-roles of cloud service Developer, where the cloud service integrator deals with the composition of a service from other services and where cloud service component Developer deals with the design, creation, testing, and maintenance of individual service components. NOTE 2 – This includes service implementations and service components that involve interactions with peer cloud service providers.
Host	OS supporting the container environment
Lateral movement [LIGHTCYBER 2016]	Lateral action from a compromised internal host to strengthen the attacker foothold inside the organizational network, to control additional machines, and to eventually control strategic assets.

Microservices [NIST SP800-180]	A microservice is a basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of well-defined APIs, independent of any vendor, product, or technology. Microservices are built around capabilities as opposed to services, build on SOA, and are implemented using Agile techniques. Microservices are typically deployed inside application containers.
Enterprise Operator	The individual or organization responsible for the set of processes to deploy and manage IT services. They ensure the smooth functioning of the infrastructure and operational environments that support application deployment to internal and external customers, including the network infrastructure, server and device management, computer operations, IT infrastructure library (ITIL) management, and help desk services. [1]
Enterprise Architect	The individual or organization responsible for strategic design recommendations. They determine, by applying their knowledge of cloud, container and microservices components to the problems of the business; the best architecture to meet the strategic needs of the business. Additionally, they develop and maintain solution roadmaps and oversee their adoption working with Developers and Operators to ensure an efficient and effective solution implementation.
Container resources	Four resources required for containers to operate are CPU, memory (+swap), disk (space + speed), and Network.
Container resource requests	The amount of CPU, memory (+swap), and disk (space + speed) that the system will allocate to the container considering the resource limit.
Container resource limit	The maximum amount of resources (CPU, memory (+swap), and disk (space + speed)) that the system will allow a container to use.

1 Wikipedia. Information technology operations. Retrieved from https://en.wikipedia.org/wiki/Information_technology_operations.

Appendix C–References	
[SP 800-180]	NIST Special Publication (SP) 800-180 (Draft), *NIST Definition of Microservices, Application Containers and System Virtual Machines*, National Institute of Standards and Technology, Gaithersburg, Maryland, February 2016, 12pp. http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf
[SP 800-160]	NIST Special Publication (SP) 800-160, *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems,* National Institute of Standards and Technology, Gaithersburg, Maryland, November 2016, 257pp. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf
[SP 800-123]	NIST Special Publication (SP) 800-123, *Guide to General Server Security*, National Institute of Standards and Technology, Gaithersburg, Maryland, July 2008, 53pp. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf
[NIST SP 800-64 Rev 2]	NIST Special Publication (SP) 800-64 Rev 2, *Security Considerations in the System Development Life Cycle*, National Institute of Standards and Technology, Gaithersburg, Maryland, October 2008, 67pp. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf
[ISO/IEC 17789:2014]	International Organization for Standardization/International Electrotechnical Commission, *Information Technology – Cloud Computing – Reference Architecture*, ISO/IEC 17789:2014, 2014. https://www.iso.org/standard/60545.html [accessed 5/11/17].
[LIGHTCYBER 2016]	*Cyber Weapons Report 2016,* LightCyber, Ramat Gan, Israel, 2016, 14pp. http://lightcyber.com/cyber-weapons-report-network-traffic-analytics-reveals-attacker-tools/ [accessed 5/11/17].

