Container Security Pattern

Published Version

The content of this document has been reviewed as follows:

Version	Valid from	Reviewed by
1.0	25.06.2021	Integration Architecture Council, Information Security Officers (IS Core Group), Security Architects

Table of Contents

Table of Figures

1 Introduction

1.1 Purpose

The Container Security Pattern guides the implementation for secure container-based applications and architecture. The document aims to:

- Explain security concerns associated with container technologies
- Provide recommendations and best practices for addressing concerns
- Strengthen Allianz' security posture
- Simplify and standardize the approach
- Align container security with Allianz organizational context and goals

This document doesn't offer step-by-step guidance on configurations, but rather describes a general approach, explaining what needs to be done, and in some cases providing examples on which tools or methods to use.

Despite the use of the verbs "must" and "should" in this document, the measures defined herein are not currently mandatory and represent the future target architecture within Allianz. This document is expected to become mandatory in the future. The measures defined in this document shall be used to guide the development of your own roadmap to reach the target state.

1.2 Scope

Security can only be effectively applied once a set of repeatable and appropriate measures have been put in place. The document covers the various stages of a container lifecycle integrated into the CI/CD pipeline (see Figure 1). All other forms of virtualizations, other than containers are out of scope. The image building process, concerning the configurations and components is partially covered, to the extent needed for container security.

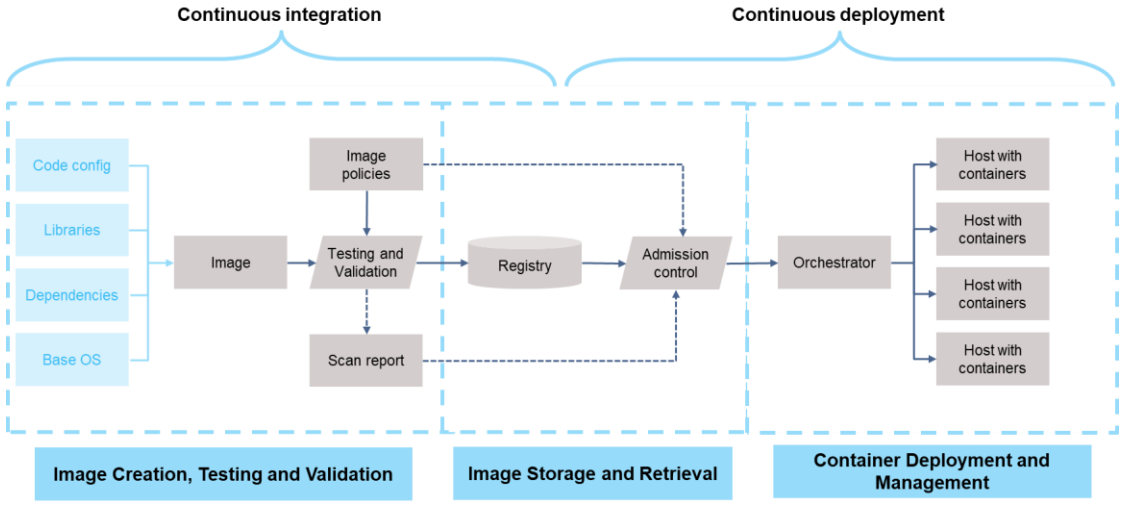

Figure 1: Container technology architecture tiers and components

The document assumes that the reader carries some existing knowledge of operating system, networking and security, and is familiar with the concepts of virtual machines (VMs) and hypervisors. Therefore the reader, as needed, may choose to refer to the following documents:

- Allianz Functional Rules for Information Security (AFRIS)

- Allianz Information Security Practices (esp. Allianz Information Security Practice #06 Secure Software Development)
- Service Security Specification CRP 2.0

This document focuses on the security aspects of containers, leaving out the topics of infrastructure and secure coding (see Figure 2). The applications within containers and some parts of an underlying host operating system will be covered by other patterns. The requirements are generalized to the extent that makes them valid for different container orchestration platforms (e.g., AKS, AWS EKS, GKE) and container runtimes (e.g., Docker, RKT, RUNC, containerd, cri-o). It is assumed that the reader works with the cloud-managed Kubernetes and is not setting up a self-managed cluster. Therefore, the primary focus of this document is to cover the security-related aspects of Allianz PaaS deployment. Since this type of deployment limits possible container runtime configurations, this topic is only partially covered.

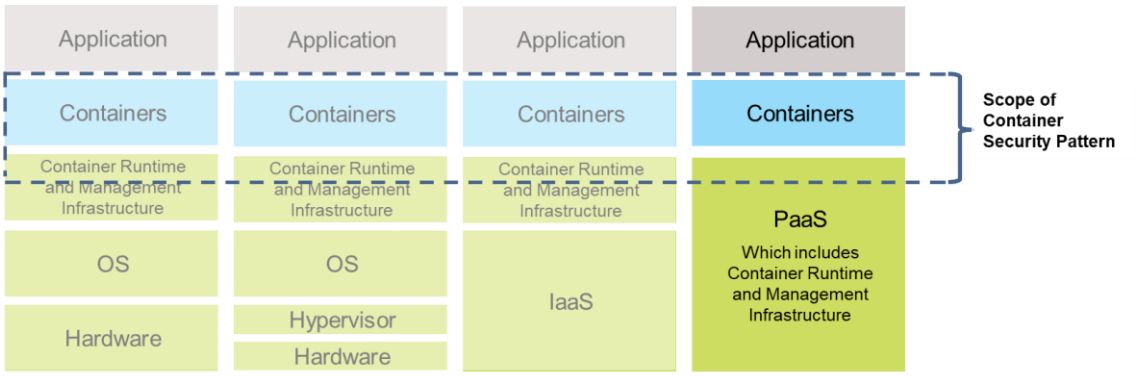

Figure 2: Scope of Container Security Pattern

The document does not cover the cases where an orchestration platform has to be set up locally from scratch. In those cases, additionally to the scope outlined within this document, the security of the underlying OS and overarching topology has to be considered.

1.3 Methodological approach

This document was established by the security architecture team at Allianz group together with technical representatives from Allianz technology and OEs, working specifically with the containerized environment.

The solution of the Container Security Pattern contains best practices, considered and incorporated from the following standards, benchmarks, and literature:

- NIST SP 800-190 Application Container Security Guide
- OWASP Application Security Verification Standard v1.0
- CIS Docker Benchmark v1.2.0
- CIS Kubernetes Benchmark v1.6.0
- CIS Amazon Elastic Kubernetes Service (EKS) Benchmark v1.0.1
- Azure AKS Security Concepts
- O'Reilly Container Security, 2020
- O'Reilly Kubernetes Security, 2018

Whilst the listed measures are mainly technical in nature, they strive to establish a solid baseline for all container projects that are implemented in Allianz, independently of their criticality level.

1.4 Exceptions

In case a mandatory requirement cannot be fulfilled, the application must undergo a risk acceptance process. This assessment is initiated and managed with the Product Owner.

An exception process must only be possible if mitigation or resolving measures do not exist or do not lower the risk level. The exception process has to involve the risk management process and include an approval of the Information Security Officer. For more details on risk acceptance process, refer to the Allianz Functional Rules for Information Risk Management (AFIRM).

1.5 Link to ESA Framework

As part of the Enterprise Security Architecture Framework (see Figure 3), Security Patterns provide guidance on the secure implementation or usage of technology at Allianz. These patterns define an important subset of security-related capabilities and should be considered critical measures to protect information assets.

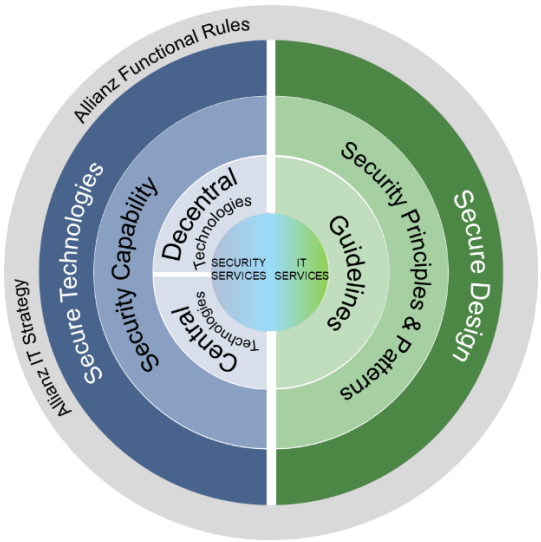

Figure 3: Link to ESA Framework

1.6 When and how to use this document

The document is intended to be used as a reference point for recommendations to be applied to any or all container technologies. The reader of this document shouldn't assume by implementing the measures mentioned later in this document that the container environment will be fully secured. This document should be used in conjunction with Allianz security directives, standards, and guidelines that cover different related aspects of container security. Where applicable, the related Allianz and industry resources are referenced.

2 Container security problem and context

2.1 Problem definition

Containers are appealing to companies for many reasons; amongst which, portability and scalability. However, when security becomes an afterthought, rolling out an application puts the business at high risk. Risk can be grouped into four main fields:

- **Exposures from misconfigurations.** Due to both the flexibility and complexity of modern cloud runtimes, configuration management poses a difficult challenge for security practitioners. Inadequate configuration management is a common security lapse in containerized environment; in turn, this exposes the business to some of the worst threats.
- **Runtime threats.** The runtime phase is critical for container security, which if neglected will expose the application to external threats.
- **Failed compliance audit.** Neglecting to implement security controls in the early stages of app development lifecycle, introduces non-compliance and unacceptable risk exposure.
- **Failed vulnerability management.** There are a lot of common threats that can be attributed to known vulnerabilities. These threats include privilege escalation, crypto mining or similar malware installations, and host access.

Managing these risks robustly will ensure that Allianz operates as responsibly as it should, and it can avoid or minimize disruption and damage to business. The pattern tries to answer the questions of how to address these risks by securely building containerized environment.

2.2 Context

The Container Security Pattern covers containerization within the context of a PaaS environment as created, configured, and managed by Allianz. In this setup, security and compliance are considered shared responsibilities. The cloud provider is responsible for the security "of" the cloud, while Allianz has to ensure the security of the containerized environment "in" the cloud. Allianz is responsible for securing its applications, data, and user access. As to the defined scope of this document, it means that Allianz is responsible for managing:

- IAM users/ roles
- Identity/ resource-based policies
- Node and pod security
- Runtime security
- Network security
- Secrets management
- Configurations as to the best practices.

The other topics are in charge of the cloud provider.

2.3 Stakeholders

This document is intended to be used by Allianz individuals involved into container-based development such as:

- Group and OE Security Teams
- IT Architects
- Development Teams
- Container Operation Teams

2.4 Roles description

This document is intended for the following user groups.

Architects	Architects are responsible for the creation of the overall design, the infrastructure and the definition of the underlying concepts of a containerized application. They are expected to take the requirements of this standard into account.
Developers and Testers	Developers and Testers implement the application design defined by architects and are therefore responsible for the initial setup and maintenance of applications and the creation of images and containers. Also, they are in charge of functional/non-functional tests to ensure everything works as expected. Developers are also responsible for the continuous operation of the images/containers.
Information Security Officer	Information Security Officer (ISO) is responsible for maintaining the security policies and standards. The ISO supports/ ensures the implementation of the requirements.
Workload Operator	To ensure the segregation of duties, the same person is not allowed to perform a deployment into a production environment. In the same way, the automated processes are not allowed to perform a deployment automatically without approval. Therefore, workload operator is a role, in which a developer is responsible to perform the actions, which ensure the Segregation of Duty (SOD) principles. A Workload Operator ensures that running applications undergo the required monitoring checks and applies the needed security measures.
Platform Operator	A Platform Operator is responsible for orchestration environment and default policies. The Platform Operator ensures there are required solutions in place that can be used by workload operators to secure the applications running in the production environment.
Automation Engineer	An Automation Engineer is responsible for building the pipeline in accordance to the defined security and technology requirements.
Product Owner	A Product Owner is responsible for conducting risk acceptance for applications that do not comply with outlined security requirements.

2.5 Issues pertaining to container lifecycle

In order to understand the related security concerns, the different components and steps within the lifecycle of a cloud service are described. The lifecycle as depicted in Figure 1 is used from left to right.

Images

An image should only include the executables and libraries required by the app itself; all other OS functionality is provided by the OS kernel within the underlying host OS. Image creation typically uses build management and automation tools to enable and assist with a continuous integration process.

Image creation is followed by testing and validation. Testing can include static and dynamic analysis, peer review, and vulnerability scanning. Depending on the environment that is being used, vulnerability scanning might happen automatically. For example, in Azure Container Registry the vulnerability scanning can be enabled, which will check images before pushing them into registries.

Validation is intended to verify the contents of images, sign images and send images to the registry. The consistency of building, testing and validating the same artifacts for an app is one of the key operational and security benefits of using containers.

Image Registries

Image Registries store ready to run images. They are the resulting artifacts of the container build process. Images are tagged and cataloged for identification and version control to assist in discovery and re-use. Registries can have triggers built into them that enable newly created images to be pushed to registries post the testing phase. For example, if working on Azure cloud, Azure Defender for container registries can scan any image that has been pulled within the last 30 days. For each environment this can be further automated to deploy new images, thus enabling faster turnaround time for projects.

Container

An instance of an image is called a container. When the image is started, it turns into a running container of this image. Containers increase the resilience of applications due to the immutable nature of container images. Containers are a good fit for micro-service architectures and thus help to implement them. For cloud-native applications, developers can leverage microservices architecture to enable fast iteration of the application and self-contained deployment.

Container technologies emphasize and enable reuse, such that container images created for an original purpose can be easily shared and reused to build new standalone or integrated solutions.

Container Orchestration

The primary job of an orchestrator is to schedule and run containers based on declared desired state. Strict access control to container orchestration platforms such as Kubernetes for the purpose of preventing risks from over-privileged accounts, attacks over the network, and unwanted lateral movement need to be addressed using whitelisting techniques. There is a requirement to secure communication between pods on an orchestration cluster shared by multiple applications. Without Service Mesh or in application implemented encryption, all traffic is unencrypted.

2.6 Container characteristics and associated concerns

Application containers and microservice architectures are used extensively to design, develop and deploy applications; typically following agile methodologies and DevOps. The security of application components needs to be considered throughout the Software Development Life Cycle. There are multiple attack vectors that can be used to compromise a containerized environment (see Figure 4); with the attacks associated with different stage of the container's lifecycle. The entry points can be vulnerable application code, badly configured container images, insecure image build processes, a badly configured registry and much more.

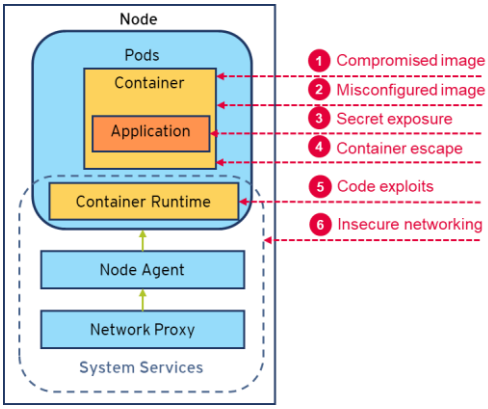

Figure 4: Container attack vectors

Compromised image (1): If container image is built with vulnerabilities in it, the weaknesses can later be used to attack the running container. When this happens, this can drive to a container getting more privileges on the host than it needs.

Misconfigured image (2): There are many opportunities to introduce weaknesses when configuring how a container image will be built. These can later be exploited to attack the running container. An example of such misconfigurations can be letting a container run as the root user.

Secret exposure (3): Application code often needs credentials to do its job. There is a couple of possibilities how secrets can be stored. If they are stored within an application, a hacker can steal them.

Container escape (4): Container escape vulnerabilities can lead to getting access to the host environment. To minimize the risk, isolation mechanisms must be considered.

Code exploits (5): The code and third-party dependencies that an application relies on, can include vulnerabilities. The best way to avoid running containers with known vulnerabilities is to scan images.

Insecure networking (6): In every distributed system different components need to communicate with each other. When communication is unencrypted, the data in transit can get stolen.

More attack vectors, including the relevant ones for the Kubernetes orchestrator, can be found in MITRE ATT&CK Framework for Kubernetes and Container Runtime Security. These have been taken into considerations when creating protection measures.

3 Container security solution

3.1 Solution introduction

The solution covers security considerations for the complete container lifecycle focusing primarily on the following aspects at each stage:

A. Image Security
B. Registry Security
C. Container and Inter-Container Security

The measures that are important but not specific to any concrete stage have been outlined as general requirements.

Each measure expresses a single responsible and is described by:

- A unique measure ID
- Measure name
- Description
- Responsible

3.2 General security requirements

The following defines best practice, which must be followed.

ID#	Measure	Description	Responsible
gR-01	Best coding practices	Refer to the best coding practices created by Allianz. Check for the technical best practices accessible in the following git repository: Patterns for container images and deployment definitions.	Developers and Testers
gR-02	Least privilege	A least privilege access model must be used, in which users are only granted the ability to perform the specific actions on the specific hosts, namespaces, containers, and images that their job roles require. For example – test team members must have no access to containers in production. Cluster admin roles must be avoided. Permission to create pods must be restricted. Administrative accounts must be reviewed on a regular basis. Multifactor authentication should be used to manage access in such cases.	Platform Operator
gR-03	Encryption enablement	Encryption for both data at rest and in transit must be enabled.	Platform Operator
gR-04	Encryption usage	Encryption for both data at rest and in transit must be enabled.	Workload Operator
gR-05	Security scanning	All critical components must undergo regular automated security scans.	Platform Operator
gR-06	Centralized security logging system	A centralized system component logging capability must be in place to enable platform debugging, troubleshooting, forensics and analytics.	Platform Operator

gR-07	Centralized workload logging system	A centralized workload logging capability must be in place to enable application debugging, troubleshooting, forensics and analytics	Platform Operator

3.3 Image security

Container images make application deployment easy and convenient. But alongside that ease and convenience, it brings security considerations that must be addressed. Since an image is at the heart of containerized environments, image security must be a priority when using different in-house or third-party registries. As such, the following must be addressed:

- Where does the image come from?
- Can the image publisher be trusted?
- Is there an objective cryptographic proof that the author is who he claims to be?
- Has the image reached its end of life, or is it patched and updated regularly?
- Which security policies are being used?
- Has the image been tampered with after it was pulled?

The image integrity must be ensured on every development stage, since there are various potential weak points, which can be exploited. To find out more about the pipeline security, refer to the "Allianz Information Security Practice – Secure Software Development" document.

A series of image security requirements must be set, and controlled. If image security does not reach the required level, further image usage must be prevented (see Figure 5).

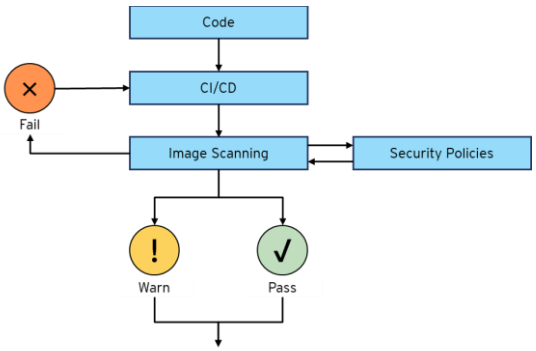

Figure 5: Image security check

Further measures to reach the needed security level of image are listed in the following chapters.

3.3.1 Image configurations

The first step involved in ensuring functioning containers, is the creation of images. As an image is the base of a potentially enormous number of containers, the security of each image is essential to ensuring safe operation of an environment.

ID#	Measure	Description	Responsible
iC-01	Trusted images	Only images from trusted sources must be used. Though official repositories such as DockerHub exist, they do not guarantee that all their images are secure. Developers should be connecting and downloading images from Allianz private registries.	Developers and Testers

iC-02	Verified software packages	When building a container image, only verified software packages must be installed. Pulling packages should be done through the tooling, which ensures secure and private provisioning. Images should only have necessary packages installed. This can be reached by using multi-stage builds as a way of eliminating unnecessary contents (layers) in the final image. All necessary packages must be included in the container image, and never installed at the runtime. When working with Dockerfiles, check for the technical best practices accessible in the following git repository: Patterns for container images and deployment definitions.	Developers and Testers
iC-03	Copying files from local storage	When building a container, the files that have to be included must be copied from the local storage, and not from external sources through URL. Arbitrary URLs specified to add files can result in MITM attacks or sources of malicious data. For example, when using Docker, this can be ensured by using the COPY directive instead of ADD. The COPY directive copies files from the local host machine to the container file system. When working with Dockerfiles, check for the technical best practices accessible in the following git repository: Patterns for container images and deployment definitions.	Developers and Testers
iC-04	Signature enforcement	Signed image enforcement should be enabled. Signed image enforcement can prohibit images that are unsigned, have malformed signatures, and/or compromised signatures from being deployed. As an example, this can be enforced by enabling Docker Content Trust. Azure Container Registry also implements Docker's Content Trust model. Images should be cryptographically signed as part of the build pipeline. Docker trust's built-in generator can be used to generate delegation key pair. Certificate should be backed up, and credentials must be stored securely, using a password manager.	Developers and Testers
iC-05	Tagging	Image tags must be used consistently. Stable tags must be used to maintain base images for container builds. Unique tags must be used for deployments, especially in an environment that could scale on multiple nodes. Labels like "latest", which are not exactly specified, should not be used.	Developers and Testers

3.3.2 Vulnerability and patch management

ID#	Measure	Description	Responsible
iV-01	Security scanning	All images must undergo regular automated security scans. An image scanner must be used to identify vulnerabilities, malware, and policy violations within an image and track components and versions included in each one.	Platform Operator

iV-02	Layers visibility	Developers must embed vulnerability management process and tools (e.g. GitHub Advanced security, Checkmarx) within the entire lifecycle of images, excluding deployment stage when the image becomes a container. Visibility into vulnerabilities across all layers of the image must be ensured, including the base layer, application framework and custom software.	Developers and Testers
iV-03	Quality gates definition	Quality gates must be designed for each stage of the build and deployment process to ensure compliance with vulnerability and configuration policies. A threshold for images that include vulnerabilities with Common Vulnerability Scoring System (CVSS) ratings must be defined. ISOs need to be consulted concerning the defined thresholds and analysis results.	Architects
iV-04	Quality gates implementation	Quality gates must be integrated at each stage of the build and deployment process to ensure compliance with vulnerability and configuration policies. A rule in the build process must be configured to prevent the proliferation of images that include vulnerabilities with Common Vulnerability Scoring System (CVSS) ratings above a selected threshold.	Automation Engineer
iV-05	Threat remediation	New applications must not start with old vulnerable images. Upon threat identification, the existing deployments running on the vulnerable image must not be shut down until a remediation solution is found. The approach taken will depend on the criticality of the impacted application. Therefore, the IS Practice Vulnerability Management must be consulted for more details. In exceptional cases, the application must undergo a risk acceptance process, which is managed by the Product Owner.	Production Operator
iV-06	Vulnerability Management Process	Identified security risks and vulnerabilities must be timely eliminated and centrally managed according to a predefined Vulnerability Management Process. The patching in a container deployment must be implemented systematically resulting in the build of a new container image. Following which, effected container must be redeployed based on the new image.	Developers and Testers

3.3.3 Access management

ID#	Measure	Description	Responsible
iA-01	Root account disablement	The root account must be disabled. Unless a container image specifies a non-root user or a non-default user when the container runs, by default the container will run as root. A dedicated user and group should be created, with minimal permissions to run the application. The same user should be used to run this process. Vendor-provided container images that require root privileges must run in dedicated container runtimes (e.g. dedicated Kubernetes cluster). They must not be collocated with other workloads.	Developers and Testers

iA-02	Secrets management	Secrets must not be stored in images. Secrets must be acquired at runtime and only be kept in the main memory for the shortest timeframe possible. Orchestrators provide storage of secrets and 'just in time' injection to containers. For example, orchestrators can be used to securely provision the database connection string into a web application container. Containers should also be integrated with password management tools that typically provide APIs to retrieve passwords.	Developers and Testers
iA-03	Passwords encryption	Passwords (including keys) must always be encrypted at rest and in transit.	Developers and Testers

3.4 Registry security

The repository in which container images are stored is called a registry. Container registry management needs to be in place to control how containers are pushed and pulled across registries in different staging environments i.e. development through to production.

Public registry contains open-source images as pre-packaged container images which makes it easy for developers to pull them. As a consequence it is more likely to have security issues when developers pull images from the public registry without sufficient prudence.

Private registry is controlled to share its custom base images within the organization, keeping a consistent, private and centralized image repository. It can be better managed by an organization because the private registry lets an organization implement image signing and access control for governance.

Most of the public registries will provide capabilities for scanning and vulnerability detection. For private registries, security tools and controls should be enabled to ensure robust and secure image management.

Allianz uses Allianz Technology integrated registry, which is a private registry.

Image registries are an important part of the container security. The images they contain must be scanned for security vulnerabilities, and depending on the results either deployed or prevented from deployment (see Figure 6)

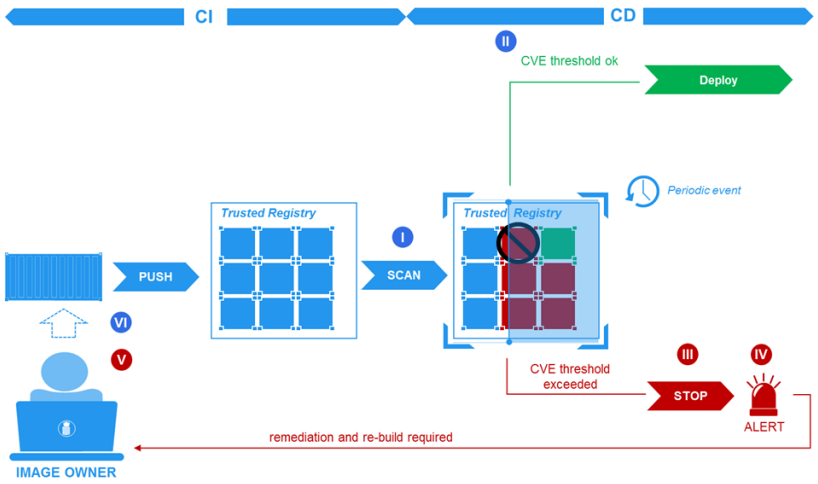

Figure 6: Container image registry scanning

Access to image registries must be limited and monitored. Measures on how to secure image registries can be found in subsequent chapters.

3.4.1 Images storage

ID#	Measure	Description	Responsible
rI-01	Set of registries	A set of private registries of Allianz must be maintained, and only these registries should be allowed to store images. This can be achieved by creating a global registry with cross-region replication and geo-based routing. The registries must be access-protected so that only authorized users are able to upload and modify images.	Platform Operator
rI-02	Adding images to registry	New images must not be published or made available within a registry without being scanned. The scanning must confirm that the image has no indicators of being compromised, has no severe vulnerabilities and malware infected content. ISOs need to be consulted concerning the defined thresholds and analysis results.	Automation Engineer
rI-03	Encryption at rest	Images must be encrypted when stored in the registry. Some registries (e.g. Azure Container Registry) enable it automatically, while others (e.g. Amazon ECR) need to be configured. In the latter case, a key management service must be used to create, manage, and control keys used to encrypt and decrypt data.	Platform Operator
rI-04	Version control	All images must be managed using version control, and assigned with a unique identifier and timestamp. This enables housekeeping and archiving process. Time triggers and labels can be associated with images.	Developers and Testers
rI-05	Scanning	There must be a process to conduct continuous scanning of all images held within a registry to verify if newly published vulnerabilities have the potential to impact stored images. This can be achieved either by a platform-native image scanning (e.g. for Amazon the ECR image scanning can be used), or by a third-party provider.	Platform Operator
rI-06	Image pull frequency reporting	Image pull frequency must be reported. This will ensure the capability to identify older versions of images that are no longer used and can be deleted. Adopting a policy of regularly updating and refreshing older images makes it easier to eliminate vulnerabilities that may have gone undetected or introduced by continued use of legacy versions of software.	Platform Operator

3.4.2 Registry connection

ID#	Measure	Description	Responsible
rC-01	Encryption in transit	Development tools, orchestrators and container runtimes must only connect to registries over encrypted channels. All	Platform Operator

		data pushed to and pulled from a registry must occur between trusted endpoints and encrypted in transit over SSL/TLS connections.	
rC-02	Transaction logging	All changing actions done to the image registries must be logged to a central logging system, which must be different than the registry systems. An audit trail must be centrally monitored. When done so, risks with specific times, dates and locations can be quantified.	Platform Operator

3.4.3 Access management

ID#	Measure	Description	Responsible
rA-01	Authentication	All access to registries that contain proprietary or sensitive images must require authentication (e.g. through usage of tokens, or user certificates), that is in line with the corporate standards. Any write access to a registry must require strong authentication to ensure that only images from trusted entities can be added to it. Allianz standard authentication solutions must be used.	Platform Operator
rA-02	Access at repository level	Access must be granular and specified at the repository level, allowing users to access specific repositories only rather than any/ all. All read/ write actions on sensitive images within a registry must be logged and audited.	Platform Operator
rA-03	Role-based access control	Access to images must be based on user roles and must be audited and implemented in accordance with the Allianz policies which define Joiners, Movers, Leavers Process (JML Process). Role-based access control must be used to enforce user permissions related to specific activities, locations and departments. RBAC defines who can publish updated images, and who can consume them. Restricting user access to image registry by limiting interaction with build images to only authorized personnel and service accounts, prevents malicious users from replacing built images with vulnerable containers.	Platform Operator

3.5 Container runtime and inter-container security

Adequate protection of containers deployed in production is an Enterprise challenge. This requires real-time monitoring of containers, putting the right policy and audit in place to discover malicious attacks or zero day vulnerabilities promptly so that actions can be taken to contain the damage.

There are many security implications associated with the ways that containers are started. Some runtime parameters can be supplied that bring about security consequences that could compromise the host and the containers running on it. It is therefore very important to verify the way in which containers are started, and which parameters are associated with them.

Container security can be seen from three main perspectives (see Figure 7):

- Inter-container protection
- Protecting the host from containers

- Protecting containers from application(s) in it

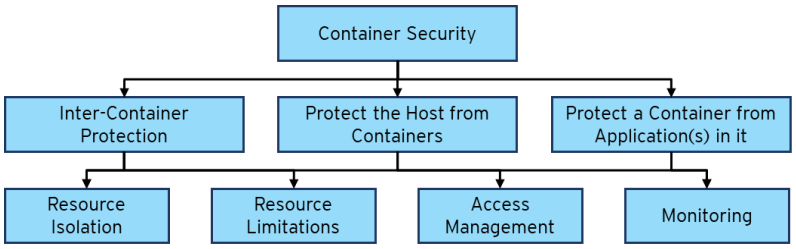

Figure 7: Container security perspectives

The following tables define important measures that must be applied in order to protect the container and inter-container environment.

3.5.1 High-level configurations

ID#	Measure	Description	Responsible
cH-01	Separate environments	Separate environments must be instituted for production, and non-production need; each with controls to provide role-based access control for container deployment and management activities. All container deployment must be associated with individual user identities, and logged to provide clear audit trails. Environment separation can be achieved by setting up a separate cluster. Usage of separate clusters provides a higher security level and better overview of different environments and is therefore a preferred option of choice.	Workload Operator
cH-02	Privileged containers prohibition	Privileged workload containers are prohibited. Docker and other container runtimes provide a possibility to specify a --privileged option when running container. This flag must not be used. If there are reasons to run containers with privileged rights, specifying individual capabilities instead must be considered. All containers must be prohibited from acquiring privileges in addition to those initially assigned, e.g. via SUID or SGID bits.	Developers and Testers
cH-03	Open ports limitation	It must be ensured that only required ports are open. At the same time, container ports should not be mapped to the privileged host ports when starting a container. Privileged ports are the ones with numbers below 1024. See exemplary Pod Descriptions in the following git repository: Pod definition standards	Developers and Testers
cH-04	Resource requests and quotas	Application and container resource quotas must be defined in order to ensure that containers do not consume more resources than reasonable and thereby impact the availability of other systems. Starving other processes might happen inadvertently by running application, or as a result of a resource exhaustion attack.	Workload Operator

cH-05	Encryption at rest	Data stored in file systems must be encrypted at rest.	Platform Operator
cH-06	Certificates rotation	Client and server certificates must be rotated in regular intervals. For example, this can be achieved by enabling kubelet client and server certificates rotation.	Platform Operator
cH-07	Distinguish environment	A proper distinguish mechanisms between the stages must be used. Proper environment labeling should allow for the grouping of resources. The labels should at least contain, but not be limited to Dev, Test, or Prod wording. For example, • "environment": "Dev" • "environment": "Test" • "environment": "Prod"	Workload Operator
cH-08	Pod policy	An OPA-based admission controller must be used. The Pod Security Standards (PSS) must be followed to address security-related best practices for Pods. The OPA-based admission controller defines a set of conditions that a pod resource must comply with in order to be accepted into the system. In addition, the OPA controller defines appropriate defaults. The controller allows an administrator to control aspects such as running privileged containers, using host namespaces, using volume types, using host file systems, etc.	Platform Operator
cH-09	Cleanup	Clean up unused containers as frequently as possible to reduce the attack surface.	Workload Operator
cH-10	Documentation	The following information must be clearly and concisely documented for reference (this includes, but is not limited to):	
		• Infrastructure	Platform Operator
		• Overall architecture and design	Architects
		• Components and networking inside and outside of the cluster	Platform Operator

3.5.2 Low-level configurations

ID#	Measure	Description	Responsible
cL-01	Network namespace	Containers (except the sidecar containers and platform components) must not share network namespace. Network namespaces (or netns) are a Linux networking primitive to isolate network devices. These are configured to impede lateral movement in case of container compromise.	Platform Operator

		Kubernetes automatically ensures network namespace separation.	
cL-02	User namespace	The host's user namespaces must not be shared. User namespaces ensure that a root process inside the container will be mapped to a non-root process outside the container.	Platform Operator
cL-03	File system namespace	Host system OS directories must not be mounted to container volumes. Each container must have its own file system namespaces such as /etc, /var, /dev and must be separated from the host OS. However, in the case when local disk PVs are used, this measure can be omitted to allow this use case for fast ephemeral storage.	Platform Operator
cL-04	Root filesystem	Containers must run with their root filesystems in a read-only mode which will prevent write activities to specifically defined directories.	Workload Operator

3.5.3 Monitoring

ID#	Measure	Description	Responsible
cM-01	Compliance	Compliance of the runtime environment must be aligned with container runtime configuration standards, and automatically assessed and continuously ensured. A tool to scan and assess configuration settings across the environment against the organization's compliance must be used. For example, Palo Alto Prisma scans cluster and pods against CIS, NIST as well as individual rules.	Platform Operator
cM-02	Technical logging	Technical logging configuration must be enabled. The logging must take place for all critical components.	Platform Operator
cM-03	Container activity monitoring	Security-related container activity must be properly monitored to identify security incidents, and to react to them in in a proper time. Therefore it is recommended to integrate the container runtime environment to Allianz standard security information and event management platform (SIEM). Automated alerts based on created thresholds must be used. Alerts must be evaluated to define risk level.	Workload Operator
cM-04	Anomalies detection	Profiling containerized apps can be automated to detect anomalies at runtime such as invalid process execution or malware execution. For example, this can be achieved by using a Container Models feature of Prisma Cloud. Deviations from normal behavior must be documented and reported.	Platform Operator

cM-05	Vulnerabilities monitoring	Container runtimes must be monitored for vulnerabilities. Vulnerabilities must be remediated according to the Vulnerability Management Guideline. Container runtimes must be configured to forward security and system event data to centralized logging systems.	Platform Operator
cM-06	Vulnerability Management Process	Identified security risks and vulnerabilities must be timely eliminated and centrally managed according to a predefined Vulnerability Management Process. The patching in a container deployment must be done through rebuilding a new container image, and redeploying containers based on the new image.	Developers and Testers

3.5.4 Network security

ID#	Measure	Description	Responsible
cN-01	Management plane	Management plane traffic must be separated from data plane traffic. This can be reached by logical or physical separation from the data that is provided to the end-user. Network policies must be enabled (like isolating pods/ firewall rules/pod-to-pod communication) with encryption in place.	Platform Operator
cN-02	API traffic	All API traffic must be encrypted via secure communication protocols. By default, this is done by a non-toxic TLS version.	Platform Operator
cN-03	Exposure level separation	Only containers with the same level of exposure (e.g. interaction layer and enterprise service layer containers, critical internal environment should be isolated) should run on the same node. The possible compromise of a single container can lead to the compromise of others. The risk-based approach must be used to define the level of exposure.	Workload Operator
cN-04	Encryption in transit enablement	To ensure a secure communication between pods, the encryption capability of the overlay network must be enabled. The platform operator must ensure there is a solution in place that enables encryption between pods.	Platform Operator
cN-05	Encryption in transit usage	To ensure a secure communication between pods, the encryption capability of the overlay network must be used. For example, this can be achieved by using Istio.	Workload Operator
cN-06	Egress traffic solution provisioning	Egress network traffic sent by pods must be controlled by creating network policies. These controls must take place at least at trust boundaries which exist between the virtual networks. Any traffic that is not explicitly allowed must be blocked by default.	Platform Operator
cN-07	Egress traffic solution usage	The egress traffic solution must be used. The workload operator must manage the allow list.	Workload Operator

3.5.5 Central management

ID#	Measure	Description	Responsible
cA-01	Restricted access	Access to the container runtime daemon/ APIs must be restricted. Unsuccessful access attempts must be logged.	Platform Operator
cA-02	Approved registries	All container images to be deployed to the cluster must be hosted within an approved container image registry. In order to ensure only those are pulled, this can be achieved through the usage of allowlist. Allowlisting only approved container registries reduces the risk of having malicious containers to be deployed into the cluster.	Platform Operator
cA-03	Root account prohibition	An Open Policy Agent must be configured in a way that prohibits containers from running with root privileges which enables access to the underlying platform.	Platform Operator
cA-04	Remote management avoidance	Remote management of container in the production environment must be prohibited. As an exception, in case of incident management, it must only be done through e.g. privileged access management.	Platform Operator
cA-05	Default service account disablement	Default service accounts must be disabled. Where access to the orchestrator's API from a pod is required, a specific service account must be created for that pod and rights must be granted to that service account.	Platform Operator
cA-06	Read-only access to registry	Depending on cluster type, registry access is constrained:	
		• The cluster service account must have the read-only access to registries, since it only requires a pull access to containers to deploy. This follows the least privilege principles by preventing credentials from being abused beyond the required role.	Platform Operator
		• In the case of a shared cluster, every team must set up their own service account to pull from registries, following the least privilege principle.	Workload Operator

4 Glossary

Name	Description
AKS	Stands for "Azure Kubernetes Service" and is an open-source fully managed container orchestration service that is available on the Microsoft Azure public cloud.
API	Stands for "Application Programming Interface" and allows applications to talk to each other by acting as a software intermediary.
Artifactory	A Repository Manager that functions as a single access point organizing all of the binary resources including proprietary libraries, remote artifacts, and other third-party resources.
CI/CD	CI and CD stand for "Continuous Integration" and "Continuous Delivery"/ "Continuous Deployment". It enables development teams to deliver code changes more frequently and reliably.
CIS	Stands for "Center for Internet Security (CIS)". Its mission is to "identify, develop, validate, promote, and sustain best practice solutions for cyber defense and build and lead communities to enable an environment of trust in cyberspace".
CLI	Stands for "Command-line interface" and is a command-line program that accepts text input to execute operating system functions.
Cluster	A set of nodes that run containerized applications.
Container	A container is a runtime instance of an image.
containerd	An Open Container Initiative (OCI) compliant core container runtime designed to be embedded into larger systems.
cri-o	An Open Container Initiative-based implementation of Kubernetes Container Runtime Interface.
CVE	Stands for "Common Vulnerabilities and Exposures" and is a list of publicly disclosed computer security flaws.
CVSS	Stands for "Common Vulnerability Scoring System". It is a free and open industry standard for assessing the severity of computer system security vulnerabilities.
DevOps	Stands for "Development and Operations". It is a set of practices and tools that are designed to remove barriers between traditionally siloed teams, development and operations.
Docker	An open-source software platform to create, deploy and manage virtualized application containers.
Dockerfile	A text document that contains all the commands a user could call on the command line to assemble an image.

ECR	Stands for Amazon "Elastic Container Registry", which is a fully managed container registry.
EKS	Stands for Amazon "Elastic Kubernetes Service", which is a managed service that can be used to run Kubernetes on AWS without installing, operating, and maintaining own Kubernetes control plane or nodes.
GIAM	Stands for "Global Identity and Access Management".
GPAM	Stands for "Global Privileged Access Management".
GKE	Stands for "Google Kubernetes Engine", which provides a managed environment for deploying, managing, and scaling containerized applications using Google infrastructure.
Image	An unchangeable, static file that includes executable code. It consists of system libraries, system tools, and other platform settings a software program needs to run on a containerization platform.
Istio	An open-source service mesh platform that provides a way to control how microservices share data with one another.
JML Process	Stands for "Joiners, Movers, Leavers Process", which describes different scenarios for new employees, employees that change their positions inside the company, and employees that leave the organization.
Kubernetes	A portable, extensible, open-source platform for managing containerized workloads and services.
MITRE ATT&CK	Stands for MITRE "Adversarial Tactics, Techniques, and Common Knowledge", which is a globally accessible knowledge base of adversary tactics and techniques based on real-world observations.
Namespace	In container context, a namespace is a high-level abstraction that subdivides cluster resources with different access controls applied to them. In the Linux context, a namespace is a low-level mechanism for isolating the machine resources that a process is aware of.
NIST	Stands for "National Institute of Standards and Technology". Its mission is to "promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life".
Node	A logical collection of IT resources that supports one or more containers. Nodes contain the necessary services to run pods, communicate with master components, configure networking and run assigned workloads.
OPA	Stands for "Open Policy Agent", which is an open-source, general-purpose policy engine that unifies policy enforcement across the stack.
Orchestration	Container orchestration is the automatic process of managing or scheduling the work of individual containers for applications based on microservices within multiple clusters.

OWASP	Stands for "Open Web Application Security Project" and is a non-profit organization with the aim of improving the security in software.
PaaS	Stands for "Platform as a Service" and is a cloud computing model where a third-party provider delivers hardware and software tools to users over the Internet.
Pod	A group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. It represents a single instance of a running process in a cluster.
PV	Stands for "Persistent Volume", which is a piece of networked storage in the cluster that has been provisioned by an administrator. It is a cluster-wide resource that can be used to store data in a way that persists beyond the lifetime of a pod.
RBAC	Stands for "Role-based access control" and is an access policy based on roles and privileges.
rkt	Pronounced as "Rocket". It is an open-source software that can be used to isolate applications in containers using operating system virtualization.
runC	A lightweight universal container runtime, developed by the Open Container Initiative (OCI).
SDLC	Stands for "Systems Development Life Cycle" and is a methodology for developing software.
Secrets	Secure objects which store sensitive data, such as passwords, OAuth tokens, and SSH keys.
Service Mesh	A dedicated infrastructure layer that controls service-to-service communication over a network.
SGID	Stands for "Set-Group Identification", which is a special permission bit available in Linux.
SIEM	Stands for "Security Information and Event Management". It is a technology, which supports threat detection, compliance and security incident management.
SOD	Stands for "Segregation of Duties" and is an internal control to prevent mistakes and fraud by dividing responsibilities.
SSL	Stands for "Secure Sockets Layer." SSL is a secure protocol developed for sending information securely over the Internet.
SUID	Stands for "Set-User Identification", which is a special permission bit available in Linux.
TLS	Stands for "Transport Layer Security", which is a data encryption technology that provides secure data transfers.
URL	Stands for "Uniform Resource Locator".

VM	Stands for “Virtual Machine”.