



# Container Image Creation and Deployment Guide

## Version 2, Release 0.6

## 02 November 2020

## Developed by DISA for the DoD


DEPARTMENT OF DEFENSE
UNITED STATES OF AMERICA


DEFENSE INFORMATION SYSTEMS AGENCY

## Trademark Information

Names, products, and services referenced within this document may be the trade names, trademarks, or service marks of their respective owners. References to commercial vendors and their products or services are provided strictly as a convenience to our users, and do not constitute or imply endorsement by DISA of any non-Federal entity, event, product, service, or enterprise.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Page**

# 1. OVERVIEW

## 1.1 Scope

The Container Image and Deployment Guide will provide the technical requirements for container image creation and deployment within a container platform. Sections 2 and 3 are a guide for security practices to limit the introduction of security flaws during the container image build process and to limit the security footprint of the container once instantiated into a container platform environment. Section 4 describes Development, Security and Operations (DevSecOps) in general, since each project will have different needs dependent on the service being developed. The guide does not give specific tasks or settings to implement.

This guide does not address specific technologies that can be deployed within a container, such as a database, webserver, or load balancer. For guidance on technologies being deployed within the container, refer to the Security Technical Implementation Guide (STIG) for a specific technology or operating system. When a STIG is not available, the Security Requirements Guide (SRG) for the technologies should be used.

## 1.2 Technology

To understand the scoping of this Container Image and Deployment document and the later requirements, it is necessary to define related terms. The next few sections define the key terms used throughout this document.

### 1.2.1 Container Image

A container image (i.e., image) is an immutable file that contains executable code that can run as an isolated process within a container platform. It is made up of system libraries and tools, and other platform specific settings and software the container needs to run on the container platform. The container image is not the executing process, but the immutable file. Once the container image is instantiated within the container platform, it is called a container; this container shares the operating system kernel of the hosting system to execute. The container shares other resources on the host or on external devices depending on the service the container offers. For additional guidance on container images and security please see *NIST Special Publication 800-190*[1].

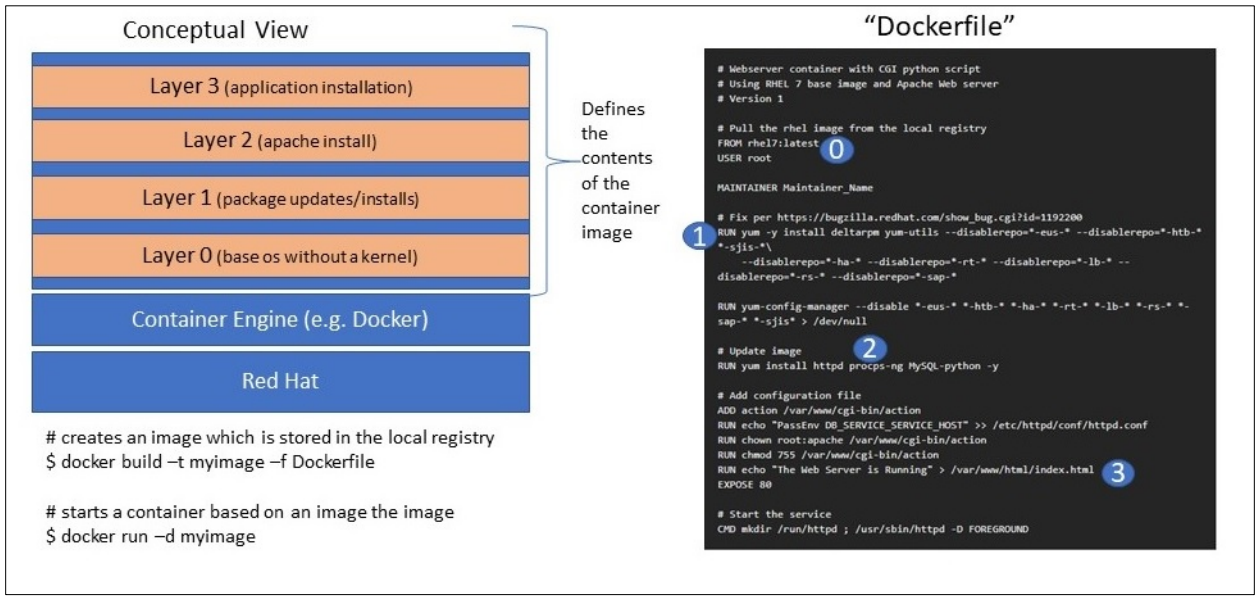
### 1.2.2 Container Image Layer

A container image layer is created when instructions within a configuration or build file are executed to create the image. Each image layer reflects a change made to the base image. Images that have no parent image and usually only have an Operating System are used to build the new

[1] National Institute of Standards and Technology, "Application Container Security Guide, NIST Special Publication 800-190. (SP 800-190)" September 2017, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

child image. It is also important to note that image layers are read-only except for the last layer, which is read/write. This is often called the "application layer", because images are usually built with the application, the most volatile in the build process, installed last. This takes advantage of the fact that layers reflect changes in the image creation and, when recreating an image, the build process does not build lower layers if they have not changed. The build begins where the first layer with a change is encountered. Figure 1-1 details how a build file, in this case, a Dockerfile is turned into layers within a container image.

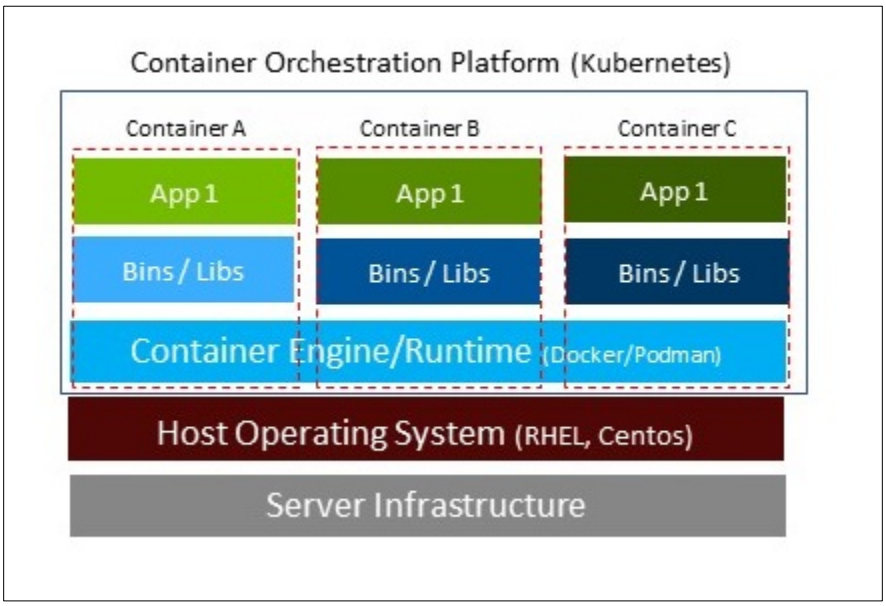
**Figure 1-1 Container Layers**




### 1.2.3 Container Platform

The container platform is the software used to orchestrate the execution of the container. Examples of commercial container platforms are Kubernetes, OpenShift, Konvoy, Tanzu, Docker, and Rancher. This platform may have other services to aid in the overall orchestration such as routing, DNS, and firewalls. For security settings specific to the container platform, review the specific technology Security Technical Implementation Guide (STIG) for the container platform, or if one does not exist, view the Container Platform Security Requirements Guides (SRG) and best practices.

**Figure 1-2 Container Platform**




As container development grows and container runtimes become more popular, standards are being developed to allow images to move between vendor environments more easily. One such standard is the Open Container Initiative (OCI). OCI is creating an open governance structure for creating industry standards for container formats and runtimes. The project was started by container industry leaders such as Docker and CoreOS. Produced so far are the specifications for container runtimes, called runtime-spec, and a container image specification called image-spec. Container images should adhere to the OCI Image Format Specification to ensure portability. For additional information see the *DoD Enterprise DevSecOps Reference Design v1.0*[2] section 5.1.1.

## 2. CONTAINER IMAGE CREATION

During the build process of the container image, security measures with measurable or definable settings can be implemented, along with non-measurable security measures that are not definable settings, but processes or techniques that lead to a more secure container image. It is important to note that this document signifies best practices for container creation and deployment, it is no replacement for a STIG or SRG. Furthermore, waivers may be required from the organization's security team in some cases where the requirement cannot be followed completely.

### 2.1 The Container Image Must Be Built with the SSH Server Daemon Disabled

A container image must only enable the applications needed for the service being implemented by the container image. The Secure Shell (SSH) daemon is run on a server to allow the system administrator (SA) to look at logs, change configurations, install patches, or backup data. A new

[2] Department of Defense CIO, "DoD Enterprise DevSecOps Reference Design." August 12, 2019, https://dodcio.defense.gov/Library/

container must be created when new packages are needed for the service, security patches are installed, or configurations change. Data backups and restoration are done by storing data to persistent volumes that can be backed up and restored when needed. If logs need to be viewed, the SA can attach them to the container or logs sent to external logging tools.

**IA Control:** CM-7 a **CCI:** CCI-000381

## 2.2 The Container Image Must Be Created to Execute as a Non-Privileged User

Containers must run as a non-privileged account. Allowing a container to run as a privileged user leads to containers that can access host system-protected resources and execute privileged commands. If the container also mounts host system directories, it can act as a privileged user on the host system.

To implement the practice of a container running as a non-privileged user, the container image must be built with a non-privileged user in the build file, often referred to as the USER statement. When a container image is not built in this way, user namespace mapping can be used to map the container runtime user to a non-privileged account.

**IA Control:** AC-6 (10) **CCI:** CCI-002235

## 2.3 The Container Image Must Have Permissions Removed from Executables that Allow a User to Execute Software at Higher Privileges

Privileged execution is a serious issue to address when securing a container. Privilege escalation occurs when a user or application gains access to the privileges of another user. The user or application can use the elevated privileges to steal confidential data or cause harm to the hosting system. For Linux container images, removing setuid and setgid permissions on executables within the container image when it is created prevents privilege escalation attacks within the container when it is deployed.

**IA Control:** AC-6 (10) **CCI:** CCI-002235

## 2.4 The Container Image Must Be Built Using Commands that Result in Known Outcomes.

Besides loading packages from trusted sources, it is important to use consistently performing commands with known outcomes when building the container image. For example, the COPY command is a safe build command that puts files inside a container image during creation. This command copies the files from the local host machine to the container file system. The ADD instruction command can be used to put files into a container image and could potentially retrieve files from remote Uniform Resource Locators (URLs) to perform operations such as unpacking. By performing operations or tasks outside the simple task of copying a file, the ADD instruction introduces risks such as adding malicious files from URLs without scanning and unpacking possible vulnerabilities. It is always good practice to ensure containers can be built

without connection to the internet. This ensures packages are loaded and built from trusted sources.

**IA Control:** CM-7a **CCI:** CCI-000381

### 2.5 The Container Image Must Only Expose Non-Privileged Ports

Privileged ports, those below 1024, can only be used by privileged users such as root on Linux. For a container to have access to these ports, it must be running as a privileged user, however this creates a security risk since the container could perform any privileged actions on the host system, especially if a malicious user were to gain access to the container. To handle services implemented within a container that typically execute on privileged ports, such as a web server listening on port 80, use the container platform to map the privileged port to the unprivileged port within the container, e.g., map port 80 to port 8080 within the container.

**IA Control:** CM-7 (1) (b) **CCI:** CCI-001762

### 2.6 The Container Image Must Be Built With a Process Health Check

Ensuring a container service is still executing and handling workloads is necessary to validate service availability. Adding the health check instruction, HEALTHCHECK, within Docker, to the container image, provides a mechanism for the container platform to periodically check the running container instance against that instruction to ensure the instance is still working. Based on the reported health status, the container platform can then exit a non-working container and instantiate a new one bringing the service back to an available state. Short lived containers that do not require a health check can be submitted for a waiver.

**IA Control:** SC-5 **CCI:** CCI-002385

### 2.7 Container Image Creation Must Use TLS 1.2 or Higher for Secure Container Image Registry Pulls

Image authenticity can be ensured by pulling images from trusted and approved sources, protecting the image during transmission, and validating the image once received. To protect the image during transmission, it is necessary to use a transport protocol that encrypts traffic and cannot be intercepted and modified.

Currently, sensitive data including container images, is protected by using Transport Layer Security (TLS) version 1.2 or newer. Further guidance can be found in *NIST Special Publication 800-52 Rev2*[3].

**IA Control:** SC-8 **CCI:** CCI-002418

[3] National Institute of Standards and Technology, "Guidelines for the Selection, Configuration and Use of Transport Layer Security (TLS) Implementations (SP 800-52 Rev. 2)." August 2019, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

## 2.8 The Container Image Should Be Built with Minimal Cached Layers

Reducing the number of container image layers is a best practice when creating container images, but often leads to container image creation files that contain a single line made up of multiple instructions. The container image contains fewer layers, but the layers can become cached during the build process. The builder can then use the cached layer when rebuilding the image and for other images during the build process if the same instruction is used. By using cached layers, the builder could potentially deny fresh updates to be included in later container image builds.

**IA Control:** SI-2 (6) **CCI:** CCI-002617

## 2.9 The Container Image Must Be Created Without Confidential Data in the Build Files

When container images are created with private or classified data (e.g., passwords, tokens, and keys) stored within the build file, the data can be backtracked easily by using native commands for the container platform or by using various external tools. These build files are often shared, and without removal of the confidential data, would become exposed and potentially exploited. Even if the data is deleted later, it can be retrieved from the container image history. When authentication is necessary during the container image build process, secrets stored in authentication management service must be used.

**IA Control:** CM-6 b **CCI:** CCI-000366

## 2.10 The Container Images Must Be Created from Signed Base Images

Image building must start with a trusted image with known content from a trusted source. Digital signatures of container images provide the capability to trust container image content when container images are sent to and received from remote registries. These signatures allow client-side verification of the integrity and publisher of specific image tags and ensures provenance of container images.

**IA Control:** CM-5 (3) **CCI:** CCI-001749

## 2.11 The Container Image Must Be Created with Verified Packages

Container image content trust starts with the installation of known and trusted packages. Verifying the authenticity of packages during each container image build step ensures the container image remains secure. Packages installed from untrusted or unknown sources could be tampered with and could potentially have malicious code or unknown vulnerabilities that could be exploited.

**IA Control:** CM-5 (3) **CCI:** CCI-001749

## 2.12 The Container Image Must Be Created with Only Essential Capabilities

To minimize the attack surface of a container, only those capabilities essential to the service offered by the container should be installed. When container images become bloated with unnecessary software, the attack surface grows along with the need for subsequent security patch maintenance. This also negates the concept of containers being microservices.

**IA Control:** CM-7 a **CCI:** CCI-000381

## 2.13 The Container Image Must Only Enable Ports Used for the Service Being Implemented

To minimize the attack surface, each container image should be a microservice, implementing only one service where possible. Many services require enabled ports for communication. Exposing ports unnecessary for service implementation allows the container to expand the attack surface by giving external malicious users information regarding applications and their versions that are enabled within the container, and possible vulnerabilities that might be used to attack the container.

**IA Control:** CM-7 (1) (b) **CCI:** CCI-001762

## 2.14 The Container Image Must Be Built from a DoD Approved Base Image

During transmission, a container base image can be changed by a malicious user. To thwart this effort, the hash of the container base image must be compared against the known hash from the trusted source.

**IA Control:** SC-8 (2) **CCI:** CCI-002422

## 2.15 Container Images No Longer in Use Due to Updated Versions Must Be Removed

Container images are stored within the container platform's registry. When deploying a container, ensure the container being instantiated is the desired container. One way to instantiate the correct container image is by using tags, but tags can be manipulated to force a container image with known vulnerabilities to be used when a more secure image is available. To help ensure newer and more secure images are always instantiated, older images must be deleted from the registry. This includes Certificate to Field (CTF) approvals where critical vulnerabilities may exist in older versions and need to be revoked.

When building a container image, ensure the base image being used for the build is the desired image. Accidentally building with an older and more vulnerable base image leads to those vulnerabilities perpetuating into child images.

**IA Control:** SI-2 (6) **CCI:** CCI-002617

### 2.16 The Container Image Must Implement Any STIG or SRG Guidance Relevant to the Container Service

Practices to handle container image build and deployment must be followed, as the container is also implementing a service. To fully harden container images, any STIG or SRG documents for the service being implemented must be applied during the container image creation. In cases where a traditional STIG may not entirely apply for a scan of a container, or a container may need to run as root as an exception, a waiver will need to be approved. In these cases, until container guidance is created for each technology, waivers will need to be approved for each exception as needed.

**IA Control:** CM-6 b **CCI:** CCI-000366

### 2.17 The Container Image Must Be Created from a Trusted and Approved Source

Choosing images from trusted and approved sources is the beginning of creating container images with known security postures. When unknown public sources are used, the container may contain malicious code and services that result in undesired implementations. Trusted sources must use valid registry certificates to sign their hosted images. Consumers of these images must verify the authenticity of signed container images when downloaded from the trusted source. Please see section 6 Appendix A for a list of trusted sources.

**IA Control:** IA-5 (2) (a) **CCI:** CCI-000185

### 2.18 The Container Image Must Be Clear of Embedded Credentials

A container may require passwords, secrets, or keys to connect to other applications such as backend databases. The credentials must be kept externally, fetched by the application, and not stored in the container image. If the credentials are stored within the container image, anyone with access to the image can parse the credentials.

**IA Control:** IA-5(7) **CCI:** CCI-002367

## 3. CONTAINER DEPLOYMENT

Container deployment occurs when the container image is instantiated within a container platform. While much of the container's security is built into the container image, there are security practices and settings that should be considered to protect the hosting system and the container platform from the container itself. Some of the security measures have measurable or definable settings. Other security measures are non-measurable security practices that are not definable settings, but processes or techniques that lead to a more secure container. It is important to note that this document signifies best practices for container creation and deployment, it is no replacement for a STIG or SRG. Furthermore, waivers may be required from the organizations security team in some cases where the requirement cannot be followed completely.

### 3.1 A Container Must Not Mount the Container Platform's Registry Endpoint

The registry socket is used by client tools to interact with the registry. If a container can mount the socket, it can execute registry commands such as running images, deleting images, or pulling nefarious images into the registry.

**IA Control:** SC-4 **CCI:** CCI-001090

### 3.2 A Container Must Be Limited in Available System Calls

Secure computing mode (SECCOMP) is a security facility in the Linux Kernel. The facility enables a security posture where containers are limited in the system calls available to the process. The default SECCOMP profile provides a default security posture for running containers while providing wide application compatibility.

**IA Control:** SC-4 **CCI:** CCI-001090

### 3.3 Enable PIDs Control Groups to Limit and Account for Container Resource Usage

Control groups (cgroups) is a Linux kernel feature that limits, accounts for, and isolates resource usage (central processing unit (CPU), memory, disk I/O, network, etc.) of a container. By implementing cgroups, each container is guaranteed a share of the resources, but more importantly, it guarantees that a single container does not bring the container platform down by exhausting resources.

**IA Control:** SC-4 **CCI:** CCI-001090

### 3.4 Sensitive Directories on the Host System Must Not Be Mounted by Containers

Sensitive data on the host system must be protected from containers running within the container platform. To protect the data, it is important to verify which directories can be, and are mounted by executing containers. If a container is granted access directly or inherits access to directories containing sensitive data when the container is not authorized to the data, the data can be used by the container for an attack or offloaded for later use. Examples on a Linux server of sensitive directories are /etc and /usr.

**IA Control:** SC-4 **CCI:** CCI-001090

### 3.5 The Container Should Have Resource Limits Set

The resource limit setting for a container provides a limit or maximum amount of resources a container can consume. Without this limit, a container can use all available resources, starving other containers of needed resources, causing a Denial of Service (DoS) across all the services executing within the container platform.

**IA Control:** SC-5 (1) **CCI:** CCI-001094

### 3.6 The Container Should Have Resource Request Set

Setting a container resource request limit allows the container platform to determine the best location for the container to execute. The container platform looks at the resources available and finds the location that will require the minimum resources for the container to execute. Examples of resources that can be specified are CPU, memory, and storage.

**IA Control:** SC-5 (2) **CCI:** CCI-001095

### 3.7 The Container root filesystem Must Be Mounted as Read-Only

Any changes to a container must be made by rebuilding the image and redeploying the new container image. Once a container is running, changes to the root filesystem should not be needed, thus preserving the immutable nature of the container. Any attempts to change the root filesystem are usually malicious in nature and can be prevented by making the root filesystem read-only.

**IA Control:** CM-5 (1) **CCI:** CCI-001813

### 3.8 The Container Must Have a Liveness Probe

A liveness probe checks for instances in which a container has run into a deadlock state. The container will appear as healthy from the process health check's point of view but will not be in a usable state. This check is performed at the container level for healthiness.

**IA Control:** SC-5 **CCI:** CCI-002385

### 3.9 The Container Must Have a Readiness Probe

Process health checks and liveness probes are helpful to determine if a container is unhealthy and needs to be restarted, however restarting does not always make the container healthy and may keep the container in an apparent unhealthy state (for example, when the container is starting and is not ready to receive any workload, including the health check or liveness probe). The readiness probe can also be used to monitor if a container is overloaded, latency has increased, and the container needs to be shielded from additional workloads.

**IA Control:** SC-5 **CCI:** CCI-002385

### 3.10 A Container Must Not Have Access to Operating System Kernel Namespaces

Namespaces provide a straightforward way of isolating containers. Using isolation, services within a container cannot see or access services running within other containers or on the container platform host system. When kernel namespaces are shared to containers, data can be

shared directly between services bypassing security measures, allowing containers to elevate their privileges.

The important namespaces to protect are the network (net) namespace, which virtualizes the network stack; the process ID (PID) namespace, which contains the process IDs of all running processes; the Inter-Process Communication (IPC) namespace, which isolates containers from using SysV inner-process communication; Unix Time Sharing (UTS), which would allow a container to appear to have different host and domain names to different processes; and the userID (user) namespace, which provides both privilege isolation and user identification segregation across containers.

**IA Control:** SC-4 **CCI:** CCI-001090

### 3.11 The Container Should Be Given Label Selectors to Help Define Container Execution Location and Type

Using label selectors, containers can be given priority based on where they are executed according to characteristics of the service or the user community. Examples of labels include the type of service, such as management versus customer, or the type of data on which the container operates, such as HR versus Sales or the DoD role of the container such as GPU vs CPU. Labeling the containers aids in the location of execution, reducing any data spillage or inadvertent resource sharing.

**IA Control:** SC-39 **CCI:** CCI-002530

# 4. DEVSECOPS

DevSecOps is a set of repeatable software development practices that combine software development (Dev), security (Sec), and operations (Ops) to shorten the development life cycle and deliver a secure product. Security practices are implemented within DevOps at both the Dev and the Ops phases and is not a separate phase in and of itself. By implementing a DevSecOps process, developers can now deliver software features, patches, and fixes more quickly to users because the process can be automated and repeatable. For more information refer to the *DoD Enterprise DevSecOps Reference Design v1.0*[4].

## 4.1 Development

Development is the phase in which development of the application is performed. This phase is typically broken into five categories: Coding, Building, Testing, Securing, and Publishing. These categories may have other names or be merged to become one category, but the same general tasks will exist within this phase. During development, any category can cause a return to a previous category.

- **Coding** is the process of application and infrastructure code development, code design and security review, configuration management of code through the use of source code management tools, and the use of static code analysis tools to locate coding flaws and security risks.
- **Building** includes the use of continuous integration tools, reporting build statuses, pushing container images to repos, image signing, and the creation of image checksums
- **Testing** involves using testing tools to perform continuous testing that provides quick and timely feedback on business risks
- **Securing** is the process of using security tools to perform continuous image scanning for vulnerabilities and security findings, and the creation of whitelist
- **Publishing** is the process of putting together the artifacts from previous steps and posting those to the artifact repository, performing application pre-deployment staging, manifest creation, and image signing

## 4.2 Operations

The operations phase consists of categories that take place in production to keep the application available and secure. Operations is usually broken into three categories: Releasing, Configuring, and Monitoring. If during any of these phases a security issue is found or a threshold is met during monitoring, a fix can be requested. Because configuration management in the coding category during development is the single source of truth, changes implemented trigger a new pipeline to execute pushing upgrades automatically into production.

- **Releasing** involves executing change management procedures, getting release approvals, and then implementing the release automation processes

[4] Department of Defense CIO, "DoD Enterprise DevSecOps Reference Design." August 12, 2019, https://dodcio.defense.gov/Library/

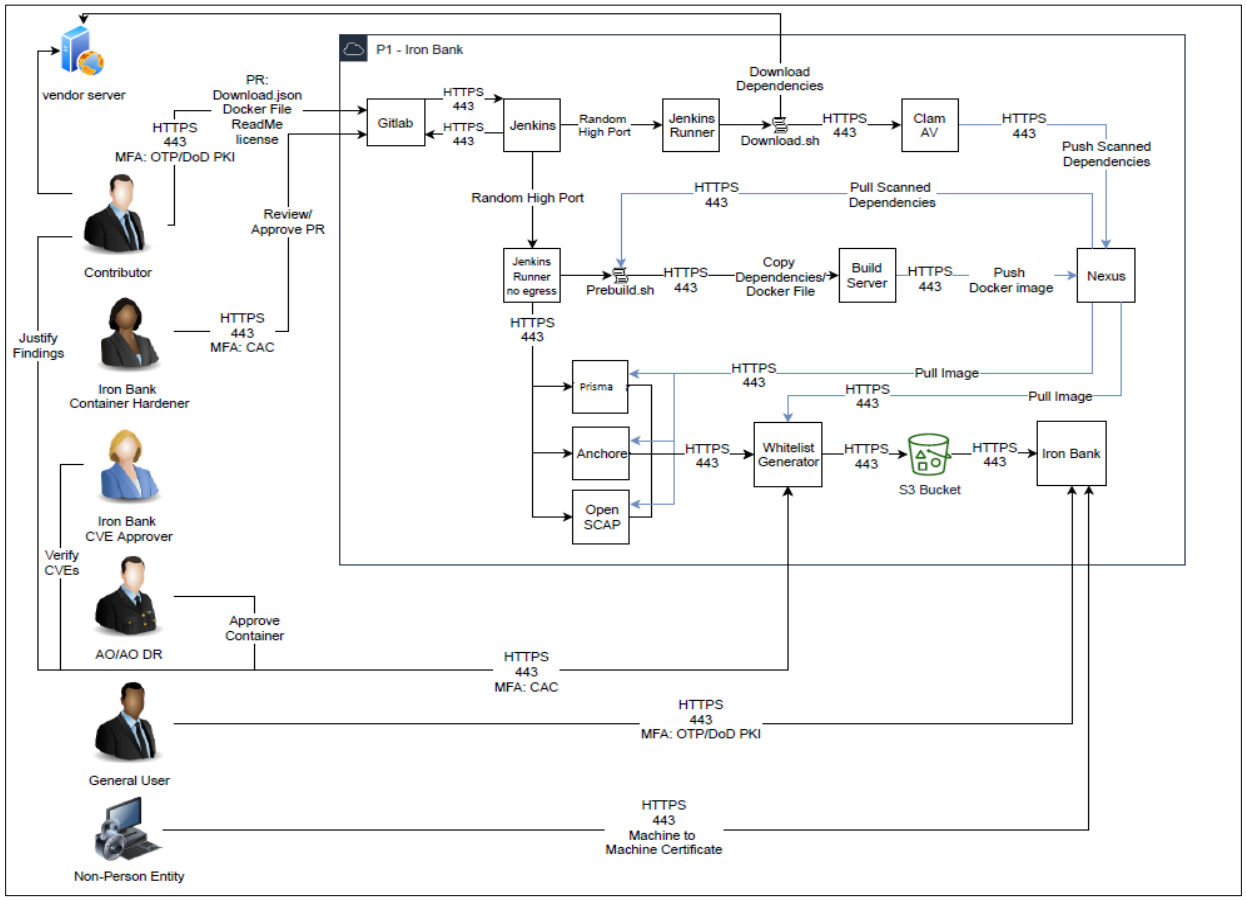
- **Configuring** is the process of implementing any infrastructure configurations and the use of infrastructure as code tools
- **Monitoring** is the continuous process of monitoring the application security posture, the applications performance, end-user experience, and infrastructure drift

## 4.3 DevSecOps Pipeline

To implement a sustainable and repeatable DevSecOps process, organizations will implement a pipeline in which DevSecOps steps are applied through code in tools such as Jenkins, Bamboo, or GitLab etc. with continuous integration and continuous delivery or continuous deployment (CI/CD). To fulfill tasks the pipeline cannot perform itself, external tools are called and executed from within the pipeline process. The process can better be understood by breaking a production pipeline into its pieces.

The DevSecOps pipeline in Figure 4-1 is an example of a CI/CD pipeline, which takes a developed container from a contributor, rebuilds the container while controlling the inclusion of dependencies, scans the container for vulnerabilities, verifies the container is secure, and then releases the hardened container for public consumption. This pipeline is called the Iron Bank pipeline and is implemented by the Air Force. While this pipeline is used to verify and release hardened container images, the practices implemented would also apply to pipelines building software for release.

**Figure 4-1 Iron Bank CI/CD Pipeline**

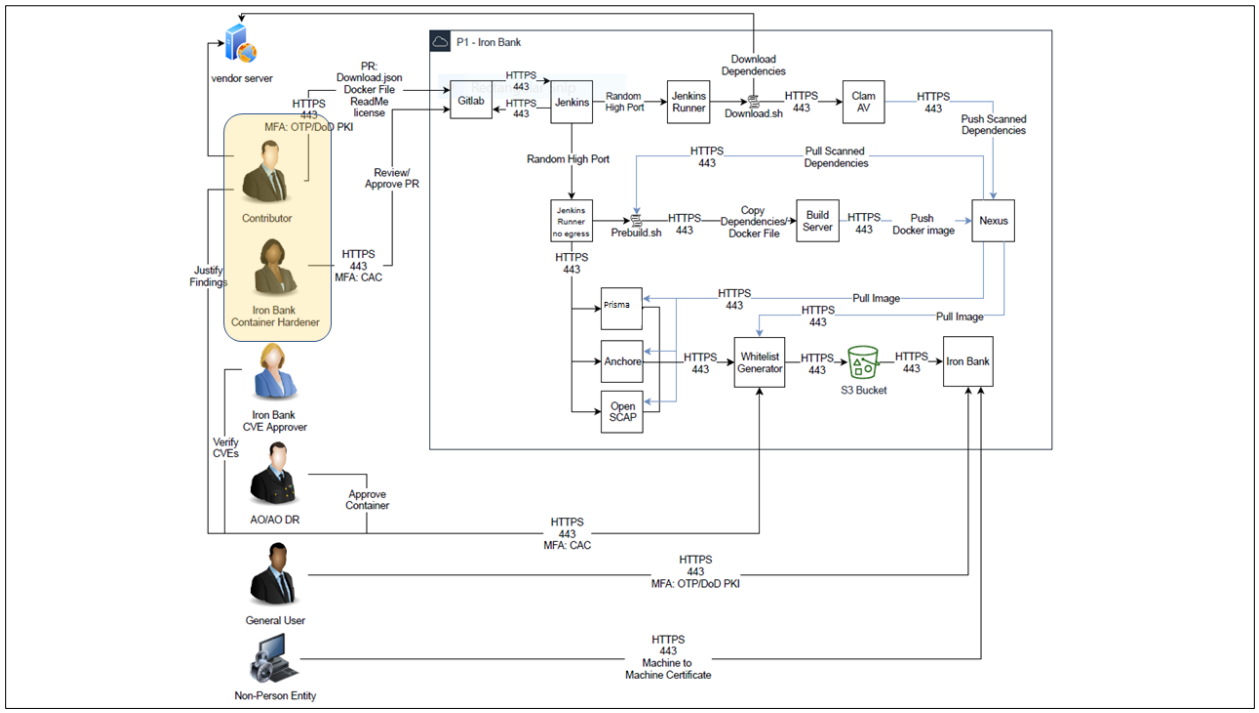


### 4.3.1 Coding and Testing

Within the Iron Bank diagram, the coding and testing tasks are done by the contributor and the Iron Bank Container Hardener. The contributor is responsible for developing the container and running tests against the container to validate that the container passes a set of test criteria set forth by the organization. Once the coding and testing are performed by the contributor, the Iron Bank Container Hardener works with the contributor to secure the container before the container is introduced to the Iron Bank pipeline.

To enable the Iron Bank pipeline to ingest the pull request (PR) from the contributor, standards are set for the form of the artifacts and what those artifacts are permitted to contain. A contributor will meet the requirements set forth by the Iron Bank pipeline for inclusion by following the security requirements in Section 2 of this document.

**Figure 4-2 Coding and Testing**

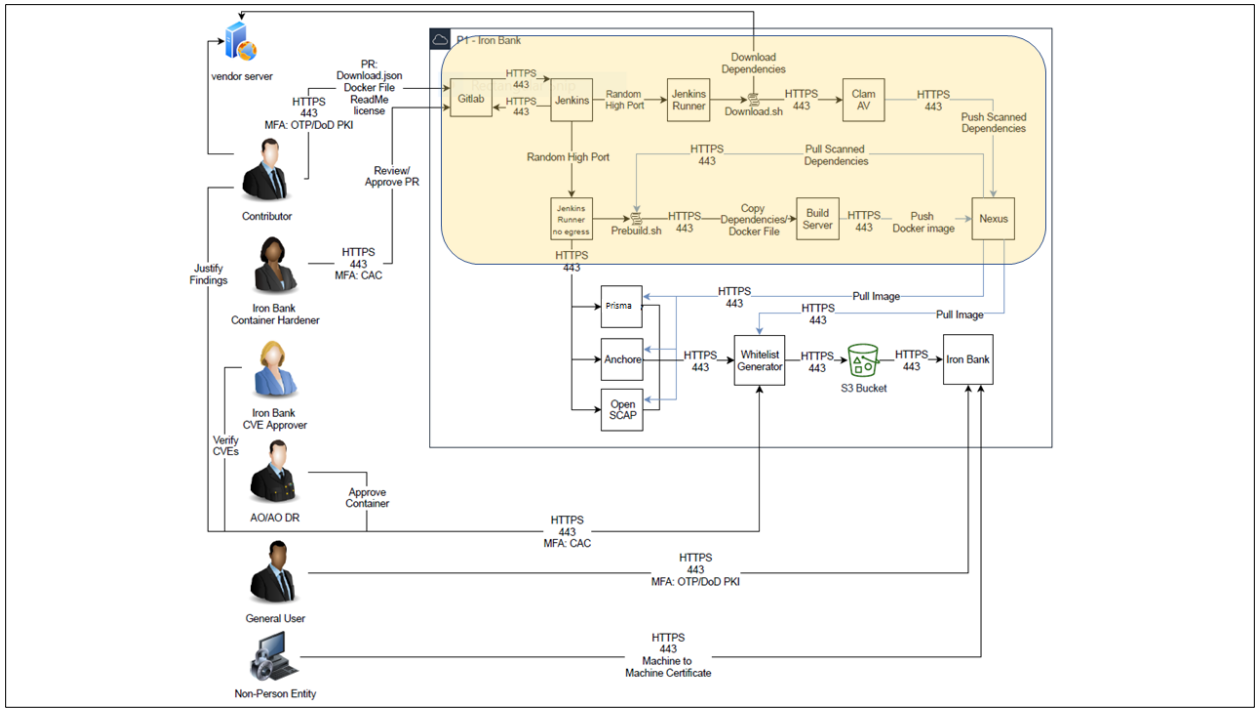



### 4.3.2 Building

During the build stage of the Iron Bank pipeline, it is necessary to control the inclusion of dependencies. Iron Bank can control the dependencies through the type of commands in the build files and the use of trusted sources. Though the dependencies come from trusted sources, they are run through Clam AntiVirus (AV) to check for viruses and malware. Once the dependencies are determined to be secure, they are moved to the artifact repository and there is no need to reach outside the Iron Bank infrastructure for the build of the container image. After the build completes, the new image is pushed to the artifact repository.

The Iron Bank pipeline is tailored for building hardened containers without having any awareness or concern for the type (i.e., java, python) of build being performed, so many of the steps in the build process reflect its general purpose.

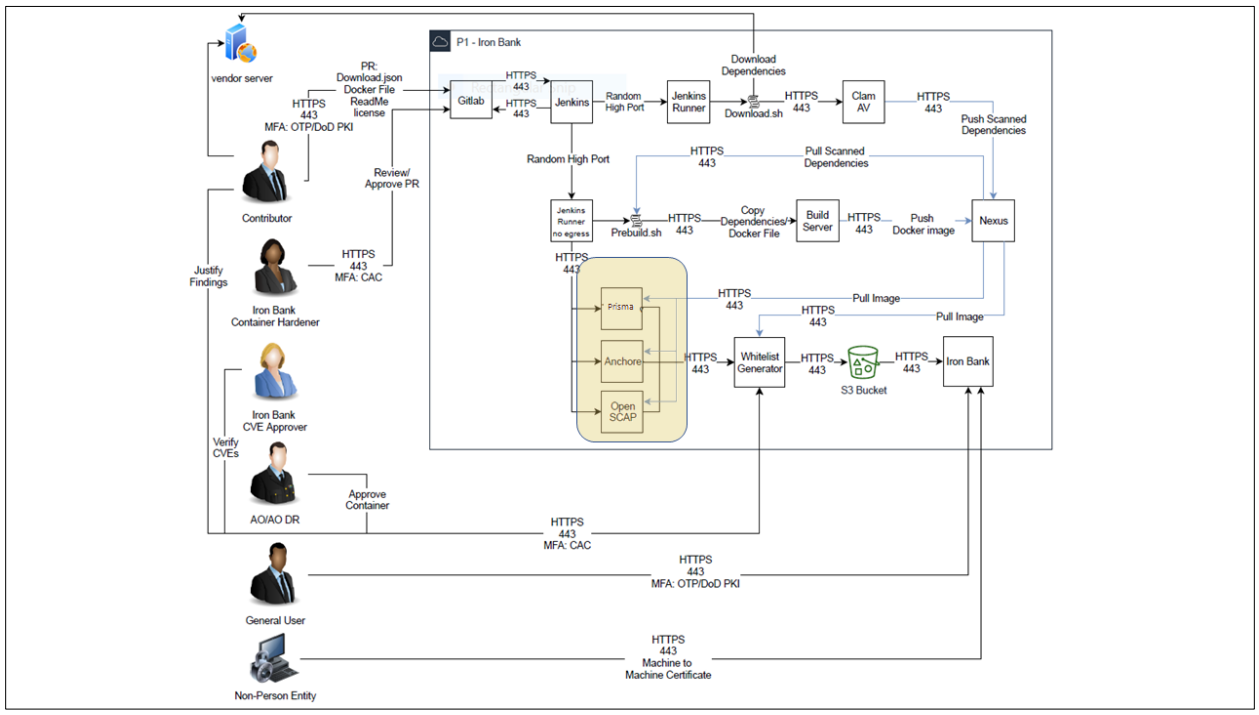
**Figure 4-3 Building**




### 4.3.3 Securing

Scanning for security vulnerabilities of a newly built container image is kicked off after the successful build of the container image within the pipeline. The new container image is scanned by security tools such as Prisma, Anchore, and OpenScap to determine what vulnerabilities are within the container image. It is necessary to utilize more than one tool to scan since results can differ. These tools generate reports showing the vulnerability, the fix, and severity. From these reports, the Iron Bank Common Vulnerabilities and Exposures (CVE) Approver and the contributor can determine which vulnerabilities need to be fixed before release during the publishing phase. By following the security requirements in Section 2, the vulnerabilities found by the security tools will be minimized allowing for a container image to require a smaller whitelist of findings.

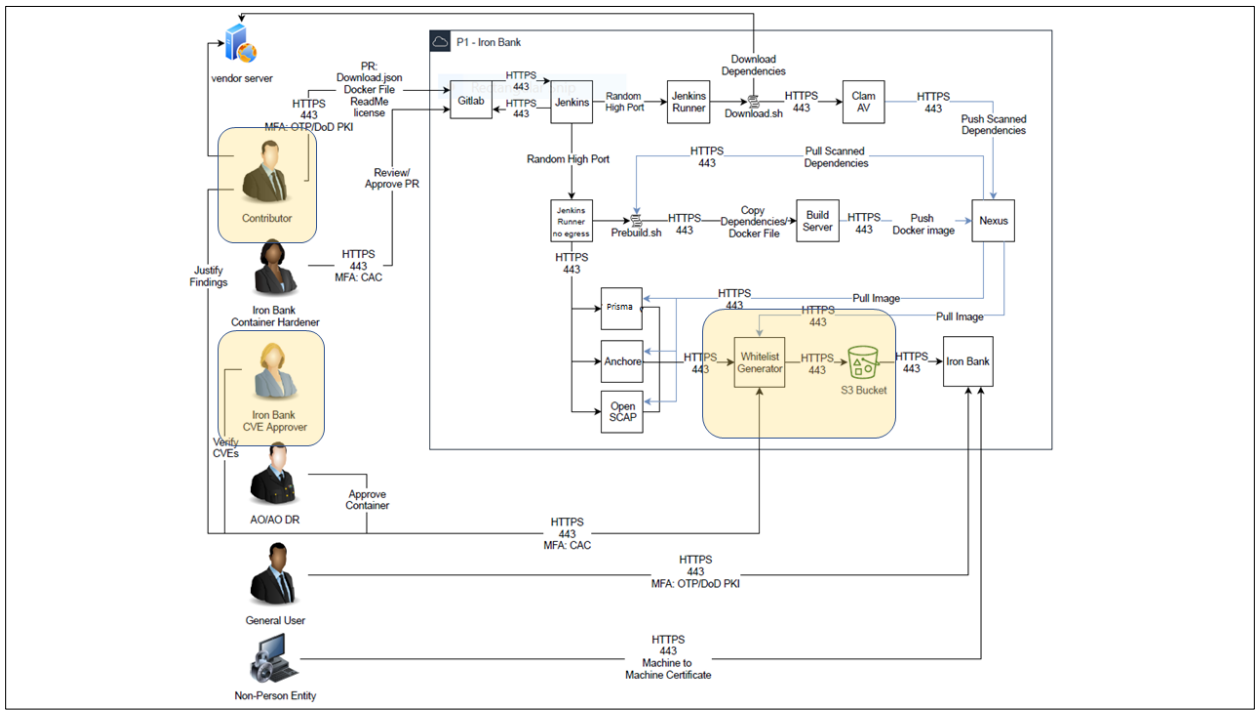
**Figure 4-4 Securing**




### 4.3.4 Publishing

Before releasing the container image to the public, any vulnerabilities found during the security scans must be addressed. Throughout this phase, the findings are reviewed by the contributor and the Iron Bank CVE Approver. Each vulnerability is either signed off by the Iron Bank CVE Approver, creating a whitelist of findings, or is resolved by the contributor. In cases where the contributor does not understand the vulnerability, the Iron Bank CVE Approver will work with the contributor to understand the vulnerability and resolve the vulnerability if necessary. If vulnerabilities within the container image are too severe to be released, the contributor must make the recommended fixes and create a new pull request to begin the process once again. The Iron Bank CVE Approver is the primary defense of the Department of Defense (DoD) networks.

**Figure 4-5 Publishing**

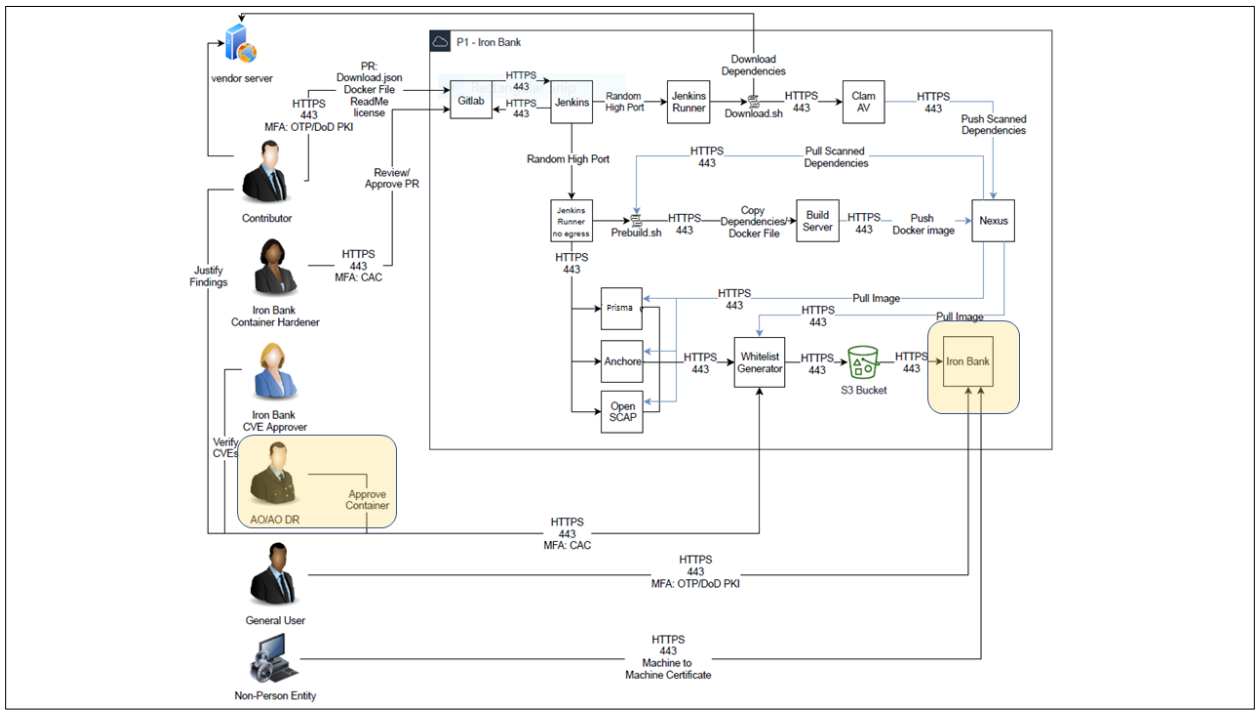



### 4.3.5 Releasing

The first step in releasing the container image to the public is getting approval from the Authorizing Official (AO). The AO is responsible for ensuring no container image that has been brought into the pipeline is made available to the public without proper security scanning and review. The AO is familiar with approving the risk behind container images.

Once approval is given, the pipeline will begin the process of moving the container image to the Iron Bank repository. The process will make available the container image and those artifacts created during the pipeline (e.g., scan reports, whitelists, and manifests), the README file and licenses from the contributor, the image will be signed, and a checksum generated.

**Figure 4-6 Releasing**




### 4.3.6 Configuring

The Iron Bank pipeline addresses configuration of Kubernetes and the platform through usage of Helm Charts & Operators as well as Kubernetes manifests. Configuration is the process of creating an infrastructure for the container. Because Iron Bank is not instantiating the container images released through the pipeline, there is no infrastructure to create; however, the pipeline could create the code to instantiate the infrastructure required for the container. By moving the code for the infrastructure through the pipeline with the container image, the infrastructure is also validated for security and function. The infrastructure will follow the same configuration management practices used by the application code.

### 4.3.7 Monitoring

Iron Bank continuously monitors the Iron Bank repository using a Sidecar Container Security Stack (SCSS). Monitoring can perform continuous scans of the Iron Bank repository and in a production environment where container images are instantiated; the executing services and infrastructure can also be scanned. Performing continuous scans of the repository and running services allows new vulnerabilities or security risks to be quickly detected and addressed. Once addressed, new builds can be initiated with the goal of quickly releasing new secure container images.

## 5. SUMMARY

The configurations and practices within this guide are meant to show what is needed to have a secure container through the complete lifecycle. Some of the guidance is measurable and can be checked by tools before a container image is instantiated within a production environment or while the container is executing, and other guidance is non-measurable but should be followed to reduce accidental security flaws from being introduced into the container lifecycle.

This document does not address the host operating systems, pipeline, and security tools, the container platform, or any other services offered by the container platform. These services need to be secured through the appropriate STIG, SRG, or best practice guidance to have a fully secure environment.

# 6. APPENDIX A DOD BASE CONTAINER IMAGE APPROVED SOURCES (DBCIAS)

To ensure container base images are from approved sources, a container base image must be downloaded from a DBCIAS as a baseline.

The following sources are currently approved and should be used in order of priority:

- Iron Bank/DCAR (approved DoD-wide; trusted)
- Product vendor proprietary repository (untrusted)
- Docker Hub (untrusted)
- Red Hat Container Repository (untrusted)

For additional information please refer to the "DISA DevSecOps Enterprise Container Hardening Guide".