DoD Enterprise DevSecOps Reference Design:
CNCF Kubernetes

March 2021
Version 2.0

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.
Document Set Reference

DoD Enterprise DevSecOps Reference Design
- Specific CNCF Kubernetes Tools & Technologies
- Specific Architecture Requirements
- cATO Status: {Draft / Provisional / Reciprocity}

Future Reference Designs
(Pending Interest, Time & Money)
- DoD Enterprise DevSecOps Reference Design
  - Low Code-No Code
  - Legacy Modernization
  - Serverless / CSP Managed Service
Document Approvals

Approved by:

Nicolas Chaillan
Chief Software Officer, Department of Defense, United States Air Force, SAF/AQ
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 readers, and do not constitute or imply endorsement by the Department of any non-Federal entity, event, product, service, or enterprise.
Figures

Figure 1: Kubernetes Reference Design Interconnects .......................................................... 11
Figure 2: Container Orchestrator and Notional Nodes ......................................................... 12
Figure 3: Sidecar Container Relationship to Application Container ...................................... 13
Figure 4: Software Factory Implementation Phases .............................................................. 17
Figure 5: Containerized Software Factory Reference Design ............................................... 20
Figure 6: DevSecOps Platform Options .............................................................................. 21
Figure 7: Software Factory - DevSecOps Services ................................................................. 22
Figure 8: Logging and Log Analysis Process ...................................................................... 31

Tables

Table 1 Sidecar Security Monitoring Components ............................................................... 15
Table 2: CD/CD Orchestrator Inputs/Outputs ....................................................................... 18
Table 3: Security Activities Summary and Cross-Reference ............................................... 23
Table 4: Develop Phase Activities ....................................................................................... 23
Table 5: Build Phase Tools .................................................................................................. 23
Table 6: Build Phase Activities ........................................................................................... 24
Table 7: Test Phase Tools .................................................................................................... 24
Table 8: Test Phase Activities ............................................................................................. 25
Table 9: Release and Deliver Phase Tools ........................................................................... 25
Table 10: Release and Deliver Phase Activities ................................................................. 25
Table 11: Deploy Phase Tools .............................................................................................. 26
Table 12: Deploy Phase Activities ....................................................................................... 27
Table 13: Operate Phase Activities ...................................................................................... 27
Table 14: Monitor Phase Tools ............................................................................................ 28
Table 15: CSP Managed Service Monitoring Tools ............................................................. 28
1  Introduction

1.1  Background

Modern information systems and weapons platforms are driven by software. As such, the DoD is working to modernize its software practices to provide the agility to deliver resilient software at the speed of relevance. DoD Enterprise DevSecOps Reference Designs are expected to provide clear guidance on how specific collections of technologies come together to form a secure and effective software factory.

1.2  Purpose

This DoD Enterprise DevSecOps Reference Design is specifically for Cloud Native Computing Foundation (CNCF) Certified Kubernetes implementations. This enables a Cloud agnostic, elastic instantiation of a DevSecOps factory anywhere: Cloud, On Premise, Embedded System, Edge Computing.

For brevity, the use of the term ‘Kubernetes’ or ‘K8s’ throughout the remainder of this document must be interpreted as a Kubernetes implementation that properly submitted software conformance testing results to the CNCF for review and corresponding certification. The CNCF lists over 90 Certified Kubernetes offerings that meet software conformation expectations.¹

It provides a formal description of the key design components and processes to provide a repeatable reference design that can be used to instantiate a DoD DevSecOps Software Factory powered by Kubernetes. This reference design is aligned to the DoD Enterprise DevSecOps Strategy, and aligns with the baseline nomenclature, tools, and activities defined in the DevSecOps Fundamentals document and its supporting guidebooks and playbooks.

The target audiences for this document include:

- DoD Enterprise DevSecOps capability providers who build DoD Enterprise DevSecOps hardened containers and provide a DevSecOps hardened container access service.
- DoD Enterprise DevSecOps capability providers who build DoD Enterprise DevSecOps platforms and platform baselines and provide a DevSecOps platform service.
- DoD organization DevSecOps teams who manage (Instantiate and maintain) DevSecOps software factories and associated pipelines for its programs.
- DoD program application teams who use DevSecOps software factories to develop, secure, and operate mission applications.
- Authorizing Officials (AOs).

This reference design aligns with these reference documents:

---

1.3 DevSecOps Compatibility

This reference design asserts version compatibility with these supporting DevSecOps documents:

- DevSecOps Tools and Activities Guidebook, Version 2.0.

1.4 Scope

This reference design is product-agnostic and provides execution guidance for use by software teams. It is applicable to developing new capabilities and to sustaining existing capabilities in both business and weapons systems software, including business transactions, C3, embedded systems, big data, and Artificial Intelligence (AI).

*This document does not address strategy, policy, or acquisition.*

---


6 White House, "Presidential Executive Order on Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure (EO 1380)," May 11, 2017.


1.5 Document Overview

The documentation is organized as follows:

- Section 1 describes the background, purpose and scope of this document.
- Section 2 identifies the assumptions relating to this design.
- Section 3 describes the DevSecOps software factory interconnects unique to a Kubernetes reference design.
- Section 4 describes the containerized software factory design.
- Section 5 captures the additional required and preferred tools and activities, building upon the DevSecOps Tools and Activities Guidebook as a baseline.

1.6 What’s New in Version 2

- Refactored the document’s overall structure to align with the shift to a DevSecOps Document Set approach.
2 Assumptions and Principles

This reference design makes the following assumptions:

- No specific Kubernetes implementation is assumed, but the selected Kubernetes implementation must have submitted conformance testing results for review and certification by the CNCF.
- Vendor lock-in is avoided by mandating a Certified Kubernetes implementation; however, product lock-in into the Kubernetes API and its overall ecosystem is openly recognized.

It is critically important to avoid the proprietary APIs that are sometimes added by vendors on top of the existing CNCF Kubernetes APIs. These APIs are not portable and may create vendor lock-in!

- Adoption of hardened containers as a form of immutable infrastructure results in standardization of common infrastructure components that achieve consistent and predictable results.
- This reference design depends upon a number of DoD Enterprise Services, which will be named throughout this document.

3 Software Factory Interconnects

The DevSecOps Fundamentals describes a DevSecOps platform as a multi-tenet environment consisting of three distinct layers: Infrastructure, Platform/Software Factory, and Application(s). Each reference design is expected to identify its unique set of tools and activities that exist at the boundaries between the discrete layers, known as Reference Design Interconnects. Well-defined interconnects in a reference design enable tailoring of the software factory design, while ensuring that core capabilities of the software factory remain intact.

*Figure 1: Kubernetes Reference Design Interconnects* identifies the specific Kubernetes interconnects that **must be present** in order to be compliant with this reference design. The specific interconnects include:

- Cloud Native Access Point (CNAP) above the Infrastructure layer to manage all north-south network traffic.
- Use of Kubernetes in each of the development environments.
- Clear identification of a locally centralized artifact repository to host hardened containers from Iron Bank, the DoD Centralized Artifact Repository (DCAR) of hardened and centrally accredited containers.
- Use of a service mesh within the K8s orchestrator to manage all east-west network traffic.
- Mandatory adoption of the Sidecar Container Security Stack (SCSS) to implement zero trust down to the container/function level, also providing behavior protection.

Each of these interconnects will be described fully next.

![Figure 1: Kubernetes Reference Design Interconnects](image)

### 3.1 Cloud Native Access Points

A Cloud Native Access Point (CNAP) provides a zero-trust architecture on Cloud One to provide access to development, testing, and production enclaves at Impact Level 2 (IL-2), Impact Level 4 (IL-4), and Impact Level 5 (IL-5). CNAP provides access to Platform One DevSecOps environments by using an internet-facing Cloud-native zero trust environment. CNAP’s zero trust architecture facilitates development team collaboration from disparate organizations. (A CNAP reference design is forthcoming.)

### 3.2 CNCF Certified Kubernetes

Kubernetes is a container orchestrator that manages the scheduling and execution of Open Container Initiative (OCI) compliant containers across multiple nodes, depicted in Figure 2. OCI is an open governance structure for creating open industry standards around both container

---

The container is the standard unit of work in this reference design. Containers enable software production automation in this reference design, and they also allow operations and security process orchestration.

Kubernetes provides an API that ensures total abstraction of orchestration, compute, storage, networking, and other core services that guarantees software can run in any environment, from the Cloud to embedded inside of platforms like jets or satellites.

The key benefits of adopting Kubernetes include:

- **Multimodal Environment**: Code runs equally well in a multitude of compute environments, benefitting from the K8s API abstraction.
- **Baked-In Security**: The Sidecar Container Security Stack is automatically injected into any K8s cluster with zero trust.
- **Resiliency**: Self-healing of unstable or crashed containers.
- **Adaptability**: Containerized microservices create highly-composable ecosystems.
- **Automation**: Fundamental support for a GitOps model and IaC speed processes and feedback loops.
- **Scalability**: Application elasticity to appropriately scale and match service demand.

The adoption of K8s and OCI compliant containers are concrete steps towards true microservice reuse, providing the Department with a compelling ability to pursue higher orders of code reuse across an array of programs.

3.3 Locally Centralized Artifact Repository

A Locally Centralized Artifact Repository is a local repository tied to the software factory. It stores artifacts pulled from Iron Bank, the DoD repository of digitally signed binary container images that have been hardened. The local artifact repository also likely stores locally developed artifacts used in the DevSecOps processes. Artifacts stored here include, but are not

---

limited to, container images, binary executables, virtual machine (VM) images, archives, and documentation.

The Iron Bank artifact repository provides hardened, secure technical implementation guide (STIG) compliance, and centrally updated, scanned, and signed containers that increases the cyber survivability of these software artifacts. At time of writing this reference design, over 300 artifacts were in Iron Bank, with more being added continuously.

Programs may opt for a single artifact repository and rely on the use of tags to distinguish between the different content types. It is also permissible to have separate artifact repositories to store local artifacts and released artifacts.

3.4 Sidecar Container Security Stack (SCSS)

The cyber arena is an unforgiving hostile environment where even a minute exposure and compromise can lead to catastrophic failures and loss of human life. Industry norms now recognize that a modern holistic cybersecurity posture must include centralized logging and telemetry, zero trust ingress/egress/east-west network traffic, and behavior detection at a minimum.

A cybersecurity stack is frequently updated as threat conditions evolve. A key benefit of a cybersecurity K8s sidecar container design is rapidly deployed updates without any recompilation or rebuild required of the microservice container itself. To support this approach, the SCSS is available from the Iron Ban repository as a hardened container that K8s automatically injects into each container group (pod). This decoupled architecture, shown in Figure 3, speeds deployment of an updated cyber stack without requiring any type of re-engineering by development teams.

![Figure 3: Sidecar Container Relationship to Application Container](image)

As shown in Figure 3, the sidecar can share state with the application container. In particular, the two containers can share disk and network resources while their running components are fully isolated from one another.
The complete set of sidecar container security monitoring components are captured in Table 1 on the next page. Capability highlights include:

- Centralized logging and telemetry that includes extract, transform, and load (ETL) capabilities to normalize log data.
- Robust east/west network traffic management (whitelisting).
- Zero Trust security model.
- Whitelisting.
- Role-Based Access Control.
- Continuous Monitoring.
- Signature-based continuous scanning using Common Vulnerabilities and Exposures (CVEs).
- Runtime behavior analysis.
- Container policy enforcement.
### Table 1 Sidecar Security Monitoring Components

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td></td>
</tr>
<tr>
<td><strong>Logging agent</strong></td>
<td>Send logs to a logging service</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Standardize log collection to a central location. This can also be used to send notifications when there is anomalous behavior.</td>
</tr>
<tr>
<td>Logging Storage and Retrieval Service</td>
<td>Stores logs and allows searching logs</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Place to store logs</td>
</tr>
<tr>
<td>Log visualization and analysis</td>
<td>Ability to visualize log data in various ways and perform basic log analysis.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Helps to find anomalous patterns</td>
</tr>
<tr>
<td>Container policy enforcement</td>
<td>Support for Security Content Automation Protocol (SCAP) and container configuration policies. These policies can be defined as needed.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Automated policy enforcement</td>
</tr>
<tr>
<td>Runtime Defense</td>
<td>Creates runtime behavior models, including whitelist and least privilege</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Dynamic, adaptive cybersecurity</td>
</tr>
<tr>
<td>Service Mesh proxy</td>
<td>Ties to the Service Mesh. Used with a microservices architecture.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Enables use of the service mesh.</td>
</tr>
<tr>
<td>Service Mesh</td>
<td>Used for a microservices architecture.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Better microservice management.</td>
</tr>
<tr>
<td>Vulnerability Management</td>
<td>Provides vulnerability management</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Makes sure everything is properly patched to avoid known vulnerabilities</td>
</tr>
<tr>
<td>CVE Service / Host Based Security</td>
<td>Provides CVEs. Used by the vulnerability management agent in the security sidecar container.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Makes sure the system is aware of known vulnerabilities in components.</td>
</tr>
<tr>
<td>Artifact Repository</td>
<td>Storage and retrieval for artifacts such as containers.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>One location to obtain hardened artifacts such as containers</td>
</tr>
<tr>
<td>Zero Trust model down to the container level</td>
<td>Provides strong identities per Pod with certificates, mTLS tunneling and whitelisting of East-West traffic down to the Pod level.</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Reduces attack surface and improves baked-in security</td>
</tr>
</tbody>
</table>
3.5 Service Mesh

A service mesh enhances cybersecurity by controlling how different parts of an application interact. It is a dedicated infrastructure layer baked-in to the software application itself; it is not a “bolt-on” component. Some of the specific capabilities of a service mesh in K8s including monitoring east-west network traffic, routing traffic based on a declarative network traffic model that can deny all network traffic by default, and dynamically injecting strong certificate-based identities without requiring access to the underlying code that built the software container. A service mesh also typically takes over ownership of the **iptables** in order to inject an mTLS tunnel with FIPS compliant cryptographic algorithms to further protect all data in motion.

Service mesh integration into the K8s cluster reduces the cyber-attack surface, and when coupled with behavior detection, it can proactively kill any container that is drifting outside of its expected operational norms. These capabilities restrict the ability of a bad actor to laterally move around within the K8s cluster and fully eliminate the ability of the bad actor to achieve escalated privileges. For these reasons, service mesh integration is a powerful component in ensuring the cyber survivability of the software factory and the containerized applications produced by the factory’s pipelines.
4 Software Factory K8s Reference Design

This section will discuss the software factory design required for this reference design. It is based on the DoD Enterprise DevSecOps Container Service offering to create a software factory using DevSecOps tools from hardened containers stored in Iron Bank.

All software factory implementations follow the DevSecOps philosophy and go through four unique phases: Design, Instantiate, Verify, Operate & Monitor. Figure 4 illustrates the phases, activities, and the relationships with the application lifecycle. Security is applied across all software factory phases. The SCSS must be used for cybersecurity monitoring of the factory in this reference design.

Figure 4: Software Factory Implementation Phases

The components of this reference design’s software factory must be instantiated as follows: A CSP-agnostic solution running a CNCF Certified K8s using hardened containers from Iron Bank. This design recognizes that K8s is well-suited to act as the engine powering a continuous integration/continuous delivery (CI/CD) orchestrator, coordinating multiple parallel DevSecOps pipelines. K8s manages pipeline creation, pipeline modification, overall pipeline execution, and finally pipeline termination.

The software factory leverages technologies and tools to automate the CI/CD pipeline processes defined in the DevSecOps lifecycle plan phase. There are no “one size fits all” or hard rules about what CI/CD processes should look like and what tools must be used. Each software team needs to embrace the DevSecOps culture and define processes that suit its software system architectural choices. The tool chain selection is specific to the software programming language choices, application type, tasks in each software lifecycle phase, and the system deployment platform.

DevSecOps teams create a pipeline workflow in the CI/CD orchestrator by specifying a set of stages, stage conditions, stage entrance and exit control rules, and stage activities. The CI/CD
orchestrator automates the pipeline workflow by validating the stage control rules. If all the entrance rules of a stage are met, the orchestrator will transition the pipeline into that stage and perform the defined activities by coordinating the tools via plugins. If all the exit rules of the current stage are met, the pipeline exits out the current stage and starts to validate the entrance rules of the next stage. Table 2 shows the features, benefits, and inputs and outputs of the CI/CD orchestrator.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI/CD orchestrator</td>
<td>Create pipeline workflow</td>
<td>Customizable pipeline solution</td>
<td>Human input about:</td>
<td>Pipeline workflow configuration</td>
<td>REQUIRED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A set of stages</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A set of event triggers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Each stage entrance and exit control gate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Activities in each stage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orchestrate pipeline</td>
<td>Automate the CI/CD tasks;</td>
<td>Event triggers (such as code commit, test</td>
<td>Pipeline workflow execution results (such as control gate validation,</td>
<td></td>
<td>REQUIRED</td>
</tr>
<tr>
<td>workflow execution by</td>
<td>Auditable trail of activities</td>
<td>results, human input, etc.); Artifacts from the</td>
<td>stage transition, activity execution, etc.); Event and activity audit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>coordinating other plugin tools or scripts.</td>
<td></td>
<td>artifact repository</td>
<td>logs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1 Containerized Software Factory

Software factory tools include a CI/CD orchestrator, a set of development tools, and a group of tools that operate in different DevSecOps lifecycle phases. These tools are pluggable and must integrate into the CI/CD orchestrator. In this reference design, instantiations must rely a containerized software factory instantiated from a set of DevSecOps hardened containers accessed directly from Iron Bank. Iron Bank containers are preconfigured and secured to reduce the certification and accreditation burden and are often available as a predetermined pattern or pipeline that will need limited or no configuration.
Running a CI/CD pipeline is a complex activity. Containerization of the entire CI/CD stack ensures there is no drift possible between different K8s cluster environments (development, test, staging, production). It further ensures there is no drift between different K8s cluster environments spanning multiple classification levels. Containerization also streamlines the update/accreditation process associated with the introduction and adoption of new DevSecOps tooling.

*Figure 5,* illustrates a containerized software factory reference design. The software factory is built on an underlying container orchestration layer powered by K8s in a host environment. For clarity, the software factory produces DoD applications and application artifacts as a product. Applications typically use different sets of hardened containers from the Iron Bank than the ones used to create the software factory.

The software factory reference design captured in *Figure 5* illustrates how cybersecurity is weaved into the fabric of each factory pipeline. All of the tooling within the factory is based on hardened containers pulled from Iron Bank.

Moving from left to right, as code is checked into a branch triggering the CI/CD pipeline workflow and resulting automated build, SAST, DAST, and unit tests are executed, as the orchestrator coordinates different tools to perform various tasks defined by the pipeline. If the build is successful and a container image is defined, a container security scan is triggered. Some tests and security tasks may require human involvement or consent before being considered complete and passed. If all of these tests are successful, then the artifact is deployed into the test environment. If all of the entrance rules of the next stage are met, the orchestrator will transition the pipeline into that stage and perform the defined activities by coordinating the tools via plugins. When all stages are complete, a significant number of security activities have completed and the artifact is eligible for deployment into production. Deployment into production should be fully automated, but may be gated by a human actually pressing a button to trigger the deployment.
DoD programs may have already implemented a DevSecOps platform. Operating a custom DevSecOps platform is an expensive endeavor because software factories require the same level of continuous investment as a software application. There are financial benefits for programs to plan a migration to a containerized software factory, reaping the benefits of centrally managed and hardened containers that have been fully vetted. In situations where a containerized software factory is impractical, or the factory requires extensive policy customizations, the program should consult with DoD CIO and (if applicable) its own DevSecOps program office to explore options and collaborate to create, sustain, and deliver program-specific hardened containers to Iron Bank.

Platform One is the first DoD-wide approved DevSecOps Managed Service. For more information: https://p1.dso.mil

4.2 Hosting Environment

The reference design does not restrict the software factory hosting environment, which could be DoD-approved Cloud Service Providers, DoD data centers or even on-premises servers. The hosting environment provides compute, storage, and network resources in either physical or virtual form.

4.3 Container Orchestration

K8s software factory responsibilities include container orchestration, interacting with the underlying hosting environment resources (compute, storage, etc.), and coordination of clusters of nodes at scale in development, testing and pre-production in an efficient manner. As
described in the opening paragraphs of this section, this reference design mandates a container orchestration layer as illustrated in Figure 6.

![Figure 6: DevSecOps Platform Options](image)

It is the mission program’s responsibility (or that of a DoD Enterprise DevSecOps Managed Service, such as Platform One), to build and maintain the K8s *container orchestration layer* using COTS solutions. The container orchestration layer can be deployed on top of a DoD authorized Cloud environment, a DoD data center, or on bare metal servers. The container orchestration system components are subject to monitoring and security control under the DoD policy in that hosting environment, such as the DoD Cloud Computing Security Requirements Guide (SRG) and DISA’s Secure Cloud Computing Architecture (SCCA) for the Cloud environment.

A notional set of DevSecOps services, an abbreviated representation of the DevSecOps workflow, and various cybersecurity mechanisms are depicted in *Figure 7*. 
5 Additional Tools and Activities

The **DevSecOps Tools and Activities Guidebook**, part of the DevSecOps Fundamentals, establishes common DevSecOps tools and activities. The guidebook recognizes that specific reference designs may elevate a specific tool from PREFERRED to REQUIRED, as well as add additional tools and/or activities that specifically support the nuances of a given reference design. The following sections identify those tools and activities unique to this reference design across the *Deploy* and *Monitor* phases of the DevSecOps lifecycle.
### Table 3: Security Activities Summary and Cross-Reference

<table>
<thead>
<tr>
<th>Activities</th>
<th>Phase</th>
<th>Activities Table Reference</th>
<th>Tool Dependencies</th>
<th>Tool Table Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container or VM hardening</td>
<td>Develop</td>
<td>Table 4</td>
<td>Container security tool; Security compliance tool</td>
<td></td>
</tr>
<tr>
<td>Container policy enforcement</td>
<td>Test</td>
<td>Table 6</td>
<td>Container policy enforcement</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4: Develop Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container image selection</td>
<td>Must leverage approved and hardened container images strictly from the Iron Bank repository</td>
<td>N/A</td>
<td>N/A</td>
<td>Artifact repo</td>
</tr>
<tr>
<td>Container hardening</td>
<td>Harden the deliverable for production deployment. Containers must follow the DISA Container Hardening Guide.¹⁰</td>
<td>Container</td>
<td>-Vulnerability report and recommended mitigation -Hardened Container &amp; Build File</td>
<td>Container security tool</td>
</tr>
</tbody>
</table>

### Table 5: Build Phase Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container builder</td>
<td>Build a container image based on a build instruction file. Must use a hardened container image from Iron Bank as the base image in all cases.</td>
<td>Container image build automation</td>
<td>Container base image; Container build file</td>
<td>OCI compliant container image</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Artifact Repository</td>
<td>Container Registry</td>
<td>Better quality software by using centrally managed, hardened containers.</td>
<td>Artifacts</td>
<td>Version controlled container</td>
<td>REQUIRED</td>
</tr>
</tbody>
</table>
### Table 6: Build Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containerize</td>
<td>Packages all required OS components, developed code, runtime libraries, etc. into a hardened container</td>
<td>Container base image; Container build file</td>
<td>Container Image</td>
<td>Container Builder</td>
</tr>
<tr>
<td>Store artifacts</td>
<td>Store artifacts to the artifact repository</td>
<td>Container Image</td>
<td>Version controlled container image</td>
<td>Artifact Repository</td>
</tr>
</tbody>
</table>

### Table 7: Test Phase Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TWO DIFFERENT</strong> Container security tool</td>
<td>Container image scan OS check. Two are required because scan results are too disparate.</td>
<td>Ease the container hardening process</td>
<td>Container images or running containers</td>
<td>Vulnerability report and recommended mitigation.</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Container policy enforcement</td>
<td>Support for Security Content Automation Protocol (SCAP) and Container configuration policies. These policies can be defined as needed.</td>
<td>Automated policy enforcement</td>
<td>Policies in SCAP form.</td>
<td>Compliance report</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Security compliance tool</td>
<td>Scan and report for compliance regulations, such as DISA Security Technical Implementation Guides (STIGs), NIST 800-53.</td>
<td>Speed up ATO process.</td>
<td>Container images.</td>
<td>Vulnerability report and recommended mitigation.</td>
<td>PREFERRE D</td>
</tr>
</tbody>
</table>
### Table 8: Test Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container policy enforcement</td>
<td>Check developed containers to be sure they meet container policies</td>
<td>Container, Policies in SCAP form</td>
<td>Container compliance report</td>
<td>Container policy enforcement</td>
</tr>
</tbody>
</table>

### Table 9: Release and Deliver Phase Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>IaC / CaC</td>
<td>Automated “push button” instantiation of the applications running on K8s in addition to the software factory itself (including the SCSS stack on top)</td>
<td>Eliminate drift between environments; ensure desired state is always accurately captured in git.</td>
<td></td>
<td></td>
<td>REQUIRED</td>
</tr>
<tr>
<td>GitOps Kubernetes Capability</td>
<td>Pull source code from git repositories instead of requiring the CI/CD pipeline to push artifacts to the next environment</td>
<td>Eliminates the need to open ports and/or require keys to be shared with CI/CD tooling. Eliminates environment drifts. Ensures desired state is always accurately captured in git.</td>
<td></td>
<td></td>
<td>RECOMMENDED</td>
</tr>
</tbody>
</table>

### Table 10: Release and Deliver Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release go / no-go decision</td>
<td>This is part of configuration audit; Decision on whether to release artifacts to the artifact repository for the production environment.</td>
<td>Design documentation; Version controlled artifacts; Version controlled test reports; Security test and scan reports</td>
<td>go / no-go decision; Artifacts are tagged with release tag if go decision is made</td>
<td>CI/CD Orchestrator</td>
</tr>
<tr>
<td>Tool</td>
<td>Features</td>
<td>Benefits</td>
<td>Inputs</td>
<td>Outputs</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CNCF-certified Kubernetes</td>
<td>Container grouping using pods; Health checks and self-healing Horizontal infrastructure scaling Container auto-scalability Domain Name Service (DNS) management Load balancing Rolling update or rollback; Resource monitoring and logging</td>
<td>Simplify operations by deployment and update automation Scale resources and applications in real time Cost savings by optimizing infrastructure resources</td>
<td>Container instance specification and monitoring policy</td>
<td>Running container</td>
</tr>
<tr>
<td>Service mesh</td>
<td>Ability to create a network of deployed services with load balancing, service-to-service authentication, and monitoring. Ability to enforce Zero trust mTLS traffic for east/west traffic</td>
<td>Support for microservice interactions.</td>
<td>Control plane: service communication routing policies, authenticaton certificates. Data plane: service communication data</td>
<td>Control plane: service status reports Data plane: routed service communication data</td>
</tr>
</tbody>
</table>
### Table 12: Deploy Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliver container to container registry</td>
<td>Upload the hardened container and associated artifacts to the container registry</td>
<td>Hardened container</td>
<td>New container instance</td>
<td>CNCF-certified Kubernetes; Artifact repository container registry</td>
</tr>
</tbody>
</table>

### Table 13: Operate Phase Activities

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Tool Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>Scale manages containers as a group. The number of containers in the group can be dynamically changed based on the demand and policy.</td>
<td>Real-time demand and container performance measures Scale policy (demand or Key Performance Indicator (KPI)threshold; minimum, desired, and maximum number of containers)</td>
<td>Optimized resource allocation</td>
<td>Container management on the hosting environment</td>
</tr>
<tr>
<td>Load balancing</td>
<td>Load balancing equalizes the resource utilization</td>
<td>Load balance policy Real time traffic load and container performance measures</td>
<td>Balanced resource utilization</td>
<td>Container management on the hosting environment</td>
</tr>
</tbody>
</table>
### Table 14: Monitor Phase Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource, Service, Container policy enforcement</td>
<td>Support for Security Content Automation Protocol (SCAP) and container configuration policies. These policies can be defined as needed.</td>
<td>Automated policy enforcement</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Vulnerability Management</td>
<td>Provides vulnerability management</td>
<td>Makes sure everything is properly patched to avoid known vulnerabilities</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>CVE Service / Host Based Security</td>
<td>Provides CVEs. Used by the vulnerability management agent in the security sidecar container.</td>
<td>Makes sure the system is aware of known vulnerabilities in components.</td>
<td>REQUIRED</td>
</tr>
</tbody>
</table>

### Table 15: CSP Managed Service Monitoring Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
<th>Benefits</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflow Analysis</td>
<td>Logs network traffic within an enclave</td>
<td>Helps to find anomalous patterns across environment and Platform</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Centralized Logging</td>
<td>Stores logs from the entire environment. Used by the SIEM/SOAR for log analysis and incident detection</td>
<td>Place to store logs across environment and Platform</td>
<td>REQUIRED</td>
</tr>
<tr>
<td>Centralized Analysis</td>
<td>SIEM/SOAR for log analysis and incident detection Tier 3 CSSP tools</td>
<td>Helps to find anomalous patterns across environment and Platform</td>
<td>RECOMMENDED</td>
</tr>
</tbody>
</table>
5.1 Additional Deployment Types

Continuous Deployment is triggered by the successful delivery of released artifacts to the artifact repository, and deployment may be subject to control with human intervention according to the nature of the application.

The following four deployment activities are intrinsically supported by K8s, so there are no new tool requirements beyond the use of Kubernetes captured earlier in this document.

5.1.1 Blue/Green Deployments

K8s offers exceptional support for what is known as Blue/Green Deployments. This style of deployment creates two identical environments, one that retains the current production container instances and the other that holds the newly deployed container instances. Both environments are fronted by either a router or a load balancer that can be configured to direct traffic to a specific environment based on a set of metadata rules. Initially, only the blue environment is getting production traffic. The green version can run through a series of post-deployment tests, some automated and some human driven. Once the new version is deemed to be stable and its functionality is working properly, the router or load balancer is flipped, sending all production traffic to the green environment. If an unanticipated issue occurs in the green environment, traffic can be instantaneously routed back to the stable blue environment. Once there is a high degree of confidence in the green environment, the blue environment can be automatically torn down, reclaiming those compute resources.

5.1.2 Canary Deployments

K8s also offers exceptional support for Canary deployments. This style of deployment pushes a new feature of capability into production and only makes it accessible to a small group of people for testing and evaluation. In some cases, these small groups may be actual users, or they may be developers. Typically, the percentage of users given access to the feature or capability will increase overtime. The goal is to verify that the application is working correctly with the new feature or capability installed in the production environment. The route to the feature is most often controlled through a route that is configured in such a way that only a small percent of the incoming traffic is forwarded to the newer (canary) version of the containerized application, perhaps based on a user's attributes.

5.1.3 Rolling Deployments

A rolling deployment occurs when a cluster slowly replaces its currently running instances of an application with newer ones. If the declarative configuration of the application calls for \( n \) instances of the application deployed across the K8s cluster, then at any point in time the cluster actually has \((n + 1)\) instances running. Once the new instance has been instantiated and verified through its built-in health checks, the old instance is removed from the cluster and its compute resources recycled.

The major benefit of this approach is the incremental roll-out and gradual verification of the application with increasing traffic. It also requires less compute resources than a Blue/Green deployment, requiring only one additional instance instead of an entire duplication of the cluster. A disadvantage of this approach is that the team may struggle with an \((n-1)\) compatibility
problem, a major consequence for all continuous deployment approaches. Lost transactions and logged-off users are also something to take into consideration while performing this approach.

5.1.4 Continuous Deployments

This style of deployment is tightly integrated with an array of DevSecOps tools, including the artifact repository for retrieving new releases, the log storage and retrieval service for logging of deployment events, and the issue tracking system for recording any deployment issues. The first-time deployment may involve infrastructure provisioning using infrastructure as code (IaC), dependency system configuration (such as monitoring tools, logging tools, scanning tools, backup tools, etc.), and external system connectivity such as DoD common security services. Continuous deployment differs from continuous delivery. In continuous delivery, the artifact is deemed production ready and pushed into the artifact repository where it could be deployed into production at a later point in time. Continuous deployment monitors these events and automatically begins a deployment process into production. Continuous deployment often works well with a rolling deployment strategy.

5.2 Continuous Monitoring in K8s

Continuous monitoring of a K8s cluster must include behavior and signature-based detection in the runtime environment. These and other container specific controls are captured in NIST Special Publication 800-190, Application Container Security Guide. CSP services also routinely monitor and scan CSP resources and services for misconfiguration, incorrect access control, and security events. These CSP specific capabilities should be integrated into every continuous monitoring strategy.

Figure 8 illustrates a notional process of monitoring, logging, and log analysis and alerting. The process starts with application logging, compute resource monitoring, storage monitoring, network monitoring, security monitoring, and data monitoring at the Kubernetes pod level (the individual subsystem level in the case of VM deployment and the service level for serverless deployments).

Each application will need to determine how it is divided into subsystems, the number of subsystems, and the specific monitoring mechanisms within the subsystems. The security tools within each subsystem will aggregate and forward the event logs gathered from monitoring to a locally centralized aggregated logs database on the mission program platform. The aggregated logs will be further forwarded to the Logs/Telemetry Analysis in the Defensive Cyber Operations / Tier 2 CSSP after passing the program application configured log filter. The program’s local log SIEM/SOAR analysis capability will analyze the aggregated logs and generate incident alerts and reports.
Incidents will be forwarded to the mission program incident management system to facilitate change request generation for incident resolution. The mission program incident management should alert or notify the responsible personnel about the incidents. The change request may be created to address the incident. These actions make the DevSecOps pipeline a full closed loop from secure operations to planning.

5.2.1 CSP Managed Services for Continuous Monitoring

The use of CSP managed services for monitoring alongside 3rd party security tools should always be viewed through a “both/and” lens instead of an “either/or” lens. CSP managed services can be utilized to monitor CSP resources & services, netflow, and entity behavior analysis at a deeper level than with 3rd party tools alone. It may also be possible to employ CSP managed services to perform log analysis (SIEM/SOAR). The monitoring ecosystem should rely on curated IaC to instantiate the monitored environment to the maximum extent possible, ensuring completeness and accelerating the A&A process.