DoD Enterprise DevSecOps Reference Design

Version 1.0
12 August 2019

Department of Defense (DoD)
Chief Information Officer

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

Document Approvals

Prepared By:

LAM.NGOAN.THOM
AS.1229438960

Digitally signed by
LAM.NGOAN.THOMAS.1229438960
Date: 2019.09.05 11:52:32 -04'00'

Thomas Lam

Acting Director of Architecture and Engineering

Department of Defense, Office of the Chief Information Officer (DoD CIO)

CHAILLAN.NICOLAS.
MAXIME.1535056524

Digitally signed by
CHAILLAN.NICOLAS.MAXIME.153505652
4
Date: 2019.09.05 12:01:37 -04'00'

Nicolas Chaillan

Special Advisor for Cloud Security and DevSecOps

Department of Defense, Office the Undersecretary of Acquisition and Sustainment (A&S)

(currently: Chief Software Officer, Department of Defense, United States Air Force, SAF/AQ)

Approved By:

RANKS.PETER.TH
OMAS.12846166
65

Digitally signed by
RANKS.PETER.THOMAS.1284
616665
Date: 2019.09.05 21:41:37
-04'00'

Peter Ranks

Deputy Chief Information Officer for Information Enterprise (DCIO IE)

Department of Defense, Office of the Chief Information Officer (DoD CIO)

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.

Executive Summary

Legacy software acquisition and development practices in the DoD do not provide the agility to deploy new software “at the speed of operations”. In addition, security is often an afterthought, not built in from the beginning of the lifecycle of the application and underlying infrastructure. DevSecOps is the industry best practice for rapid, secure software development.

DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. In DevSecOps, testing and security are shifted to the left through automated unit, functional, integration, and security testing - this is a key DevSecOps differentiator since security and functional capabilities are tested and built simultaneously.

The benefits of adopting DevSecOps include:

- Reduced mean-time to production: the average time it takes from when new software features are required until they are running in production;
- Increased deployment frequency: how often a new release can be deployed into the production environment;
- Fully automated risk characterization, monitoring, and mitigation across the application lifecycle;
- Software updates and patching at "the speed of operations".

This DoD Enterprise DevSecOps Reference Design describes the DevSecOps lifecycle, supporting pillars, and DevSecOps ecosystem; lists the tools and activities for DevSecOps software factory and ecosystem; introduces the DoD enterprise DevSecOps container service that provides hardened DevSecOps tools and deployment templates to the program application DevSecOps teams to select; and showcases a sampling of software factory reference designs and application security operations. This DoD Enterprise DevSecOps Reference Design provides implementation and operational guidance to Information Technology (IT) capability providers, IT capability consumers, application teams, and Authorizing Officials.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). The main characteristic of DevSecOps is to improve customer outcomes and mission value by automating, monitoring, and applying security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. Practicing DevSecOps provides demonstrable quality and security improvements over the traditional software lifecycle, which can be measured with these metrics:

- Mean-time to production: the average time it takes from when new software features are required until they are running in production.
- Average lead-time: how long it takes for a new requirement to be delivered and deployed.
- Deployment speed: how fast a new version of the application can be deployed into the production environment.
- Deployment frequency: how often a new release can be deployed into the production environment.
- Production failure rate: how often software fails during production.
- Mean-time to recovery: how long it takes applications in the production stage to recover from failure.

In addition, DevSecOps practice enables:

- Fully automated risk characterization, monitoring, and mitigation across the application lifecycle.
- Software updates and patching at a pace that allows the addressing of security vulnerabilities and code weaknesses.

DevSecOps practice enables application security, secure deployment, and secure operations in close alignment with mission objectives. In DevSecOps, testing and security are shifted to the left through automated unit, functional, integration, and security testing - this is a key DevSecOps differentiator since security and functional capabilities are tested and built simultaneously. In addition, some security features are automatically injected into the application without developer intervention via a sidecar container.

1.2 Purpose

The main purpose of this document is to provide a logical 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.

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 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)

The DoD Enterprise DevSecOps reference design leverages a set of hardened DevSecOps tools and deployment templates that enable DevSecOps teams to select the appropriate template for the program application capability to be developed. For example, these templates will be specialized around a specific programming language or around different types of capabilities such as web application, transactional, big data, or artificial intelligence (AI) capabilities. A program selects a DevSecOps template and toolset; the program then uses these to instantiate a DevSecOps software factory and the associated pipelines that enable Continuous Integration and Continuous Delivery (CI/CD) of the mission application.

This reference design aligns with these reference documents:

- DoD Cloud Computing Strategy [1]
- DoD Cloud Computing Security Requirements Guide [2]
- DoD Secure Cloud Computing Architecture (SCCA) [3]
- Presidential Executive Order on Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure (Executive Order (EO) 1380) [4]
- National Institute of Standards and Technology (NIST) Cybersecurity Framework [5]
- DoD Container Hardening Security Requirements Guide [6].

1.3 Scope

This document describes the reference design to enable DevSecOps to scale across the department. DevSecOps is an established mature capability in industry, and it is already used within some pockets of the Government; this reference design formalizes its usage across the

DoD. This 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, C2, embedded systems, big data, and Artificial Intelligence (AI).

This document does not address policy or acquisition.

1.4 Document Overview

The documentation is organized as follows.

- Section 1 describes the background, purpose and scope of this document.
- Section 2 describes the assumptions made in developing this reference design, as well as stating foundational principles.
- Section 3 describes the DevSecOps lifecycle, the four pillars to assist DevSecOps adoption, and a technical architecture of the DevSecOps software factory and its ecosystem.
- Section 4 describes the DevSecOps ecosystem tools and the activities along software lifecycle phases.
- Section 5 describes the DoD Enterprise DevSecOps Service. The target audience of this section is DoD Enterprise DevSecOps capability providers.
- Section 6 describes the reference design for DoD programs to build their DevSecOps software factory and ecosystem.

2 Assumptions and Principles

2.1 Assumptions

This document makes the following assumptions:

- For most organizations, deploying to a certified and monitored cloud environment will become their preferred solution technically and culturally.
- Rapidly changing technology dictates designing the DevSecOps pipelines and patterns for flexibility as new development capabilities enter/exit the commercial product market.
- The DoD Enterprise DevSecOps software factory is designed to avoid vendor lock-in and leverage Open Container Initiative (OCI) compliant containers and Cloud Native Computing Foundation (CNCF) certified Kubernetes to orchestrate and manage the containers.
- The government must balance open source integration risks vs. using pre-integrated Commercial Off-The-Shelf (COTS) products that have vendor “cost of exit” and vendor insider risks.
- It must be possible to host a DevSecOps software factory in any DoD general-purpose cloud environment, as well as in disconnected and classified environments.
- The DevSecOps architecture must have the capability to scale to any type of operational requirement needing a software solution, including:
    - Business systems
    - Command and Control systems
    - Embedded and Weapon systems
    - Intelligence analysis systems
    - Autonomous systems
    - Assisted human operations

2.2 Principles

There are several key principles to implementing a successful DevSecOps approach:

- Remove bottlenecks (including human ones) and manual actions.
- Automate as much of the development and deployment activities as possible.
- Adopt common tools from planning and requirements through deployment and operations.

- Leverage agile software principles and favor small, incremental, frequent updates over larger, more sporadic releases.
- Apply the cross-functional skill sets of Development, Cybersecurity, and Operations throughout the software lifecycle, embracing a continuous monitoring approach in parallel instead of waiting to apply each skill set sequentially.
- Security risks of the underlying infrastructure must be measured and quantified, so that the total risks and impacts to software applications are understood.
- Deploy immutable infrastructure, such as containers. The concept of immutable infrastructure is an IT strategy in which deployed components are replaced in their entirety, rather than being updated in place. Deploying immutable infrastructure requires standardization and emulation of common infrastructure components to achieve consistent and predictable results.

3 DevSecOps Concepts

DevSecOps describes an organization’s culture and practices enabling organizations to bridge the gap between developers, security team, and operations team; improve processes through collaborative and agile workflows; drive for faster and more secure software delivery via technology; and achieve consistent governance and control. There is no uniform DevSecOps practice. Each DoD organization needs to tailor its culture and its DevSecOps practices to its own unique processes, products, security requirements, and operational procedures. Embracing DevSecOps requires organizations to shift their culture, evolve existing processes, adopt new technologies, and strengthen governance.

This section will briefly discuss the DevSecOps lifecycle, supporting pillars, and the DevSecOps ecosystem.

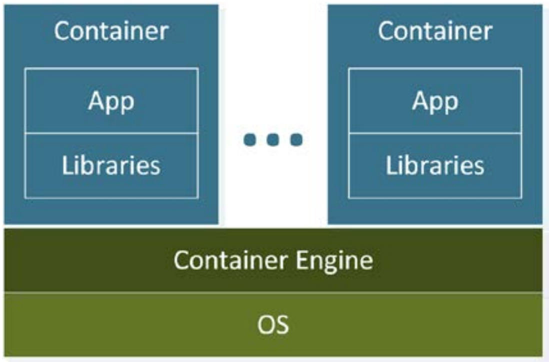
3.1 Key Terms

Here are some key terms used throughout the document. Refer to the glossary in Appendix B for the full list.

Table 1: Key Terms

Term	Definition
DevSecOps Ecosystem	A collection of tools and process workflows created and executed on the tools to support all the activities throughout the full DevSecOps lifecycle. The process workflows may be fully automated, semi-automated, or manual.
Software Factory	A software assembly plant that contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The software factory supports multi-tenancy.
CI/CD Pipeline	The set of tools and the associated process workflows to achieve continuous integration and continuous delivery with build, test, security, and release delivery activities, which are steered by a CI/CD orchestrator and automated as much as practice allows.

Term	Definition
CI/CD Pipeline Instance	A single process workflow and the tools to execute the workflow for a specific software language and application type for a project. As much of the pipeline process is automated as is practicable.
Environment	Sets a runtime boundary for the software component to be deployed and executed. Typical environments include development, integration, test, pre-production, and production.
Software Factory Artifact Repository	A local repository tied to the software factory. It stores artifacts pulled from DoD Centralized Artifact Repository (DCAR) as well as locally developed artifacts to be used in DevSecOps processes. The artifacts include, but are not limited to, virtual machine (VM) images, container images, binary executables, archives, and documentation. It supports multi-tenancy. Note that programs may have a single artifact repository and use tags to distinguish the content types. It is also possible to have separate artifact repositories to store local artifacts and released artifacts.
Code	Software instructions for a computer, written in a programming language. These instructions may be in the form of either human-readable source code, or machine code, which is source code that has been compiled into machine executable instructions.

Term	Definition
Containers	A standard unit of software that packages up code and all its dependencies, down to, but not including the Operating System (OS). It is a lightweight, standalone, executable package of software that includes everything needed to run an application except the OS: code, runtime, system tools, system libraries and settings. Several containers can run in the same OS without conflicting with one another.  Container App Libraries Container App Libraries Container Engine OS **Figure 1: Containers** Containers run on the OS, so no hypervisor (virtualization) is necessary (though the OS itself may be running on a hypervisor). Containers are much smaller than a VM, typically by a factor of 1,000 (MB vs GB), partly because they don't need to include the OS. Using containers allows denser packing of applications than VMs. Unlike VMs, containers are portable between clouds or between clouds and on-premise servers. This helps alleviate Cloud Service Provider (CSP) lock-in, though an application may still be locked-in to a CSP, if it uses CSP-specific services. Containers also start much faster than a VM (seconds vs. minutes), partly because the OS doesn't need to boot.

3.1.1 Conceptual Model

The following conceptual model shows some of the most important concepts described in this paper along with their relationships. It should help to clarify these relationships. When reading text along an arrow, follow the direction of the arrow. So, a DevSecOps Ecosystem contains one or more software factories, and each software factory contains one or more pipelines. The diagram also shows that each software factory contains only one CI/CD Orchestrator, and that many software factories use DCAR.

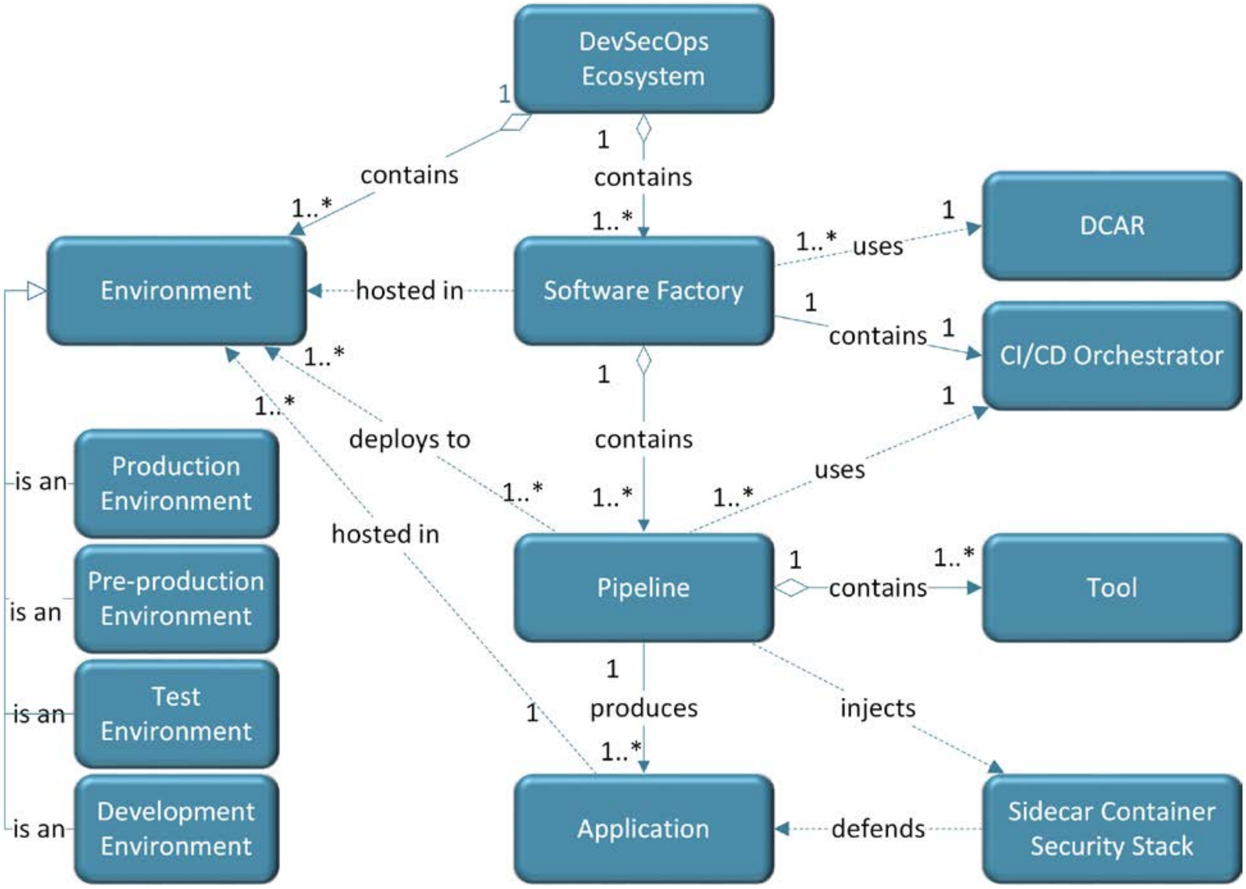


Figure 2: Conceptual Model

3.2 DevSecOps Lifecycle

The DevSecOps software lifecycle phases are illustrated in Figure 3. There are nine phases: plan, develop, build, test, release, deliver, deploy, operate, and monitor. Security is embedded within each phase.

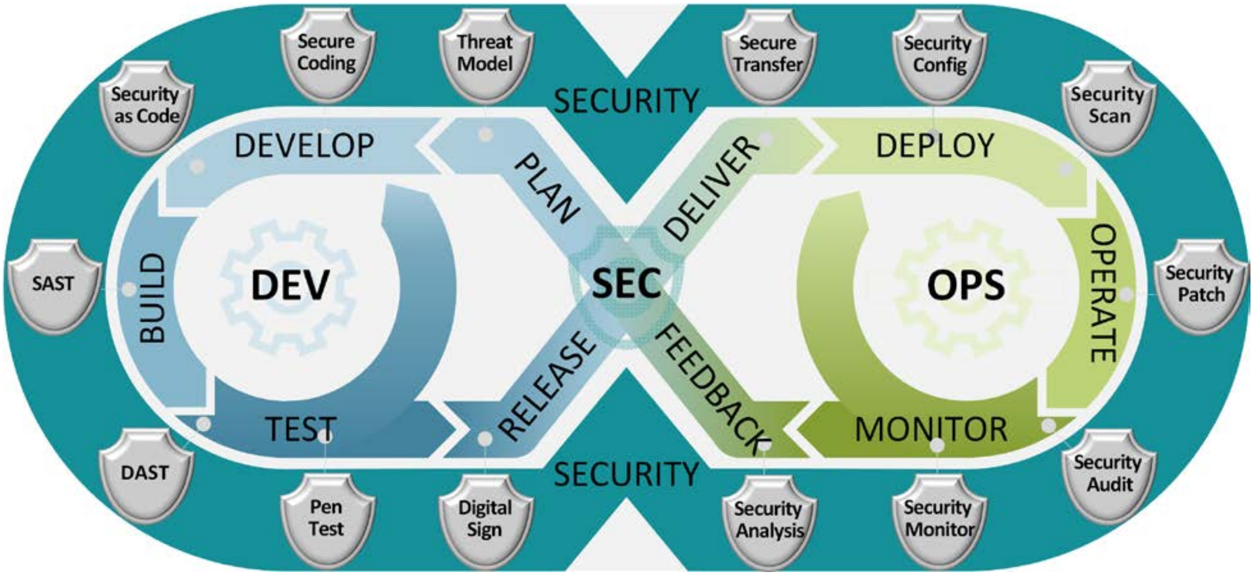


Figure 3: DevSecOps Software Lifecycle

With DevSecOps, the software development lifecycle is not a monolithic linear process. The “big bang” style delivery of the Waterfall process is replaced with small but more frequent deliveries, so that it is easier to change course as necessary. Each small delivery is accomplished through a fully automated process or semi-automated process with minimal human intervention to accelerate continuous integration and delivery. The DevSecOps lifecycle is adaptable and has many feedback loops for continuous improvement.

3.3 DevSecOps Pillars

DevSecOps are supported by four pillars: organization, process, technology, and governance, as illustrated in Figure 4.

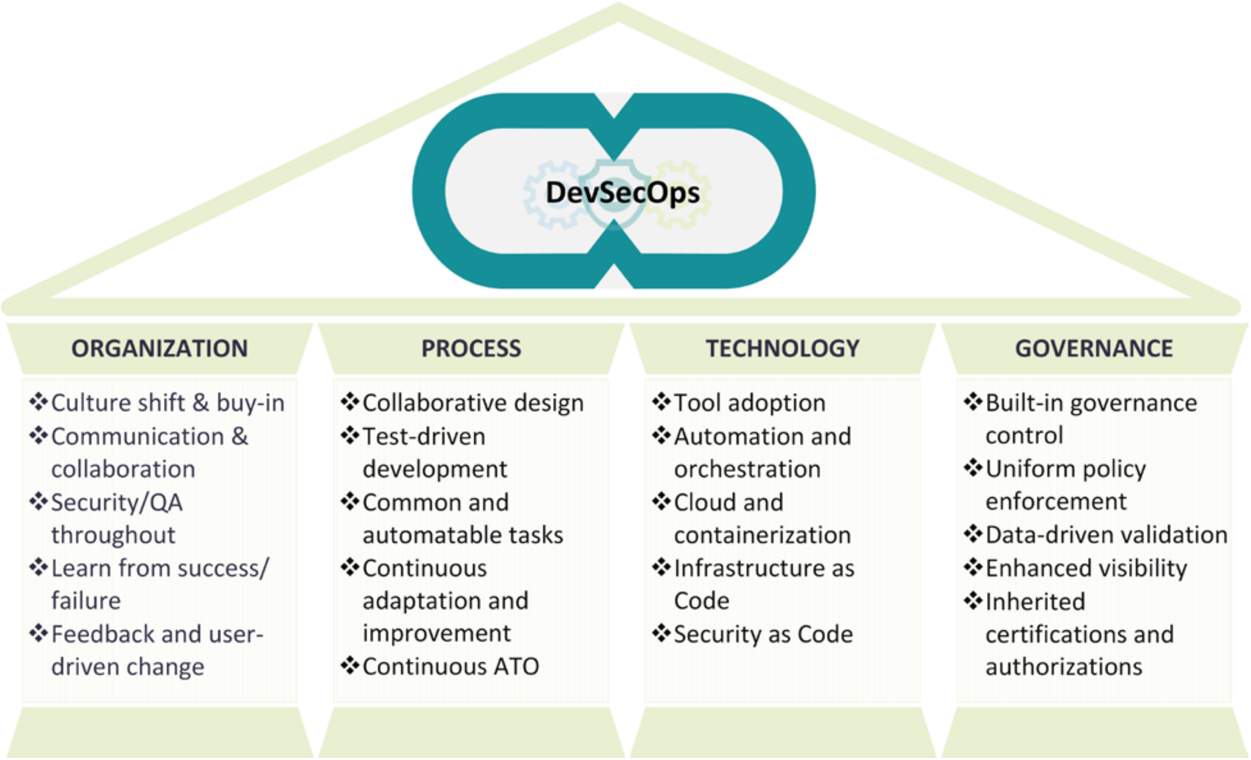

DevSecOps
ORGANIZATION
❖Culture shift & buy-in
❖Communication & collaboration
❖Security/QA throughout
❖Learn from success/ failure
❖Feedback and user-driven change
PROCESS
❖Collaborative design
❖Test-driven development
❖Common and automatable tasks
❖Continuous adaptation and improvement
❖Continuous ATO
TECHNOLOGY
❖Tool adoption
❖Automation and orchestration
❖Cloud and containerization
❖Infrastructure as Code
❖Security as Code
GOVERNANCE
❖Built-in governance control
❖Uniform policy enforcement
❖Data-driven validation
❖Enhanced visibility
❖Inherited certifications and authorizations

Figure 4: DevSecOps Pillars

For each DoD organization, the practice of DevSecOps starts with buy-in of the DevSecOps philosophy by senior leaders within the organization. This leads to a change to the organizational culture, along with the development of new collaborative processes, technologies and tools to automate the process and to apply consistent governance. A project must advance in all four areas to be successful.

3.3.1 Organization

The organization should embrace the following philosophies and ideas and incorporate them into their daily activities and software lifecycle management processes.

- Change the organizational culture to take a holistic view and share the responsibility of software development, security and operations. Train staff with DevSecOps concepts and new technologies. Gradually gain buy-in from all stakeholders.
- Break down organizational silos. Increase the team communication and collaboration in all phases of the software lifecycle.
- Actionable security and quality assurance (QA) information, such as security alerts or QA reports, must be automatically available to the teams at each software lifecycle phase to make collaborative actions possible.

- Build a culture of safety by sharing after-action reports on both positive and negative events across the entire organization. Teams should use both success and failure as learning opportunities to improve the system design, harden the implementation, and enhance the incident response capability as part of the DevSecOps practice.
- Make many small, incremental changes instead of fewer large changes. The scope of smaller changes is more limited and thus easier to manage.
- Embrace feedback and user driven change to respond to new, emerging, and unforeseen requirements.
- Plan and budget for continuous code refactoring to ensure constant buy down of accumulated technical debt.

3.3.2 Process

Depending on the mission environment, system complexity, system architecture, software design choices, risk tolerance level, and system maturity level, each program’s software lifecycle has its own unique management processes.

For example, suppose a mature web application software system has adopted a microservices design. Its development, pre-production, and production environment are on the same cloud. The test procedures are fully automated. This system could have a process flow to automate the develop, build, test, secure, and delivery tasks to push updates into production quickly without human intervention. On the other hand, a complex mission critical embedded system, such as a weapons system, may have a different process that requires some tests that cannot be fully automated. The software lifecycle process for that system will be significantly different from the process for the web application system.

To adopt a DevSecOps process successfully, implement it in multiple, iterative phases. Start small with some tasks that are easy to automate, then gradually build up the DevSecOps capability and adjust the processes to match. Figure 5 illustrates this concept; it shows that a software system can start with a *Continuous Build* pipeline, which only automates the build process after the developer commits code. Over time, it can then progress to *Continuous Integration*, *Continuous Delivery*, *Continuous Deployment*, *Continuous Operation*, and finally *Continuous Monitoring*, to achieve the full closed loop of DevSecOps. A program could start with a suitable process and then grow progressively from there. The process improvement is frequent, and it responds to feedback to improve both the application and the process itself.

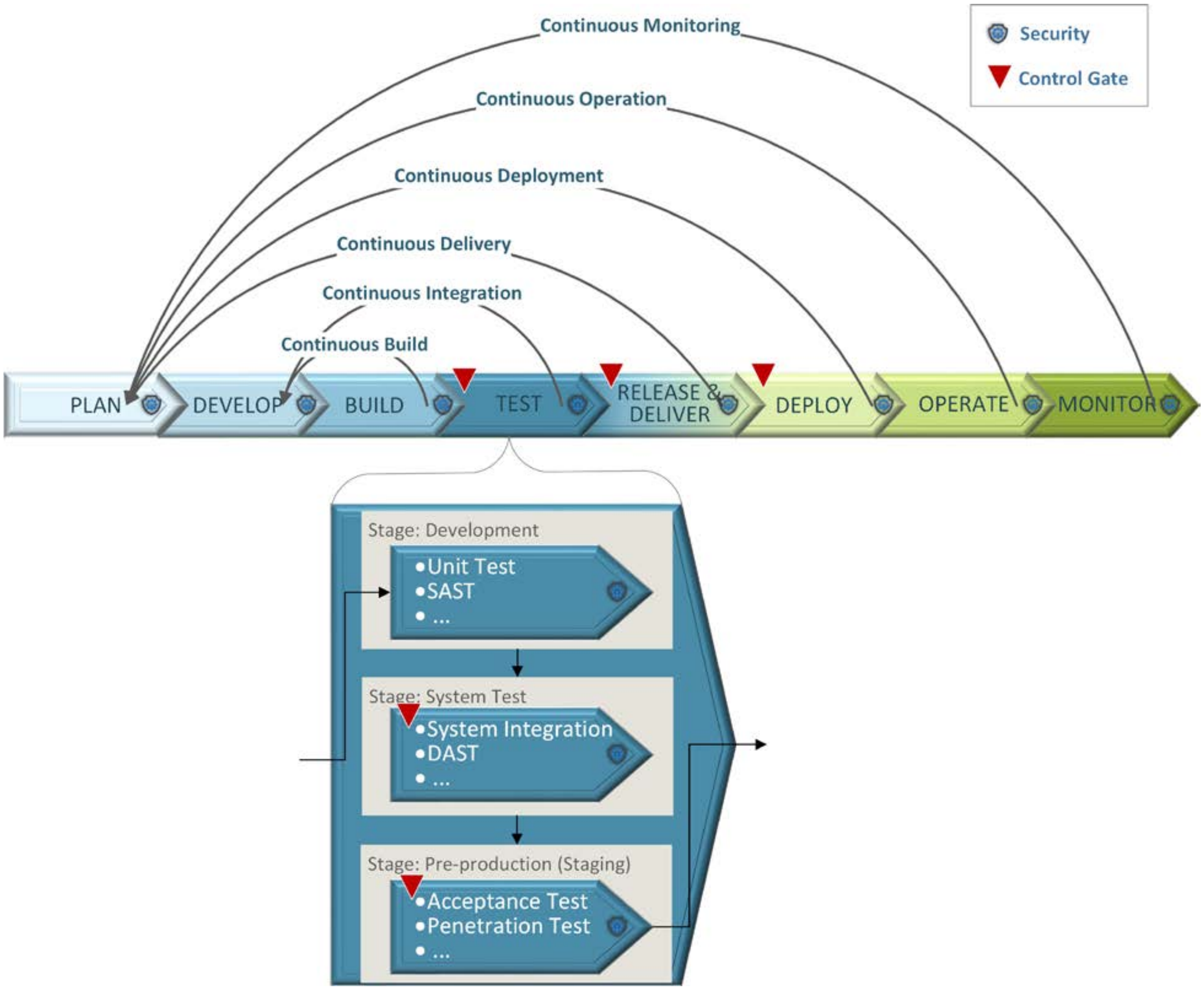


Figure 5: Application DevSecOps Processes

There is no "one size fits all" solution for process design. Each software team has its own unique requirements and constraints. Below is a list of some best practices to guide the process design:

1. The process design is a collective effort from multidisciplinary teams.
2. Most of the processes should be automatable via tools and technologies.
3. The DevSecOps lifecycle is an iterative closed loop. Start small and build it up progressively to strive for continuous improvement. Set up human intervention at the control gates when necessary, depending on the maturity level of the process and the team's confidence level in the automation. Start with more human intervention and gradually decrease it as possible.
4. AO should consider automating the Authority to Operate (ATO) process as much as possible.

To help organizations evolve their DevSecOps capabilities and processes, the DoD has developed a DevSecOps Maturity Model. This model details many steps that organizations can take to move incrementally towards a higher DevSecOps maturity level. That maturity model is presented in the DoD DevSecOps Playbook [7].

3.3.3 Technology

Many DevSecOps tools automate many tasks in the software lifecycle without human involvement. Other DevSecOps tools, such as collaboration and communication tools, facilitate and stimulate human interaction to improve productivity. Some DevSecOps tools aim to help an activity at a specific lifecycle phase. For example, an Integrated Development Environment (IDE) DevSecOps plug-in for develop phase, or a static application security test tool for the build phase. Most tools assist a particular set of activities. The tags added to artifacts in the artifact repository help guarantee that the same set of artifacts move together along a pipeline. Section 4 will introduce a variety of DevSecOps tools.

The instantiation of the DevSecOps environments can be orchestrated from configuration files instead of setting up one component at a time manually. The infrastructure configuration files, the DevSecOps tool configuration scripts, and the application run-time configuration scripts are referred to as Infrastructure as Code (IaC). Taking the same approach as IaC, security teams program security policies directly into configuration code, as well as implement security compliance checking and auditing as code, which are referred as Security as Code (SaC). Both IaC and SaC are treated as software and go through the rigorous software development processes including design, development, version control, peer review, static analysis, and test.

Technologies and tools play a key role in DevSecOps practice to shorten the software lifecycle and increase efficiency. They not only enable software production automation as part of a software factory, but also allow operations and security process orchestration.

3.3.4 Governance

Governance actively assesses and manages the risks associated with the mission program throughout the lifecycle. Governance activities do not stop after ATO but continue throughout the software lifecycle, including operations and monitoring. DevSecOps can facilitate and automate many governance activities.

3.3.4.1 Management Structure

The management objective of DevSecOps must be both “top-down” and “bottom-up” to balance larger strategic goals. Studies (e.g., [8]) have shown that senior leader buy-in is crucial for success. But buy-in at the staff level is also important to engender a sense of ownership, to encourage the appropriate implementation of processes related to governance, and to enable team members to support continuous process improvement. Continuous process improvement – seeking opportunities to simplify and automate whenever and wherever possible – is essential for governance to keep pace with a rapidly changing world.

Early DevSecOps efforts in the DoD, such as [9] have leveraged and adopted commercial best practices. That document identifies Five Fundamental Principles of Next Generation Governance (NGG):

1. *Run IT with Mission Discipline*: Tie requirements back to your organization's mission. Every action should be aligned to the mission. If they are not, then an evaluation should be performed with Continuous Process Improvement to address how to tie actions to missions.
2. *Invest in Automation*: Automate everything possible, including actions, business processes, decisions, approvals, documentation, and more. Automation, including designs, interfaces, functional and security tests, and their related documentation, create the Artifacts of Record that provide the body of evidence required by the Risk Management Framework (RMF) and for historical audits when needed.
3. *Embrace Adaptability*: Accept that change can be required at any time, and all options are available to achieve it. Fail fast, fail small, and fail forward. An example of failing forward is when a developer finds that a release does not work. Then instead of restoring the server to its pre-deployment state with the previous software, the developer's change should be discrete enough that they can fix it and address the issue through a newer release.
4. *Promote Transparency*: Offer open access across the organization to view the activities occurring within the automated process and to view the auto-generated Artifacts of Record. Transparency generates an environment for sharing ideas and developing solutions comprised of Subject Matter Experts (SMEs) or leads from across the enterprise in the form of cross-functional teams to avoid the "silo effect." When composed of all representative stakeholders, the team possesses the skills needed to build a mission system and the collective ingenuity necessary to overcome all encountered challenges.
5. *Inherent Accountability*: Push down or delegate responsibility to the lowest level:
   - Strategic: This is related to the Change Control Board (CCB) or Technical Review Board (TRB); it involves "Big Change" unstructured decisions. These infrequent and high-risk decisions have the potential to shape the strategy and mission of an organization.
   - Operational: (Various Scrum) Cross-cutting, semi-structured decisions. In these frequent and high-risk decisions, a series of small, interconnected decisions are made by different groups as part of a collaborative, end-to-end decision process.
   - Tactical: (Global Enterprise Partners (GEP)/Product Owner/Developers Activities) Delegated, structured decisions. These frequent and low-risk decisions are effectively handled by an individual or working team, with limited input from others.

These 5 principles are summarized in Figure 6, which is from [9].

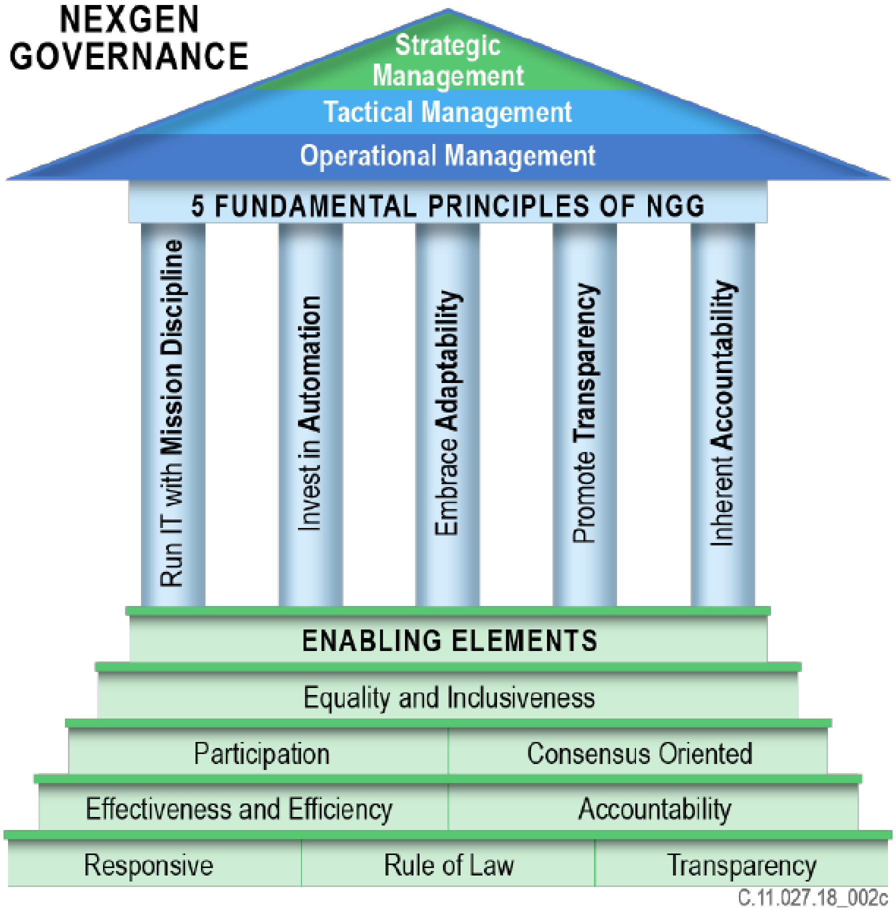


Figure 6: Five Principles of Next Generation Governance

3.3.4.2 Authorizing Official

DoD Instruction (DoDI) 8510.01 [10] is the ultimate governance policy and states processes that all DoD information system and platform information technology system must follow. It is under revision and the following DevSecOps related governance information will be incorporated in the future release.

For initial standup of a new DevSecOps software factory instance and a production operations environment, the RMF process follows an enterprise level process. The assessment and authorization (A&A) should inherit the certifications and authorizations of the underlying infrastructure (e.g., a DoD cloud provisional authorization) and of the DoD Enterprise Hardened Containers, without having to re-certify them. The program should have a formal Service Level Agreement (SLA) with the underlying infrastructure provider about what services are included and what authorizations can be inherited. This affects the status of applicable assessment procedures and prepares the stage for inheritance into the operations environment and application. Once the instance is authorized and operational, the specialty AO for the functional area or the local AO for the program has cognizance for Continuous Authorization of the environment. Figure 7 illustrates the A&A inheritance.

The specialty AO for the functional area or the local AO for the program has cognizance for Continuous Authorization of the mission applications.

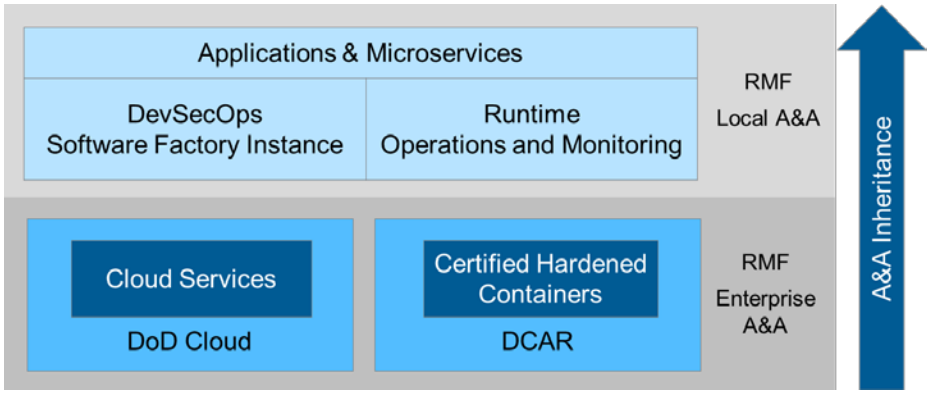

Applications & Microservices
DevSecOps Software Factory Instance
Runtime Operations and Monitoring
RMF Local A&A
Cloud Services
DoD Cloud
Certified Hardened Containers
DCAR
RMF Enterprise A&A
A&A Inheritance

Figure 7: Assessment and Authorization Inheritance

Table 2: Roles of Authorizing Officials in DevSecOps

Capability	Authorizing Official
DoD Enterprise Hardened Containers	Enterprise AO (e.g., Defense Information Systems Agency (DISA))
DevSecOps software factory instances	Enterprise AO (e.g., DISA, Military Department (MilDep) CIO)
Continuous Process Improvement / Continuous Authorization of DevSecOps software factory instances	Specialty or Local AO (e.g., Program executive Officer (PEO))
AO for mission applications	Specialty or Local AO (e.g., PEO)

3.4 DevSecOps Ecosystem

The DevSecOps ecosystem is a collection of tools and the process workflows created and executed on the tools to support all the activities throughout the full DevSecOps lifecycle. As illustrated in Figure 8, the DevSecOps ecosystem is composed of three subsystems: planning, a software factory, and production operations. The DevSecOps ecosystem interacts with external enterprise services to get all dependency support, and with enterprise and local AO to gain operation authorization.

The DevSecOps administration team is responsible for administrating the ecosystem tools and automating the process workflows. The mission application team focuses on the development, testing, security and operations tasks using the ecosystem.

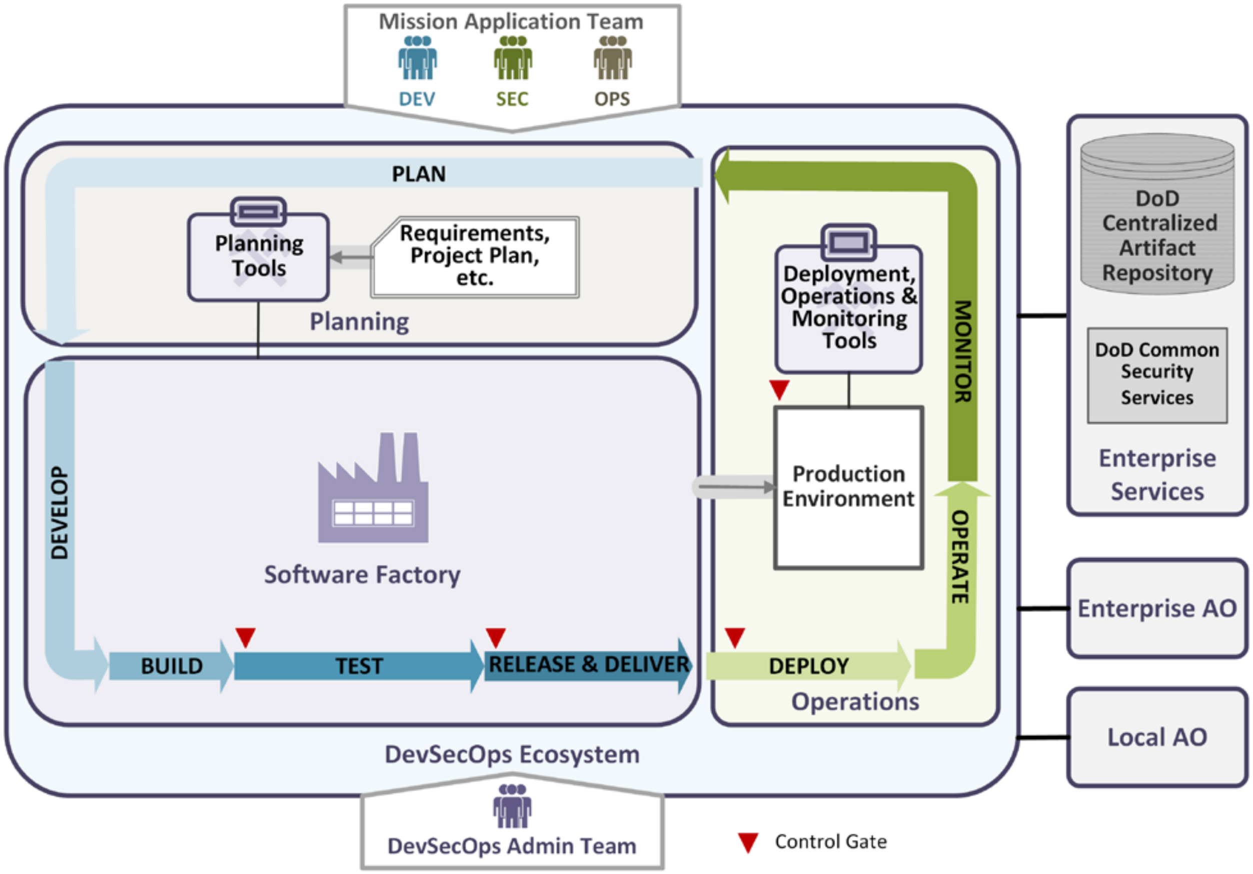


Figure 8: DevSecOps Ecosystem

3.4.1 Planning

The plan phase involves activities that help the project manage time, cost, quality, risk and issues. These activities include business-need assessment, project plan creation, feasibility analysis, risk analysis, business requirements gathering, business process creation, system design, DevSecOps design and ecosystem instantiation, etc. The plan phase repeats when DevSecOps the lifecycle recycles. It is a best practice to develop a minimum viable product (MVP) for critical business needs as the first thing to develop. Then get into the feedback loop process as quickly as possible; this is recommended in the Lean Startup methodology [11]. The DevSecOps design creates the DevSecOps processes and control gates, which will guide the automation throughout the lifecycle. DevSecOps ecosystem tools will facilitate process automation and consistent process execution.

The DevSecOps planning subsystem supports the activities in the plan phase using a set of communication, collaboration, project management, and change management tools. In this phase, the process workflows are not fully automated. The planning tools assist human interaction and increase team productivity.

3.4.2 Software Factory

A software factory, illustrated in Figure 9, contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The environments that are set up in the software factory should be orchestrated with scripts that include IaC and SaC and which run on various tools. A software factory must be designed for multi-tenancy and automate software production for multiple projects. A DoD organization may need multiple pipelines for different types of software systems, such as web applications or embedded systems.

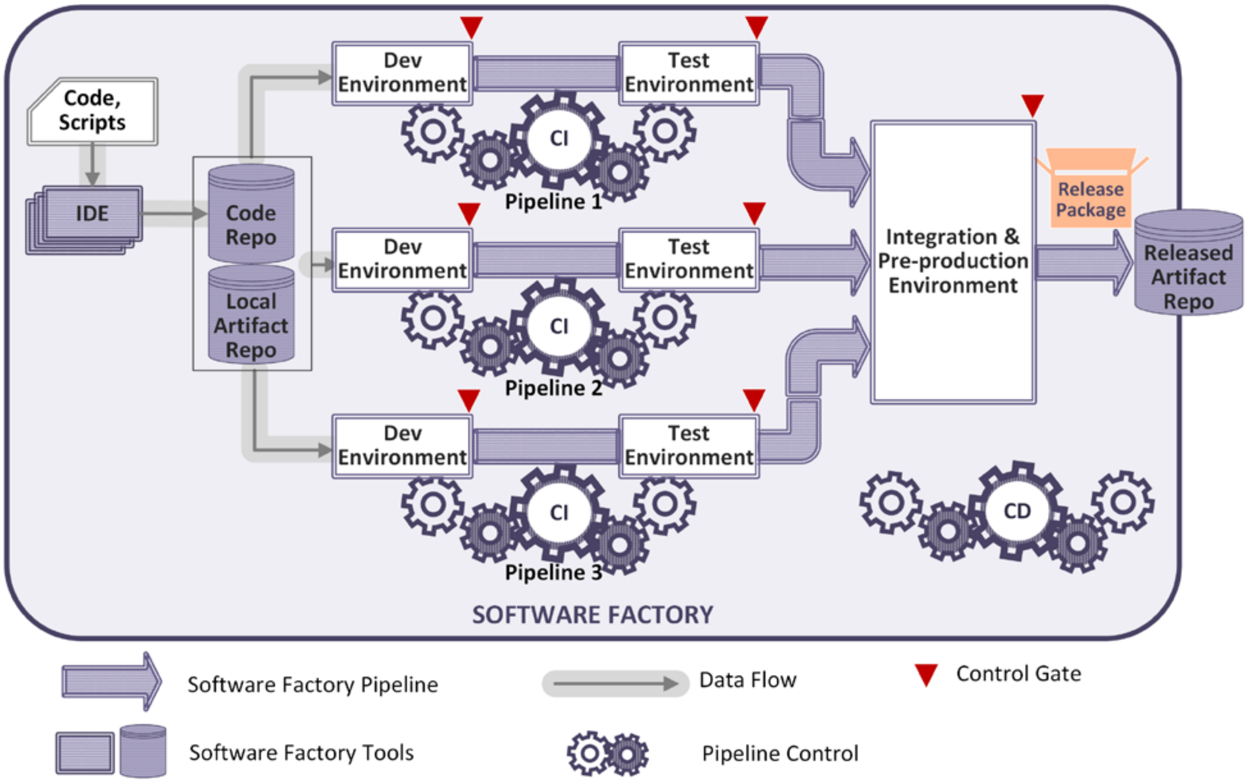


Figure 9: DevSecOps Software Factory

The factory starts with the development team developing application code and IaC, QA developing test scripts, and the security team developing SaC in their suitable IDEs. The entire Software Factory should leverage OCI compliant containers and DoD Hardened Containers whenever available. Once COTS, Government off the Shelf (GOTS), or newly developed code and scripts are committed into the Software Factory's code repository, the assembly line automation kicks in. There could be multiple CI pipeline instances as assembly lines. Each is for a specific application subsystem, such as a JavaScript assembly line for a web front-end, a Python or R assembly line for data analytics, or a GoLang assembly line for a backend application. The CI assembly line guides the subsystem through continuous integration by building the code and incorporating dependencies (such as libraries) from the local artifact repository. In addition, it performs tests in the development and test environments, such as unit tests, static code analysis, functional tests, interface tests, dynamic code analysis, etc. The subsystems that pass CI assembly line control gate policies will move into the pre-production environment for systems integration. The CD assembly line takes over control from this point. More tests and security scans are performed in this environment, such as performance tests, acceptance test, security compliance scan, etc. The CD assembly line releases and delivers the final product package to the released artifact repository if the control gate policies are met.

Developing applications using a DevSecOps software factory provides many benefits:

- Improved software product consistency and quality
- Shortened time to market and increased productivity
- Simplified governance

3.4.3 Operations

In the production environment, the released software is pulled from the released artifact repository and deployed. Operations, operation monitoring, and security monitoring are performed. Production operation tools aim to streamline and automate the deployment, operations, and monitoring activities. Tools should be selected based on system functional requirements and their suitability for the production environment infrastructure.

3.4.4 External Systems

The DevSecOps ecosystem itself and program applications depend on some DoD enterprise services to acquire the necessary baseline tools, application dependencies, and security services to operate.

- **DoD Centralized Artifact Repository (DCAR)** holds the hardened VM images and hardened OCI compliant container images of: DevSecOps tools, container security tools, and common program platform components (e.g. COTS or open source products) that DoD program software teams can utilize as a baseline to facilitate the authorization process.
- **DoD Common Security Services** are DoD enterprise-level common services that facilitate cybersecurity enforcement and IT management. One security service will perform traffic inspection and filtering to protect the mission enclave and mission applications. Some security service examples include firewalls; Intrusion Detection System (IDS)/Intrusion Prevention System (IPS); malware detection; data loss prevention; host-based security; log/telemetry aggregation and analysis; and Identity, Credential, and Access Management (ICAM). A Cybersecurity Service Provider (CSSP) will provide additional services, including Attack Sensing and Warning (ASW), Forensic Media Analysis (FMA), Assurance Vulnerability Management (AVM), Incident Reporting (IR), Incident Handling Response (IHR), Information Operations Condition (INFOCON), Cyber Protection Condition (CPCON), Malware Notification Protection (MNP), and Network Security Monitoring (NSM).

The DevSecOps ecosystem interacts with the enterprise AO for the initial software factory ATO and initial application ATO, as well as the local AO for continuous ATO for the application.

4 DevSecOps Tools and Activities

This section describes both tools for the DevSecOps ecosystem and DevSecOps activities for each phase. Activities and tools are listed in table format. The Baseline column in the tool tables has two values: Minimal Viable Product (MVP) and objective. They indicate whether the tool must be available in the DevSecOps ecosystem MVP as threshold or if the tool is an objective to be reached as the ecosystem matures. Activity tables list a wide range of activities for DevSecOps practice. DoD organizations should define their own processes, choose proper activities, and then select tools suitable for their systems to build software factories and DevSecOps ecosystems. With the DevSecOps maturity progression, the level of activity automation will increase.

4.1 Planning Tools and Activities

The Planning tools support software development planning, which includes configuration management planning, change management planning, project management planning, system design, software design, test planning, and security planning. Table 3 lists some tools that can assist the planning process. Some tools will be used throughout the software lifecycle, such as a team collaboration tool, an issue tracking system, and a project management system. Some tools are shared at the enterprise level across programs. Policy and enforcement strategy should be established for access controls on various tools.

Table 3: Plan Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline

Team collaboration system	Audio/video conferencing;
	chat/messaging;
	brainstorming discussion board;
	group calendars;
	file sharing;
	Wiki website
		Simplify communication and boost team efficiency	Team meetings;
			Design notes;
			Documentation
				Organized teamwork;
				Version controlled documents
					MVP

Issue tracking system	Bugs and defect management;
	Feature and change management;
	Prioritization management;
	Assignment management;
	Escalation management;
	Knowledge base management
		Easy to detect defect trends
		Improve software product quality
		Reduce cost and improve Return on Investment (ROI)
			Bug report
			Feature/change request
			Root cause analysis
			Solutions
				Issues feature/change tickets.
				Issue resolution tracking history
					MVP

Asset inventory management	Collect information about all IT assets;	Increase situation awareness	IT assets (applications, software	Asset inventory	Objective

Tool	Features	Benefits	Inputs	Outputs	Baseline
	Maintain a "real-time" inventory of all applications, software licenses, libraries, operating systems, and versioning information		licenses, libraries, operating systems, and versioning information)		

Configuration management database (CMDB)	Auto-discovery;
	Dependency mapping;
	Integration with other tools;
	Configuration auditing
		Centralized database used by many systems (such as asset management, configuration management, incident management, etc.) during development and operations phases.	IT hardware and software components information	Configuration items	Objective

Project management system	Task management Scheduling and time management Resource management Budget management Risk management	Assist project progress tracking Optimize resource allocation	Tasks, scheduling, resource allocation, etc.	Project plan	MVP
Software system design tool	Assist requirement gathering, system architecture design, components design, and interface design	Independent of programming languages Helps visualize the software system design	User requirements Design ideas	System design documents, Function design document, Test plan, System deployment environment configuration plan	Objective
Threat modeling tool	Document system security design; Analyze the design for potential security issues; Review and analysis against common attack patterns; Suggest and manage mitigation	Allows software architects to identify and mitigate potential security issues early.	System design	Potential threats and mitigation plan	Objective
Data modeling tool	Model the interrelationship and flows between different data elements	Ensure the required data objects by the system are accurately represented	System requirement; Business logic	Data model	Objective if using a database

The activities supported by the plan phase are listed in Table 4. Some activities are suitable at enterprise or program level, such as DevSecOps ecosystem design, project team onboarding planning, and change management planning. Others fit at the project level and are considered continuous in the DevSecOps lifecycle.

Table 4: Plan Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependencies
DevSecOps ecosystem design	Design the DevSecOps process workflows that are specific to this project	- Change management process; - System design; - Release plan & schedule.	DevSecOps process flow chart; DevSecOps ecosystem tool selection; Deployment platform selection	Team collaboration system
Project team onboarding planning	Plan the project team onboarding process, interface, access control policy	Organization policy	Onboarding plan	Team collaboration system

Change management planning	Plan the change control process	- Organizational policy;
		- Software development best practice.
			Change control procedures;
			Review procedures;
			Control review board;
			change management plan
				Team collaboration system;
				Issue tracking system

Configuration management (CM) planning	Plan the configuration control process;
	Identify configuration items
		- Software development, security and operations best practice;
		- IT infrastructure asset;
		- Software system components.
			CM processes and plan;
			CM tool selection;
			Responsible configuration items;
			
			Tagging strategy
				Team collaboration system;
				Issue tracking system

Software requirement analysis	Gather the requirements from all stakeholders	- Stakeholder inputs or feedback; - Operation monitoring feedback; - Test feedback.	-Feature requirements -Performance requirements -Privacy requirements -Security requirements	Team collaboration system; Issue tracking system
System design	Design the system based the requirements	Requirements documents	Documents: -System architecture -Functional design -Data flow diagrams -Test plan -Infrastructure configuration plan -Tool selections -Development tool -Test tool -Deployment platform	Team collaboration system; Issue tracking system Software system design tools

Activities	Description	Inputs	Outputs	Tool Dependencies
Project planning	Project task management Release planning		Task plan & schedule; Release plan & schedule.	Team collaboration system; Project management system
Risk management	Risk assessment	- System architecture; - Supply chain information; - Security risks.	Risk management plan	Team collaboration system;

Configuration identification	Discover or manual input configuration items into CMDB;
	Establish system baselines
		-IT infrastructure asset;
		- Software system components (include DevSecOps tools);
		-code baselines
		-document baselines.
			Configuration items	CMDB;
				Source code repository;
				Artifact repository;
				Team collaboration system

Threat modeling	Identify potential threats, weaknesses and vulnerabilities. Define the mitigation plan	System design	Potential threats and mitigation plan	Threat modeling tool

Database design	Data modeling;
	database selection;
	Database deployment topology
		System requirement;
		System design
			Database design document	Data modeling tool;
				Team collaboration system

Design review	Review and approve plans and documents	Plans and design documents;	Review comments; Action items	Team collaboration system
Documentation version control	Track design changes	Plans and design documents;	Version controlled documents	Team collaboration system

4.2 Software Factory Tools and Activities

Software factory tools include a CI/CD orchestrator, a set of development tools, and a group of tools in the build, test, release, and deliver phases that are pluggable to the CI/CD orchestrator.

4.2.1 CI/CD Orchestrator

The CI/CD Orchestrator is the central automation engine of the CI/CD pipeline. It manages pipeline creation, modification, execution, and termination.

The DevSecOps team creates a pipeline workflow in the Orchestrator by specifying a set of stages, stage conditions, stage entrance and exit control rules, and stage activities. The

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 5 shows the features, benefits, and inputs and outputs of the CI/CD Orchestrator.

Table 5: CI/CD Orchestrator

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

4.2.2 Develop

The Develop phase uses tools to support the development activities that convert requirements into source code. the source code includes application code, test scripts, Infrastructure as Code, Security as Code, DevSecOps workflow scripts, etc. The development team may rely on a single modern integrated development environment (IDE) for multiple programming language support. The IDE code assistance feature aids developers with code completion, semantic coloring, and library management to improve coding speed and quality. The integrated compiler, interpreter, lint tools, and static code analysis plugins can catch code mistakes and suggest fixes before developers check code into the source code repository. Source code peer review or pair programming are other ways to ensure code quality control. All the code generated during development must be committed to the source code repository and thus version controlled. Committed code that breaks the build should be checked in on a branch and not merged into the trunk until it is fixed.

The following tables list the components that facilitate code development, along with their inputs and outputs.

Table 6: Develop Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Integrated development environment (IDE)	Source code editor Intelligent code completion Compiler or interpreter Debugger Build automation (integration with a build tool)	Visual representation Increase efficiency Faster coding with less effort Improved bug fixing speed Reproducible builds via scripts	Developer coding input	Source code	MVP
Integrated development environment (IDE) security plugins	Scan and analyze the code as the developer writes it, notify developer of potential code weakness and may suggest remediation	Address source code weaknesses and aid developers to improve secure coding skills	Source code; known weaknesses	source code weakness findings	Objective
Source code repository	Source code version control Branching and merging Collaborative code review	Compare files, identify differences, and merge the changes if needed before committing. Keep track of application builds	Source code Infrastructure as code	Version controlled source code	MVP
Source code repository security plugin	Check the changes for suspicious content such as Secure Shell (SSH) keys, authorization tokens, passwords and other sensitive information before pushing the changes to the main repository. If it finds suspicious content, it notifies the developer and blocks the commit.	Helps prevent passwords and other sensitive data from being committed into a version control repository	Locally committed source code	Security findings and warnings	Objective
Code quality review tool	View code changes, identify defects, reject or approve the changes, and make comments on specific lines. Sets review rules and automatic notifications to ensure that reviews are completed on time.	Automates the review process which in turn minimizes the task of reviewing the code.	Source code	Review results (reject or accept), code comments	Objective

The activities supported by the develop phase are listed in Table 7.

Table 7: Develop Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependencies
Application code development	Application coding	Developer coding input	Source code	IDE
Infrastructure code development	-System components and infrastructure orchestration coding -Individual component configuration script coding	Developer coding input	Source code	IDE
Security code development	Security policy enforcement script coding	Developer coding input	Source code	IDE

Test development	Develop detailed test procedures, test data, test scripts, test scenario configuration on the specific test tool	Test plan	Test procedure document;
			Test data file;
			Test scripts
				IDE;
				Specific test tool

Database development	Implement the data model using data definition language or data structure supported by the database; Implement triggers, views or applicable scripts; Implement test scripts, test data generation scripts.	Data model	Database artifacts (including data definition, triggers, view definitions, test data, test data generation scripts, test scripts, etc.)	IDE or tools come with the database software
Code commit	Commit source code into version control system	Source code	Version controlled source code	Source code repository
Code commit scan	Check the changes for sensitive information before pushing the changes to the main repository. If it finds suspicious content, it notifies the developer and blocks the commit.	Locally committed source code	Security findings and warnings	Source code repository security plugin
Code review	Perform code review to all source code. Note that pair programming counts.	Source code	Review comments	Code quality review tool
Documentation	Detailed implementation documentation	User input; Source code	Documentation; Auto generated Application Programming Interface (API) documentation	IDE or document editor or build tool
Static code scan before commit	Scan and analyze the code as the developer writes it. Notify developers of potential code weakness and suggest remediation.	Source code; known weaknesses	source code weakness findings	IDE security plugins
Container or VM hardening	Harden the deliverable for production deployment	Running VM or container	Vulnerability report and recommended mitigation	Container security tool Security compliance tool

4.2.3 Build

The build tools perform the tasks of building and packaging applications, services, and microservices into artifacts. For languages like C++, building starts with compiling and linking. The former is the act of turning source code into object code and the latter is the act of combining object code with libraries to create an executable file. For Java Virtual Machine (JVM) based languages, building starts with compiling to class files, then building a compressed file such as a jar, war or ear file, which includes some metadata, and may include other files such as icon images. For interpreted languages, such as Python or JavaScript, there is no need to compile, but lint tools help to check for some potential errors such as syntax errors. Building should also include generating documentation, such as Javadoc, copying files like libraries or icons to appropriate locations, and creating a distributable file such as a tar or zip file. The build script should also include targets for running automated unit tests.

Modern build tools can also be integrated into both an IDE and a source code repository to enable building both during development and after committing. For those applications that use containers, the build stage also includes a containerization tool.

The following tables list build-related tools along with their inputs and outputs.

Table 8: Build Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Build tool	Dependency Management Compile Link (if appropriate) Built-in lint stylistic checking Integration with IDE	Reduces human mistakes Saves time	Source code under version control Artifacts	Binary artifacts stored in the Artifact repository	MVP

Lint tool	Analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs.
	Applicable to both compiled or interpreted languages
		Improve code readability;
		Pre-code review;
		Finding (syntax) errors before execution for interpreted languages
			Source code or scripts	Analyze results	Objective

Container builder	Build a container image based on a build instruction file	Container image build automation	Container base image; Container build file	OCI compliant container image	MVP

Tool	Features	Benefits	Inputs	Outputs	Baseline
Artifact Repository	Binary artifact version control; Container registry	Separate binary control from source control to avoid external access to source control system. Improved build stability by reducing reliance on external repositories. Better quality software by avoiding outdated artifacts with known issues.	Artifacts	Version controlled artifacts	MVP
Static Application Security Test (SAST) tool	SAST analyzes application static codes, such as source code, byte code, binary code, while they are in a non-running state to detect the conditions that indicate code weaknesses.	Catch code weaknesses at an early stage. Continuous assessment during development.	Source code; known vulnerabilities and weaknesses	Static code scan report and recommended mitigation.	MVP
Dependency checking /Bill of Materials (BOM) checking tool	Identify vulnerabilities in the dependent components based on publicly disclosed open source vulnerabilities	Secure the overall application; Manage the supply chain risk	Dependency list or BOM list	Vulnerability report	Objective

The activities supported by the build phase are listed in Table 9.

Table 9: Build Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependencies

Build	Compile and link	Source code; dependencies	Binary artifacts	Build tool;
				Lint tool;
				Artifact repository

Static application security test and scan	Perform SAST to the software system	Source code; known vulnerabilities and weaknesses	Static code scan report and recommended mitigation.	SAST tool
Dependency vulnerability checking	Identify vulnerabilities in the open source dependent components	Dependency list or BOM list	Vulnerability report	Dependency checking / BOM checking tool
Containerize	Packages all required components OS, developed code,	Container base image; Container build file	Container image	Container builder

	libraries, etc.) into a hardened container			

Release packaging	Package binary artifacts, container or VM images, infrastructure configuration scripts, proper test scripts, documentation, checksum, digital signatures, and release notes as a package.	Binary artifacts;
		Scripts;
		Documentation;
		Release notes
			Released package with checksum and digital signature	Release packaging tool

Store artifacts	Store artifacts to the artifact repository	Binary artifacts;
		Database artifacts;
		Scripts;
		Documentation;
		Container images
			Versioned controlled artifacts	Artifact Repository

Build configuration control and audit	Track build results, SAST and dependency checking report;
	Generate action items;
	Make go/no-go decision to the next phase
		Build results;
		SAST report;
		Dependency checking report
			Version controlled build report;
			Action items;
			Go/no-go decision
				Team collaboration system;
				Issue tracking system;
				CI/CD orchestrator

4.2.4 Test

Test tools support continuous testing across the software development lifecycle. Test activities may include, but are not limited to, unit test, functional test, integration test, system test, regression test, acceptance test, performance test, and variety of security tests. . Mission programs can select their own test activities and merge several tests together based on the nature of their software and environment. All tests start with test development, which develops detailed test procedures, test scenarios, test scripts, and test data. Automated test can be executed by running a set of test scripts or running a set of test scenarios on the specific test tool without human intervention. If full automation is not possible, the highest percentage of automation is desired. It is highly recommended to leverage emulation and simulation to test proper integration between components such as microservices and various sensors/systems, so integration testing can be automated as much as possible. Automation will help achieve high test coverage and make continuous ATO practicable, as well as significantly increase the quality of delivered software.

The components involved with the test phase are listed in the following table.

Table 10: Test Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Test development tool	Assists test scenario, test script, and test data development. The specific tool varies, depending on the test activity (such as unit test,	Increase the automation and rate of testing	Test plan	test scenarios, test scripts, test data	MVP

Tool	Features	Benefits	Inputs	Outputs	Baseline
	penetration test) and the application type (e.g., web application, or Hadoop data analytics)				
Test data generator	Generates test data for the system (such as network traffic generator, web request generator)	Increase test fidelity	Test scenario, test data	Input data for the system under test	Objective
Test tool suite	A set of test tools to perform unit test, interface test, system test, integration test, performance test and acceptance test of the software system. Generate test report Specific tool varies depending on the type of tests, software application, and programming language	Increase test automation, speed	Test scenario, test scripts, test data	Test results, test report	MVP
Test coverage tool	Measures how much code is exercised while the automated tests are running	Shows the fidelity of the test results	Application code, automated tests	The percentage of code that is exercised by the tests.	MVP
Test Management Tool	Manages requirements, streamlines test case design from requirements, plans test activities, manages test environment, tracks test status and results.	Increases QA team collaboration and streamlines test processes.	Requirements, test cases, test results	Test progress, test results statistics	Objective
Non-security compliance scan	Such as Section 508 accessibility compliance	Ensures compliance	Artifacts	Compliance report	Objective
Software license compliance checker	Inventory software license; Audit the compliance.	Software license compliance and software asset management	Purchased license info; Software instances	Compliance report	Objective
Dynamic Application Security Test (DAST) tool	DAST tools analyze a running application dynamically and can identify runtime vulnerabilities and environment related issues.	Catch the dynamic code weakness in runtime and under certain environment setting. Identify and fix issues during continuous integration.	Running software application; fuzz inputs	dynamic code scan report and recommended mitigation.	Objective
Interactive Application Security Test (IAST) tool	Analyze code for security vulnerabilities while the application is run by an auto-test, human tester, or any activity "interacting"	Provide accurate results for fast triage; pinpoint the source of vulnerabilities	Running application, and operating systems; Fuzz inputs	Analysis report and recommended mitigation.	Objective

Tool	Features	Benefits	Inputs	Outputs	Baseline
	with the application functionality				
Container security tool	Container image scan OS check	Ease the container hardening process	Container images or running containers	Vulnerability report and recommended mitigation.	MVP
Container policy enforcement	Support for Security Content Automation Protocol (SCAP) and Container configuration policies. These policies can be defined as needed.	Automated policy enforcement	Policies in SCAP form.	Compliance report	MVP
Security compliance tool	Scan and report for compliance regulations, such as DISA Security Technical Implementation Guides (STIGs), NIST 800-53.	Speed up ATO process.	Container images.	Vulnerability report and recommended mitigation.	Objective
Network security test tool	Simulate real-world legitimate traffic, distributed denial of service (DDOS), exploits, malware, and fuzzing.	Validate system security; increase attack readiness; reduce the risk of system degradation.	Test configuration	Test traffic	Objective
Database test tool suite	Tools that facilitate database test; It includes test data generator, database functional test tool, database load test tool;	Automate or semi-automate the database tests	Test data; Test scenario	Test results	Objective if using a database
Database security scan and test tool	Find the database common security vulnerabilities, such as weak password, known configuration risks, missing patches; Structured Query Language (SQL) injection test tool; Data access control test; User access control test; Denial of service test	Reduce the security risks	Test data; Test scenarios	Vulnerability findings; Recommended mitigation actions	Objective if using a database

The activities supported by the test phase are listed in Table 11. These activities happen at different test stages.

- Development stage: unit test, SAST discussed in the build phase
- System test stage: DAST or IAST, integration test, system test
- Pre-production stage: manual security test, performance test, regression test, acceptance test, container policy enforcement, and compliance scan

Test audit, test deployment, and configuration audit happen at all stages.

Table 11: Test Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependencies
Unit test	Assist unit test script development and unit test execution. It is typically language specific.	Unit test script, individual software unit under test (a function, method or an interface), test input data, and expected output data	Test report to determine whether the individual software unit performs as designed.	Test tool suite, Test coverage tool
Dynamic application security test and scan	Perform DAST or IAST testing to the software system	Running application and underlying OS; fuzz inputs	Vulnerability, static code weakness and/or dynamic code weakness report and recommended mitigation	DAST tool or IAST tool
Integration test	Develops the integration test scripts and execute the scripts to test several software units as a group with the interaction between the units as the focus.	Integration test scripts, the software units under test, test input data, and expected output data	Test report about whether the integrated units performed as designed.	Test tool suite
System test	System test uses a set of tools to test the complete software system and its interaction with users or other external systems.	System test scripts, the software system and external dependencies, test input data and expected output data	Test result about if the system performs as designed.	Test tool suite
Manual security test	Such as penetration test, which uses a set of tools and procedures to evaluate the security of the system by injecting authorized simulated cyber-attacks to the system. CI/CD orchestrator does not automate the test, but the test results can be a control point in the pipeline.	Running application, underlying OS, and hosting environment	Vulnerability report and recommended mitigation	Varies tools and scripts (may include network security test tool)
Performance test	Ensure applications will perform well under the expected workload. The test focus is on application response time, reliability, resource usage and scalability.	Test case, test data, and the software system	Performance metrics	Test tool suite, Test data generator
Regression test	A type of software testing to confirm that a recent program or code change has not adversely affected existing features.	Functional and non-functional regression test cases; the software system	Test report	Test tool suite

Activities	Description	Inputs	Outputs	Tool Dependencies
Acceptance test	Conduct operational readiness test of the system. It generally includes: Accessibility and usability test failover and recovery test performance, stress and volume test security and penetration test interoperability test compatibility test supportability and maintainability	The tested system Supporting system Test data	Test report	Test tool suite, Non-security compliance scan
Container policy enforcement	Check developed containers to be sure they meet container policies	Container, Policies in SCAP form	Container compliance report	Container policy enforcement

Compliance scan	Compliance audit	Artifacts;
		Software instances;
		System components
			Compliance reports	Non-security compliance scan;
				Software license compliance checker;
				Security compliance tool

Test audit	Test audit keeps who performs what test at what time and test results in records	Test activity and test results	Test audit log	Test management tool
Test deployment	Deploy application and set up testing environment using Infrastructure as Code	Artifacts (application artifacts, test code) Infrastructure as Code	The environment ready to run tests	Configuration automation tool; IaC
Database functional test	Perform unit test and functional test to database to verify the data definition, triggers, constrains are implemented as expected	Test data	Test results	Database test tools
Database non-functional test	Conduct performance test, load test, and stress test; Conduct failover test	Test data; Test scenarios	Test results	Database test tools
Database security test	Perform security scan; Security test	Test data; Test scenarios	Test results	Vulnerability findings; Recommended mitigation actions

Test configuration control and audit	Track test and security scan results;
	Generate action items;
	Make go/no-go decision to the next phase.
		Test results;
		Security scan and compliance scan report
			Version controlled test results;
			Action items;
			Go/no-go decision
				Team collaboration system;
				Issue tracking system;
				CI/CD orchestrator

Activities	Description	Inputs	Outputs	Tool Dependencies
	(There may be several iterations for several tests across stages)			

4.2.5 Release and Deliver

In the release and deliver phase, the software artifacts are digitally signed to verify that they have passed build, all tests, and security scans. They are then delivered to the artifact repository. The content of the artifacts depends on the application. It may include, but is not limited to, container images, VM images, binary executables (such as jar, war, ear files), test results, security scan results, and Infrastructure as Code deployment scripts. Artifacts will be tagged with the release tag if GO release decision is made based on the configuration audit results. The artifacts with the release tag are delivered to production.

Table 12: Release and Deliver Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Release packaging tool	Package binary artifacts, container or VM images, infrastructure configuration scripts, proper test scripts, documentation, release notes as a package; generate checksum and digital signature for the package. The package may be prepared for a specific installer or it is a self-extracting installer itself.	Release package (such as a bundle of artifacts, self-extracting software installer, software tar file, etc.)	Binary artifacts, base containers or VM images, infrastructure configuration scripts, proper test scripts, documentation, release notes	Release package with checksum and digital signature (a bundle of artifacts, such as a self-extracting software installer, or a tar file, etc.)	MVP if using VMs

The mission program could have more than one artifact repository, though more likely there is a centralized one and tags separate artifact types. One artifact repository (or set of tags) is used in the build stage to store build results. The test deployment activity can fetch the artifacts from the build stage artifact repository to deploy the application into various environments (development, test, or pre-production). Another artifact repository (or set of tags) may be used by the production environment, which is the one that the store artifacts stage uses to push the final deliverables to production. The production deployment will get all the artifacts from the production artifact repository to deploy the application.

Some mission program application systems have geographically distributed operational regions across the country or even overseas. In order to increase deployment velocity, a remote operational region may have its own local artifact repository that replicates the artifact repository completely or partially. During release, a new artifact is pushed into the artifact repository and then replicated to other regional artifact repositories.

The activities supported by the release and deliver phase are listed below.

Table 13: Release and Deliver Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependency
Release go / no-go decision	This is part of configuration audit; Decision on whether to release artifacts to the artifact repository for the production environment.	Design documentation; Test reports; Security test and scan reports; Artifacts	go / no-go decision; Artifacts are tagged with release tag if go decision is made	CI/CD Orchestrator
Deliver released artifacts	Push released artifacts to the artifact repository	The release package	New release in the artifact repository	Artifacts repository
Artifacts replication	Replicate newly release artifacts to all regional artifact repositories	Artifacts	Artifacts in all regional artifact repositories	Artifacts repositories (release, regional)

4.3 Production Operation Tools and Activities

The production operations tools provide the capability to deploy artifacts, including containers and VM images, to the production environment, and to monitor their operations. For Cloud-native applications, it is recommended to use containers whenever possible over virtual machines due to the baked-in security provided by the Sidecar Container Security Stack.

4.3.1 Deploy

The tools used in deploy phase are environment and deployment mode dependent.

4.3.1.1 Virtual Machine deployment

While it is highly recommended to leverage containers for new system design and development, if the application is deployed as a VM, the virtualization manager in the hosting environment is the key component with which IaC will interface to deploy and configure the application system. The virtualization manager manages the virtual compute, storage and network resources. In some hosting environments, such as a general-purpose cloud, the virtualization manager also provides some security capabilities, such as micro-segmentation, which creates security zones to isolate VMs from one another and secure them individually. Several capabilities of the virtualization manager are keys to the success of mission application runtime operation and security, such as health checking, virtual resource monitoring, and scaling. The application production environment infrastructure has to leverage these capabilities in its architecture and configuration.

The use of “clones” from a master image library enables VMs to be created quickly. A clone is made from a snapshot of the master image. The use of clones also enables the concept of immutable infrastructure by pushing updated, clean images to the VM each time it is started. Only the master image needs to be patched or updated with the latest developed code; each running image is restarted to pick up these changes.

4.3.1.2 Container deployment

A container manager provides capabilities that check for new versions of containers, deploys the containers to the production environment, and performs post-deployment checkout.

The container manager consists of an OCI-compliant container runtime and a CNCF-certified Kubernetes, which is an orchestration tool for managing microservices or containerized applications across a cluster of nodes. The nodes could be bare metal servers or VMs. The container manager may be owned by a mission program or provided by the cloud hosting environment. It simplifies container management tasks, such as instantiation, configuration, scaling, monitoring, and rolling updates. The CNCF-certified Kubernetes interacts with the underlying virtualization manager in the cloud environment to ensure each node’s health and performance, and scale it as needed. This scaling includes container scaling within the CNCF-certified Kubernetes cluster, but when running in a cloud, it also includes the ability to auto-scale number of nodes in a cluster by adding or deleting VMs.

The following tables list deployment-related tools and their inputs and outputs.

Table 14: Deploy Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Virtualization Manager	VM instance management VM resource monitoring (provided on hosting environment)	Centralized VM instantiation, scaling, and monitoring	VM instance specification and monitoring policy	Running VM	MVP if using VMs
CNCF-certified Kubernetes	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	Simplify operations by deployment and update automation Scale resources and applications in real time Cost savings by optimizing infrastructure resources	Container instance specification and monitoring policy	Running container	MVP

Tool	Features	Benefits	Inputs	Outputs	Baseline
Data masking tool	Shield personally identifiable information or other confidential data	Provide data privacy; Reduce the risk of data loss during data breach	Original data	Masked data	Objective if database contains sensitive data
Database encryption tool	Encrypt data at rest and in transit	Provide data privacy and security; Prevent data loss	Original data	Encrypted data	MVP if database contains highly sensitive data

Database automation tool	Automate database tasks, such as deployments, upgrades, discovering and troubleshooting anomalies, recovering from failures, topology changes, running backups, verifying data integrity, and scaling.	Simplify database operations and reduce human errors	Database artifacts;
			Data;
			Running status and events
				Status report;
				Warnings;
				alerts
					Objective if using a database

Configuration automation tools	Execute the configuration scripts to provision the infrastructure, security policy, environment, and the application system components.	Configuration automation Consistent provisioning	Infrastructure configuration scripts Infrastructure configuration data	Provisioned deployment infrastructure	MVP
Service mesh	Ability to create a network of deployed services with load balancing, service-to-service authentication, and monitoring.	Support for microservice interactions.	Control plane: service communication routing policies, authentication certificates. Data plane: service communication data	Control plane: service status reports Data plane: routed service communication data	MVP

The activities supported by the deploy phase are listed in Table 15.

Table 15: Deploy Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependency
Artifact download	Download newly release artifacts from the artifact repository	Artifact download request	Requested artifacts	Artifact repository
Infrastructure provisioning automation	Infrastructure systems auto provisioning (such as software defined networking, firewalls, DNS, auditing and logging system, user/group permissions, etc.)	Infrastructure configuration scripts / recipes / manifests / playbooks	Provisioned and configured infrastructure	Configuration automation tools; IaC

Activities	Description	Inputs	Outputs	Tool Dependency
Create linked clone of VM master image	Instantiate VM by creating a link clone of parent VM with master image	VM parent New VM instance parameters	New VM instance	Virtualization Manager
Deliver container to container registry	Upload the hardened container and associated artifacts to the container registry	Hardened container	New container instance	CNCF-certified Kubernetes; Artifact repository container registry
Post-deployment security scan	System and infrastructure security scan	Access to system components and infrastructure components	Security vulnerability findings	Security compliance tool
Post-deployment checkout	Run automated test to make sure the important functions of system are working	Smoke test scenarios and test scripts	Test results	Test scripts

Database installation and database artifact deployment	Database software installation; Cluster or high availability setup;
	Database artifacts deployment and data loading
		Artifacts in the repository; data	Running database system	Artifact repository;
				Database automation tool;
				Data masking or encryption tool if needed

4.3.2 Operate

The Operate phase uses tools for system scaling, load balancing, and backup.

Load balancing monitors resource consumption and demand, and then distributes the workloads across the system resources. Scaling helps dynamic resource allocation based on demand. Both virtualization manager and CNCF-certified Kubernetes support load balancing and scaling capabilities. CNCF-certified Kubernetes handles the load balancing and scaling at the container level. The virtualization manager works at the VM level.

Application deployment must have proper load balancing and scaling policies configured with the virtualization manager or the CNCF-certified Kubernetes based on VM deployment or container deployment respectively. During runtime, the management layer will continuously monitor the resources. If the configured threshold is met (for example if memory or Central Processing Unit (CPU) usage meets a pre-set threshold), then the system triggers the load balancing or scaling action automatically. Auto-scaling must be able to scale both up and down.

Table 16: Operate Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Backup management	Data backup System components (VM or container) snapshot	Improve failure recovery	Access to the backup source	Backup data System VM or container snapshot	MVP
Operations dashboard	Provide operators a visual view of operations status, alerts, and actions.	Improve operations management	All operational monitoring status, alerts, and recommended actions	Dashboard display	Objective

The activities supported by the operate phase are listed in the table below.

Table 17: Operate Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependency
Backup	Data backup; System backup	Access to backup system	Backup data or image	Backup management; Database automation tool
Scale	Scale manages VMs/containers as a group. The number of VMs/containers in the group can be dynamically changed based on the demand and policy.	Real-time demand and VM/container performance measures Scale policy (demand or Key Performance Indicator (KPI)threshold; minimum, desired, and maximum number of VMs/containers)	Optimized resource allocation	VM management capability on the hosting environment; Container management on the hosting environment
Load balancing	Load balancing equalizes the resource utilization	Load balance policy Real time traffic load and VM/container performance measures	Balanced resource utilization	VM management capability on the hosting environment; Container management on the hosting environment

4.3.3 Monitor

In the monitor phase, tools are utilized to collect and assess key information about the use of the application to discover trends and identify problem areas. Monitoring spans the underlying hardware resources, network transport, applications / microservices, containers, interfaces, normal and anomalous endpoint behavior, and security event log analysis.

It also includes behavior and signature-based detection in the runtime environment. All these security capabilities are mapped against the NIST controls and follow NIST Special Publication 800-190 Application Container Security Guide [12] for continuous compliance.

Table 18: Monitor Phase Tools

Tool	Features	Benefits	Inputs	Outputs	Baseline
Logging	Logging events for all user, network, application, and data activities	Assist troubleshooting the issues. Assist detection of advanced persistent threats and forensics.	All user, network, application, and data activities	Event logs	MVP
Log aggregator	Filter log files for events of interest (e.g., security), and transform into canonical format		Logs	Aggregated, filtered, formatted event log	MVP
Log analysis & auditing	Analyze and audit to detect malicious threats / activity; Automated alerting and workflows for response Forensics for damage assessment		Logs	Alert messages, emails, etc. Remediation report and log	MVP
Operations monitoring	Report various performance metrics such as resource utilization rates, number of concurrent user sessions, and Input/Output (IO) rates; Provide dashboards to display performance; Alert performance issues Establish a baseline for comparison	Improve operations continuity Identify the area to improve Better end-user experience	Performance KPI and Service Level Agreement (SLA)	Performance statistics Performance alerts	MVP
Information Security Continuous Monitoring (ISCM)	Monitor network security Monitor personnel activity Monitor configuration changes Perform periodical security scan to all system components Monitor the IT assets and detect deviations from security, fault tolerance, performance best practices. Monitor and analyze log files Audit IT asset's configuration compliance Detect and block malicious code	Detect unauthorized personnel, connections, devices, and software Identify cybersecurity vulnerability Detect security and compliance violation Verify the effectiveness of protective measures	IT asset Network Personnel activities Known vulnerabilities	Vulnerabilities Incompliance Findings, assessments and recommendations	MVP

Tool	Features	Benefits	Inputs	Outputs	Baseline
	Continuous security vulnerability assessments and scans Provide browse, filter, search, visualize, analysis capabilities Generate findings, assessments and recommendations. Provide recommendations and/or tools for remediating any non-compliant IT asset and/or IT workload.				
Alerting and notification	Notify security teams and/or administrators about detected events. Support automatic remediation of high-priority time-critical events.	Improve visibility of system events Reduce system downtime Improve customer service	Logs, monitoring data. Support automatic remediation of high-priority time-critical events.	Alert messages, emails, etc. Remediation report Issue ticket	MVP
Database monitoring tool	Baseline database performance and database traffic; Detect anomalies	Improve database operations continuity	Running database	Logs; Warnings and alerts	Objective if using a database
Database security audit tool	Perform user access and data access audit; Detect anomalies from events correlation; Detect SQL injection; Generate alert	Enhance database security	Running database	Audit logs; Warnings and alerts	MVP if using a database

The activities supported by the monitor phase are listed in Table 19.

Table 19: Monitor Phase Activities

Activities	Description	Inputs	Outputs	Tool Dependencies
Logging	Log system events	All user, network, application, and data activities	Logs	Logging
Log analysis & auditing	Filter or aggregate logs; Analyze and correlate logs	Logs	Alerts and remediation report	Log aggregator Log analysis & auditing

System performance monitoring	Monitor system hardware, software, database, and network performance;
	Baselining system performance;
		Running system	Performance KPI measures;
			Recommended actions;
			Warnings or alerts
				Operation monitoring
				Issue tracking system; Alerting and notification;

	Detect anomalies			Operations dashboard

System Security monitoring	Monitor security of all system components
	Security vulnerability assessment
	System security compliance scan
		Running system	Vulnerabilities;
			Incompliance Findings; assessments and recommendations;
			Warnings and alerts.
				ISCM;
				Issue tracking system;
				Alerting and notification;
				Operations dashboard

Asset Inventory	Inventory system IT assets	IT assets	Asset inventory	Inventory Management;

System configuration monitoring	System configuration (infrastructure components and software) compliance checking, analysis, and reporting	Running system configuration;
		Configuration baseline
			Compliance report;
			Recommended actions;
			Warnings and alerts
				ISCM;
				Issue tracking system; Alerting and notification;
				Operations dashboard

Database monitoring and security auditing	Database performance and activities monitoring and auditing	Database traffic, event, and activities	Logs;
			Warnings and alerts
				Database monitoring tool;
				Database security audit tool;
				Issue tracking system;
				Alerting and notification;
				Operations dashboard

4.4 Security Tools and Activities Summary

Security is not a separate phase of the DevSecOps lifecycle; rather security activities occur in all phases. This DevSecOps security practice facilitates automated risk characterization, monitoring, and mitigation across the application lifecycle. Table 20 summarizes the security activities of all phases.

Table 20: Security Activities Summary

Activities	Phase	Activities Table Reference	Tool Dependencies	Tool Table Reference
Threat modeling	Plan	Table 4	Threat modeling tool	Table 3
Security code development	Develop	Table 7	IDE	Table 6
Static code scan before commit	Develop	Table 7	IDE security plugins	Table 6
Code commit scan	Develop	Table 7	Source code repository security plugin	Table 6
Container or virtual machine hardening	Develop	Table 7	Container security tool Security compliance tool	Table 10
Static application security test and scan	Build	Table 9	SAST tool	Table 8

Dependency vulnerability checking	Build	Table 9	Dependency checking / BOM checking tool	Table 8
Dynamic application security test and scan	Test	Table 11	DAST tool or IAST tool	Table 10
Manual security testing (such as penetration test)	Test	Table 11	Varies tools and scripts (may include network security test tool)	Table 10
Container policy enforcement	Test	Table 11	Container policy enforcement	Table 10
Post-deployment security scan	Deploy	Table 15	Security compliance tool	Table 10
System Security monitoring	Monitor	Table 19	Information Security Continuous Monitoring (ISCM)	Table 18

4.5 Configuration Management Tools and Activities Summary

Configuration management plays a key role in DevSecOps practice. It ensures the configuration of a software system's infrastructure, software components, and functionalities are not only known initially but also knowable and well controlled throughout the DevSecOps lifecycle.

Configuration management consists of three sets of activities:

- Configuration identification: identify the configuration items. This can be done manually or with assistance from a discovery tool. The configuration items include infrastructure components, COTS or open source software components used in the system, documented software design, features, software code or scripts, artifacts, etc.
- Configuration control: control the changes of the configuration items. Each configuration item has its own attributes, such as model number, version, configuration setup, license, etc. The CMDB, source code repository, and artifact repository are tools to track and control the changes. The source code repository is used primarily during development. The other two are used in both development and operations.
- Configuration verification and audit: verify that the configuration items meet the documented requirements and design. Configuration verification and audit are control gates along a pipeline to control the go/no-go decision to the next phase.

Table 21: Configuration Management Activities Summary

Activities	Phase	Activities Table Reference	Tool Dependencies	Tool Table Reference
Configuration management planning	Plan	Table 4	Team collaboration system; Issue tracking system	Table 3
Configuration identification	Plan	Table 4	CMDB	Table 3
Design review	Plan	Table 4	Team collaboration system	Table 3
Documentation version control	Plan	Table 4	Team collaboration system	Table 3
Code review	Develop	Table 7	Code quality review tool	Table 6

Activities	Phase	Activities Table Reference	Tool Dependencies	Tool Table Reference
Code Commit	Develop	Table 7	Source code repository	Table 6
Store artifacts	Build	Table 9	Artifact repository	Table 8
Build phase configuration control and audit	Build	Table 9	Team collaboration system; Issue tracking system CI/CD orchestrator	Table 3 Table 5
Test phase configuration control and audit	Test	Table 11	Team collaboration system; Issue tracking system CI/CD orchestrator	Table 3 Table 5
Release go / no-go decision	Release	Table 13	CI/CD orchestrator	Table 5
Infrastructure provisioning automation	Deploy	Table 15	Configuration automation tool	Table 14
Post-deployment security scan	Deploy	Table 15	Security compliance tool	Table 10
Post-deployment checkout	Deploy	Table 15	Test scripts	
Asset inventory	Monitor	Table 19	Asset inventory tool	Table 18

System performance monitoring	Monitor	Table 19	Operation monitoring
			Issue tracking system;
			Alerting and notification;
			Operations dashboard
				Table 3
				Table 16
				Table 18

System configuration monitoring	Monitor	Table 19	ISCM;
			Issue tracking system;
			Alerting and notification;
			Operations dashboard
				Table 3
				Table 16
				Table 18

4.6 Database Management Tools and Activities Summary

Databases are commonly used in the DoD software systems. They hold some of the most critical information of an enterprise or a mission and are typically the center piece of the software system. Data security and privacy protection are paramount to enterprises and missions. Relational databases continue to be a prime target for data thieves, and security vulnerabilities are compound by adoption of big data platforms, such as Hadoop, NoSQL databases, and Database as a Service (DBaaS) in the cloud. Here we discuss some database activities throughout the DevSecOps lifecycle to improve database security and operations.

In development phases, database design, development, and testing activities generate database artifacts, which are data models, database schema files, trigger definitions, view definition, test data, test data generation scripts, test scripts, etc. These database artifacts must be under configuration management control. During test phase, database functional test is like application code unit test and functional test to validate the schema, triggers, and data compliance. The non-functional test includes load testing, stress test, and performance test. The security test focuses on vulnerability scan, user authentication and authorization, unauthorized access to data, data encryption, privilege elevation, SQL injection, and denial of service.

During operations, the deployment and operational activities can be automated via database automation tools. The continuous monitoring is achieved using database monitoring tool and security audit tool.

Table 22 summarizes the database activities of all phases.

Table 22: Database Management Activities Summary

Activities	Phase	Activities Table Reference	Tool Dependencies	Tool Table Reference
Database design	Plan	Table 4	Data modeling tool	Table 3
Database development	Develop	Table 7	IDE or tools come with the database software	Table 6
Code review (database schemas, codes)	Develop	Table 7	Code quality review tool	Table 6
Code Commit (database schemas, codes	Develop	Table 7	Source code repository	Table 6
Store artifacts (database artifacts)	Build	Table 9	Artifact repository	Table 8
Database functional test	Test	Table 11	Database test tool	Table 10
Database non-functional test	Test	Table 11	Database test tool	Table 10
Database security test	Test	Table 11	Database security test tool	Table 10

Database installation and database artifact deployment	Test and Deploy	Table 11
		Table 15
			Artifact repository;
			Database automation tool;
			Data masking or encryption tool if needed
				Table 8
				Table 14

Backup	Operate	Table 17	Database automation tool	Table 14
Database monitoring and security auditing	Monitor	Table 19	Database monitoring tool; Database security audit tool	Table 18

5 DoD Enterprise DevSecOps Container Service

The DoD Enterprise DevSecOps Container Service creates DevSecOps hardened containers and provides hardened container access service to DoD programs to instantiate their own DevSecOps ecosystem.

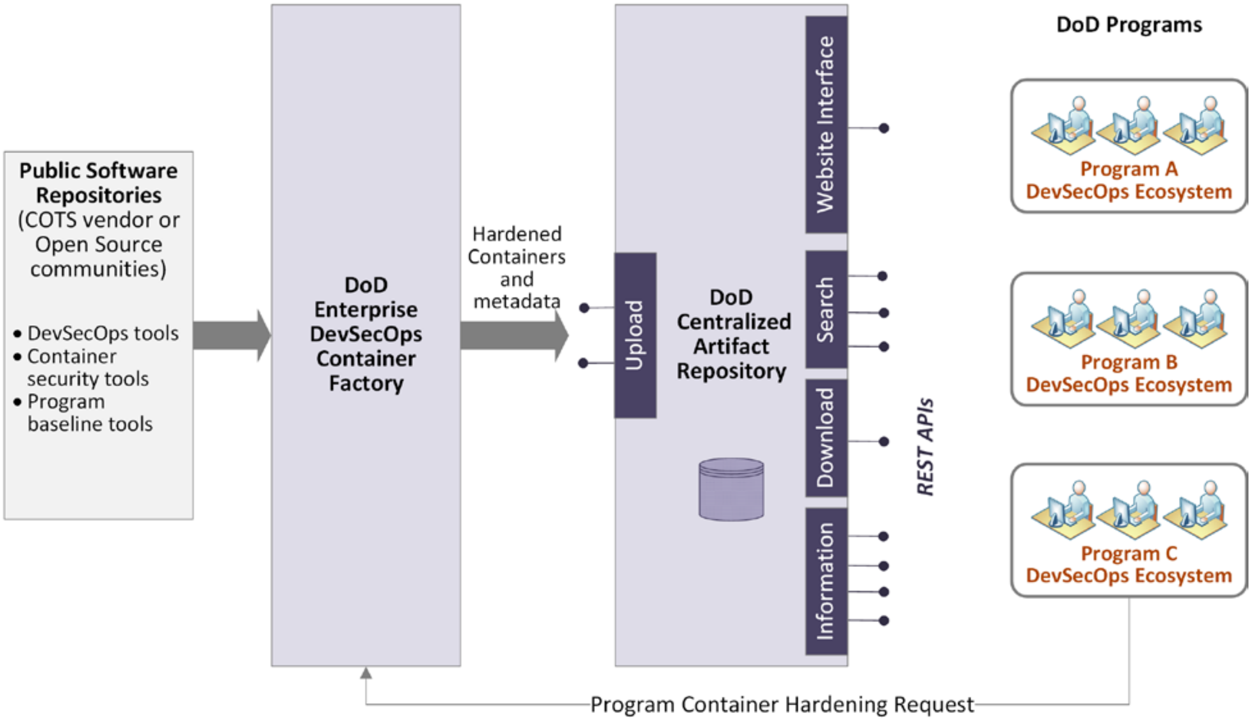

Public Software Repositories
(COTS vendor or Open Source communities)
• DevSecOps tools
• Container security tools
• Program baseline tools
DoD Enterprise DevSecOps Container Factory
Hardened Containers and metadata
Upload
DoD Centralized Artifact Repository
Website Interface
Search
Download
Information
REST APIs
DoD Programs
Program A DevSecOps Ecosystem
Program B DevSecOps Ecosystem
Program C DevSecOps Ecosystem
Program Container Hardening Request

Figure 10: DoD Enterprise DevSecOps Container Service Architecture

Figure 10 illustrates the DoD Enterprise DevSecOps Container Service architecture. It contains a DoD Enterprise DevSecOps Container Factory and a DoD Centralized Artifact Repository (DCAR). The Container Factory takes public container images as input and automates the container hardening process to produce the hardened container images. The DCAR stores the hardened container images and allows DoD programs access these images.

5.1 DoD Enterprise DevSecOps Container Factory

The Container Factory produces the hardened containers of DevSecOps tools. It is imperative for the Container Factory to automate its hardening process as much as possible. It does this by leveraging CI/CD pipelines in an instance of the software factory specifically configured for hardening of DevSecOps tool containers.

5.1.1 DoD Hardened Containers

A DoD hardened container is an Open Container Image (OCI) compliant image that is secured and made compliant with the DoD Container Hardening Security Requirements Guide [6].

Container images should adhere to the OCI Image Format Specification to ensure portability. Each hardened container includes the "global configuration", which includes all security hardening configuration. The packaged container is tagged with integrity metadata, such as a digital signature or a digital hash. Tags may be implemented as metadata bound to the object, or as attributes in a file associated with the object. Some configuration values can be changed for local use, such as the DNS location. These "local configuration" values are outside the scope of the integrity tag, and do not "break" the chain of trust for the hardened container.

Artifacts related to the hardened container should include the Information Assurance (IA) controls that the hardened container has successfully addressed, so that users of the container know which controls they can inherit, versus which controls they must address. This capability may not exist in the MVP, but it is an objective that enables reciprocity.

The Hardened Container Factory produces following types of containers:

- Hardened containers of DevSecOps CI/CD pipeline tools
- Sidecar Container Security Stack (see details in Section 6.4.4) containers to be used in runtime environments for container security
- Common containers (such as OS, database, web servers, etc.) to be used as a program development baseline

5.1.2 Container Hardening Process

The Container hardening process, including required documentation, sustainment of the hardened containers, and cybersecurity requirements, is fully described in the DoD Enterprise DevSecOps Initiative Hardening Containers [13]. The basic process is depicted in Figure 11.

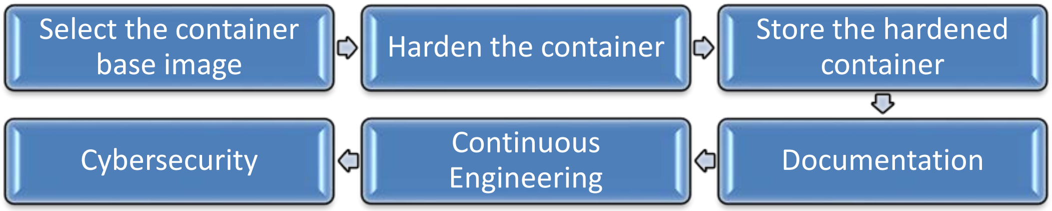


Figure 11: Major Steps in the Container Hardening Process

5.1.2.1 Select the Container Base Image

A base image is a container image that comes from a vendor or an open source community; it is used as the starting point to create a hardened container image. Use the base image without creating forks to enable direct coupling with its updates. The container should be built starting with the respective DoD hardened base OS STIG image.

5.1.2.2 Harden the Container

The Container Factory uses a set of instructions given in [13] to harden the container to mitigate findings and ensure proper DoD compliance. Reuse the instructions as much as possible between versions so that the hardening will be consistent across versions. It is possible that new versions bring new features, which may require additional hardening.

The DoD Centralized Container Source Code Repository (DCCSCR) is used to store instruction files to build container images, associated checksums, and various documentation. The source code repository is centrally hosted so hardeners can store their code and leverage a CI/CD pipeline (Container Factory). This is what feeds the container hardening process and DCAR repository.

Use the CI/CD orchestration tool; download the DCCSCR folder content into the pipeline and use the Instructions file to build the container.

The CI/CD pipeline will then run the required Container Hardening Scanners, scanning the container image. Based on the findings of the Container Hardening Scanners, add instructions to mitigate the findings as needed. Rebuild and rescan until findings are mitigated or accepted.

5.1.2.3 Store the Hardened Container

The DCAR is used to store the hardened containers, associated checksums, and various documentation. This repository will be centrally hosted. Each container will have its own folder in the DCAR. Subfolders should be used for versioning.

Store the hardened container and checksum inside the DCAR. It will be tagged as "pre-production" as well until the artifact receives an ATO, in which case it will then be tagged for "production".

5.1.2.4 Documentation

Content and documentation provided for the hardened container will include:

- A description of the container, how to deploy it, the functional capabilities it provides, and its interfaces.
- Scripts, including the instructions file for building the container, and related configuration files for deploying and scaling the hardened image or container
- Security test results including findings, false positives, and a recommended mitigation plan.
- Accepted risks, including a Plan of Action and Milestones (POA&M) for critical and high findings that are not yet resolved.
- Change log of significant changes since the last version.
- Human-readable licenses for all products are included within the container. COTS license keys will not be included, and they will need to be acquired separately.

5.1.2.5 Continuous Engineering

The hardened container factory CI/CD pipeline searches for and downloads new base images that are posted by the vendor or in an open source community repository and runs the steps in the Section 5.1.2.2. These steps are triggered automatically, as soon as a new image is released into the open source repository. If the build passes all scans, it should automatically store the new container into DCAR, where it will be tagged as “pre-production”. It also automatically notifies the team if a build fails to pass any of the scans.

5.1.2.6 Cybersecurity

The hardened containers produced by the factory should meet the related cybersecurity requirements, which include NIST Special Publication (SP) 800-53 [14], NIST SP 800-37 [15], the DoD DISA Security Technical Implementation Guides (STIGs) and Security Requirements Guides (SRGs), and industry best practices.

5.2 DoD Centralized Artifact Repository

The DCAR holds the DoD hardened container images that the DoD Enterprise DevSecOps Container Factory produces. DoD program DevSecOps teams can utilize these to instantiate their own DevSecOps ecosystem and software factory. The DCAR also holds the DoD hardened containers for base operating systems, web servers, application servers, databases, API gateways, message buses, and additional enterprise capabilities for use by DoD program software teams as a program system deployment baseline. Separate hardened container images are created for different versions of base images. DCAR hardened container images are version controlled. These hardened containers, along with security accreditation reciprocity, greatly simplify and speed the process of obtaining an Approval to Connect (ATC) or Authority to Operate (ATO).

The DCAR provides the capability to allow DoD programs (including approved DoD contractors) to search, list information about, and download artifacts from the repository for DoD software development on the approved environment.

6 DevSecOps Ecosystem Reference Designs

This section will discuss two software factory reference designs. One is based on the DoD Enterprise DevSecOps Container Service offering to create a software factory using DevSecOps tool hardened containers from DCAR. The other is based on DoD authorized cloud DevSecOps service offerings as provided by a CSP.

This section also discusses secure operations for containerized applications in a production environment by leveraging container security tools in the DCAR.

6.1 Containerized Software Factory

A containerized software factory can be instantiated using a set of DevSecOps hardened containers that are offered in the DCAR. These enterprise 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. Figure 12 illustrates a containerized software factory reference design. The software factory is built on an underlying container orchestration layer and a host environment. It produces DoD applications as the product. These applications use different sets of hardened containers from the DCAR than the ones used to create the software factory.

DoD programs may have already implemented a DevSecOps platform. One of the pain points is sustaining that platform. It is highly recommended that, as incremental updates are made to the existing platform, the program migrates capabilities to the DoD Enterprise DevSecOps hardened containers. For those cases where a DoD Enterprise hardened container is not available, or requires a custom policy, the program in conjunction with the DoD Enterprise DevSecOps program office is encouraged to create, sustain, and deliver the hardened container to the DCAR.

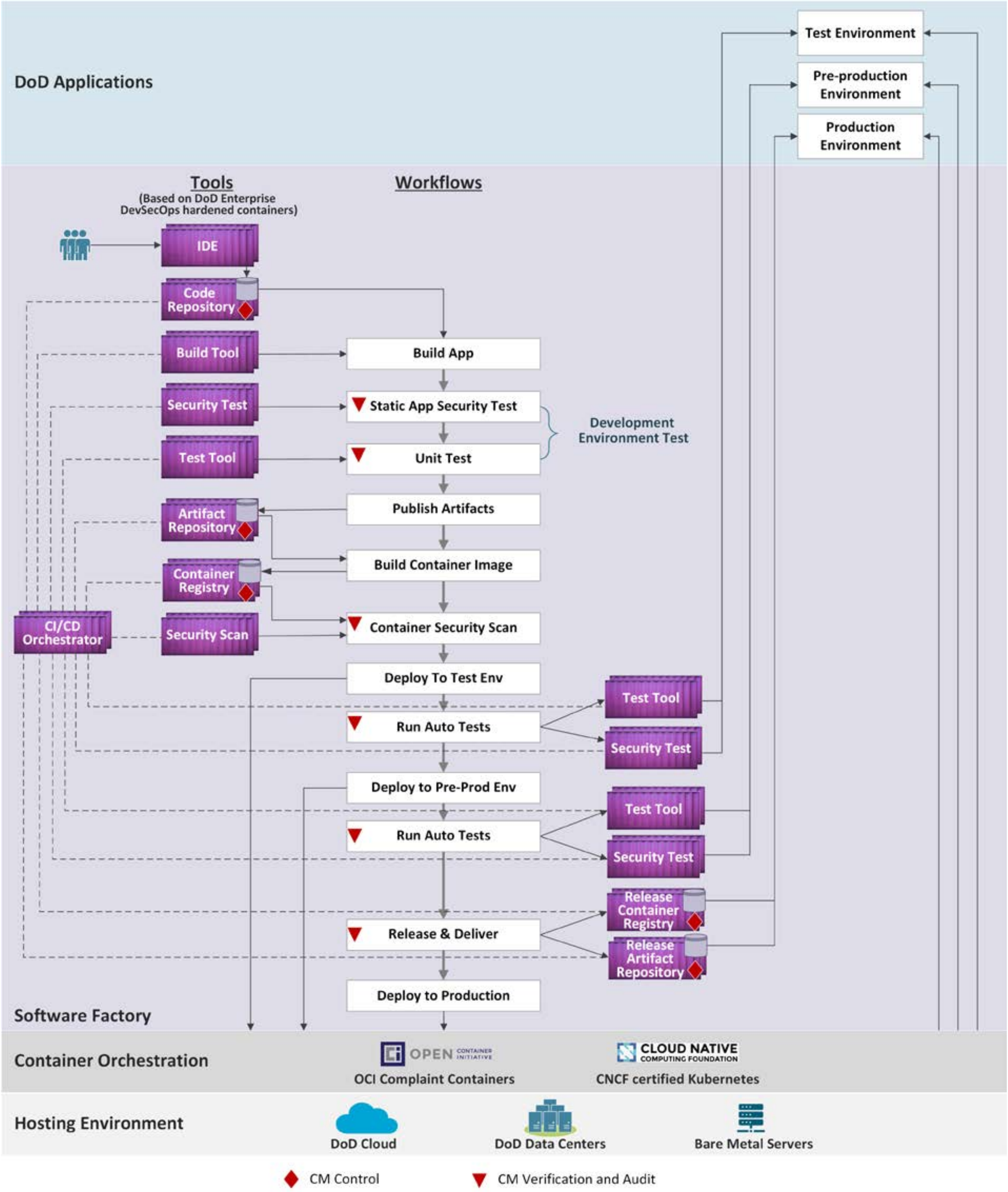


Figure 12: Containerized Software Factory Reference Design

6.1.1 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.

6.1.2 Container Orchestration

In order to support containerized software factory tools, the underlying container orchestration must use CNCF certified Kubernetes and support OCI compliant containers. CNCF-certified Kubernetes orchestrates containers, interacts with underlying hosting environment resources, and coordinates clusters of nodes at scale in development, testing and pre-production in an efficient manner. There are two options for the container orchestration layer as illustrated in Figure 13.

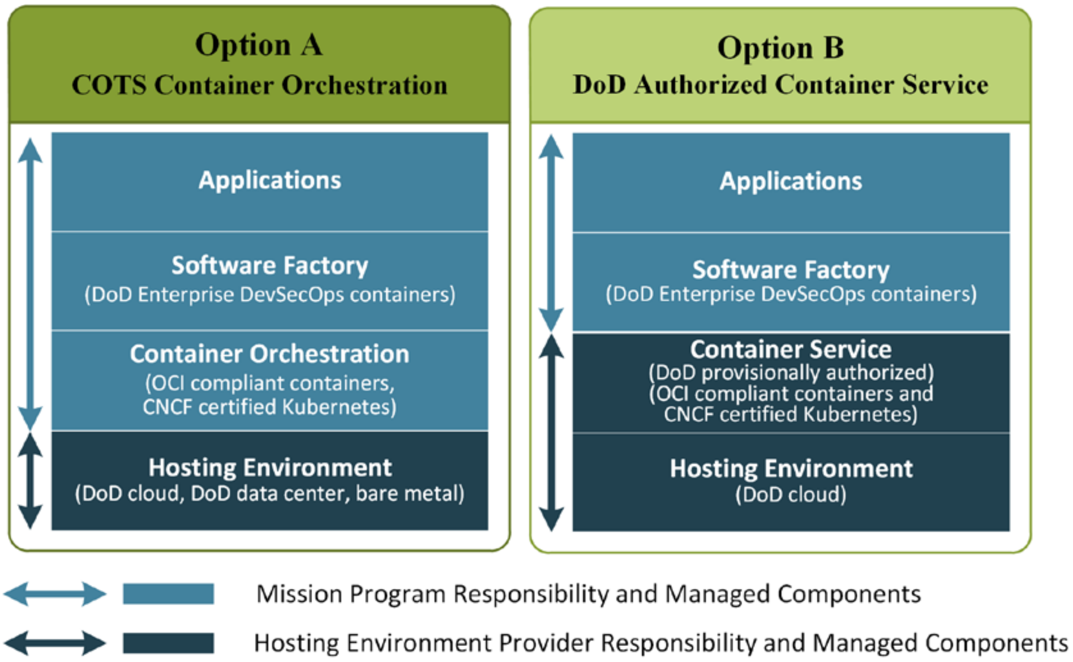

Option A
COTS Container Orchestration
Applications
Software Factory
(DoD Enterprise DevSecOps containers)
Container Orchestration
(OCI compliant containers,
CNCF certified Kubernetes)
Hosting Environment
(DoD cloud, DoD data center, bare metal)
Option B
DoD Authorized Container Service
Applications
Software Factory
(DoD Enterprise DevSecOps containers)
Container Service
(DoD provisionally authorized)
(OCI compliant containers and
CNCF certified Kubernetes)
Hosting Environment
(DoD cloud)
Mission Program Responsibility and Managed Components
Hosting Environment Provider Responsibility and Managed Components

Figure 13: DevSecOps Platform Options

In Option A**,** it is the mission program's responsibility to build and maintain the container orchestration layer (CNCF-certified Kubernetes) 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) [2] and DISA's Secure Cloud Computing Architecture (SCCA) [3] for the cloud environment.

In Option B**,** the mission program uses a CSP container service, which must have a DoD provisional authorization and must be based on CNCF-certified Kubernetes.

6.1.3 Software Factory Using Hardened Containers

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 its 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.

Software factory building itself follows the DevSecOps philosophy and goes through its own design, instantiate, verify, operate and monitor phases. It evolves through the application lifecycle iteration. Figure 14 illustrates the software factory phases, activities, and the relationship with the application lifecycle. Security must be applied across the software factory phases. Sidecar Container Security Stack (SCSS) discussed in 6.4.4 can be used for software factory runtime cybersecurity monitoring.

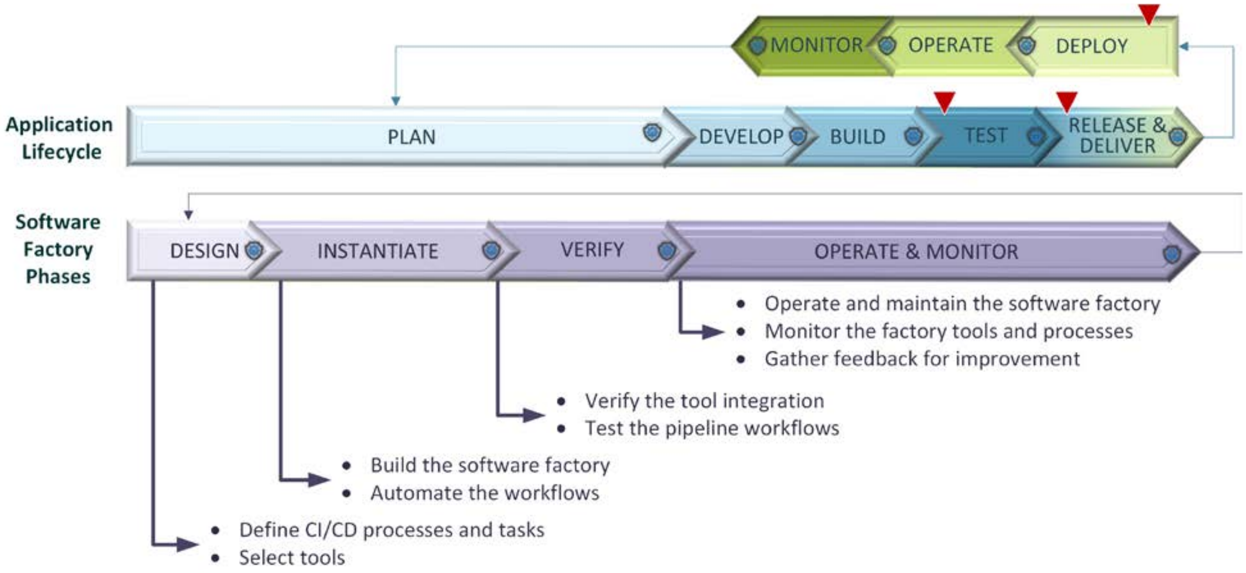

MONITOR
OPERATE
DEPLOY
Application Lifecycle
PLAN
DEVELOP
BUILD
TEST
RELEASE & DELIVER
Software Factory Phases
DESIGN
INSTANTIATE
VERIFY
OPERATE & MONITOR
Operate and maintain the software factory
Monitor the factory tools and processes
Gather feedback for improvement
Verify the tool integration
Test the pipeline workflows
Build the software factory
Automate the workflows
Define CI/CD processes and tasks
Select tools

Figure 14: Software Factory Phases in the Application Lifecycle

Figure 12 is a software factory reference design. It includes the tools and process workflows to develop, build, test, secure, release, and deliver software application for production deployment. All the tools are based on the DoD enterprise DevSecOps hardened containers. Committing code into the code repository kicks off the automated factory CI/CD pipeline workflow. The CI/CD orchestrator executes the workflow by coordinating different tools to perform various tasks. Some tasks are completed by a set of DevSecOps tools, such as build, static code analysis, unit test, publish artifacts, build container image, etc. Other tasks may need assistance from underlying container orchestration layer, such as deploy application to test, pre-production, and final production environments. Some test and security tasks may need human involvement.

6.1.4 DoD Applications

The term “DoD Application” refers to a DoD software program hosted by an information system [16], which spreads widely from legacy monolithic infrastructure-dependent applications to modern modular infrastructure-agnostic applications. Most systems are in brownfield with legacy applications or mixed legacy and modern applications. Programs should consider the nature of their application and the deployment environment when designing their software factory. It is recommended to leverage the Strangler Pattern [17] [18] to refactor legacy applications to modern microservices/containerized applications.

- DoD application environments at all phases (development, test, pre-production, production) are subject to security control from DoD common security services.
- While refactoring legacy code or writing new code, it is highly recommended to leverage the DoD hardened containers or hardening scripts to facilitate the application’s ATO.

6.2 Software Factory using Cloud DevSecOps Services

The DoD authorized cloud may already or will soon offer DevSecOps services, such as code repository, artifact repository, build service, code deploy service, etc. Programs should consider using these native managed services as alternatives to self-built and self-maintained DevSecOps tool sets, but they should understand that using them may lead to vendor lock-in with the CSP. The CSP is responsible for maintaining the service offering and the DoD Provisional Authorization (PA) for each service. The program still needs to follow the software factory lifecycle and performs the design, DevSecOps service selection instead of tool selection, CI/CD pipeline process workflow automation, verify the workflows, operates and monitors the workflows.

Another consideration is that the CSP may not offer a full solution set. For the capability that a DevSecOps service with a DoD PA is not available, the corresponding DoD Enterprise hardened container that provides the proper capability should be used. A CNCF-certified Kubernetes container orchestration service is required for container runtime. Figure 15: Software Factory illustrates a software factory using both cloud DevSecOps services and self-maintained security tools.

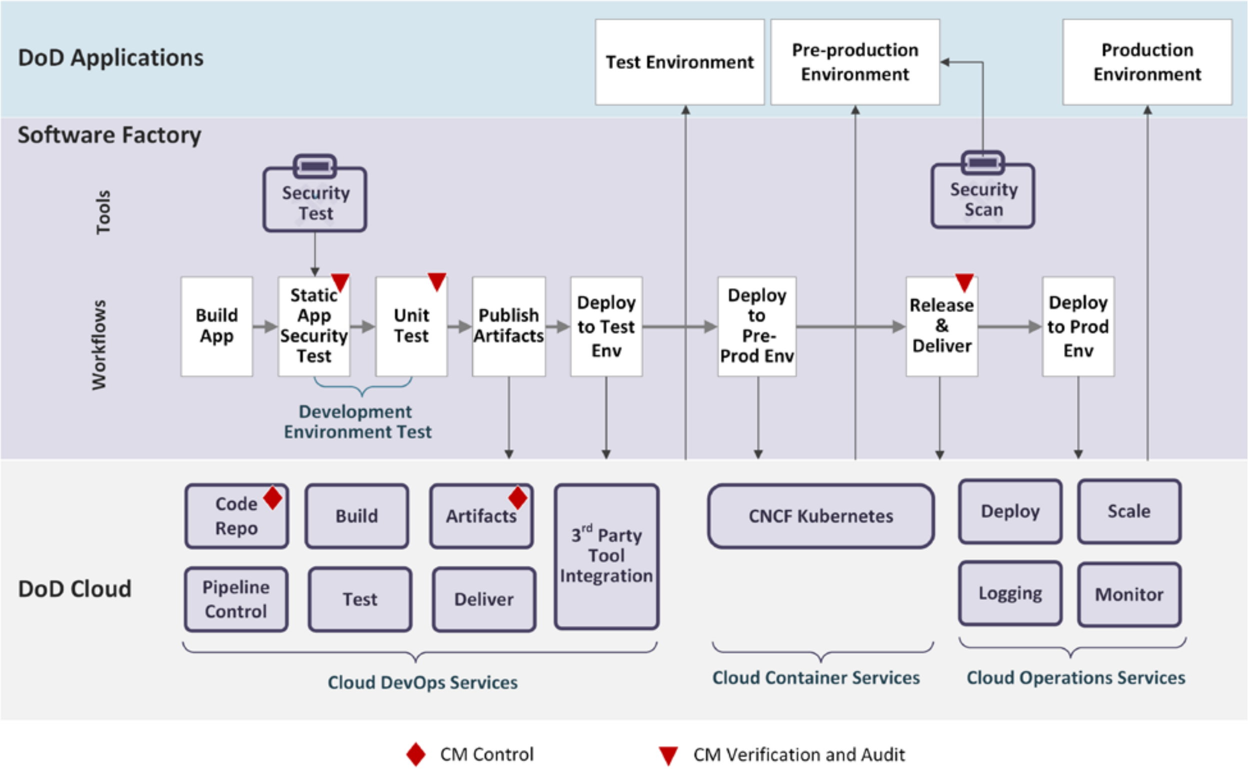


Figure 15: Software Factory using Cloud DevSecOps Services

6.3 Serverless Support

So-called “serverless computing” is becoming more popular in the DoD, and it is being used extensively in industry. This is a kind of Platform as a Service (PaaS) that is sometimes called Function as a Service (FaaS). Despite the “serverless” moniker, a FaaS still needs servers, but developers don’t have to worry about the servers, but rather how to deploy the code to them, how to set up autoscaling, and other deployment tasks. This frees the developers to focus on the code.

Figure 16 illustrates that a FaaS can reduce development complexity and increase efficiency over an Infrastructure as a Service (IaaS), a Containers as a Service (CaaS), or some Platform as a Service (PaaS) offerings. Although a Software as a Service (SaaS) offering is even more efficient, and good to use when it meets requirements, a SaaS is typically focused on only a few capabilities, and cannot provide the flexibility the DoD needs to develop custom applications. A good FaaS, on the other hand, does provide that flexibility, although some applications will need the even greater flexibility of an IaaS.

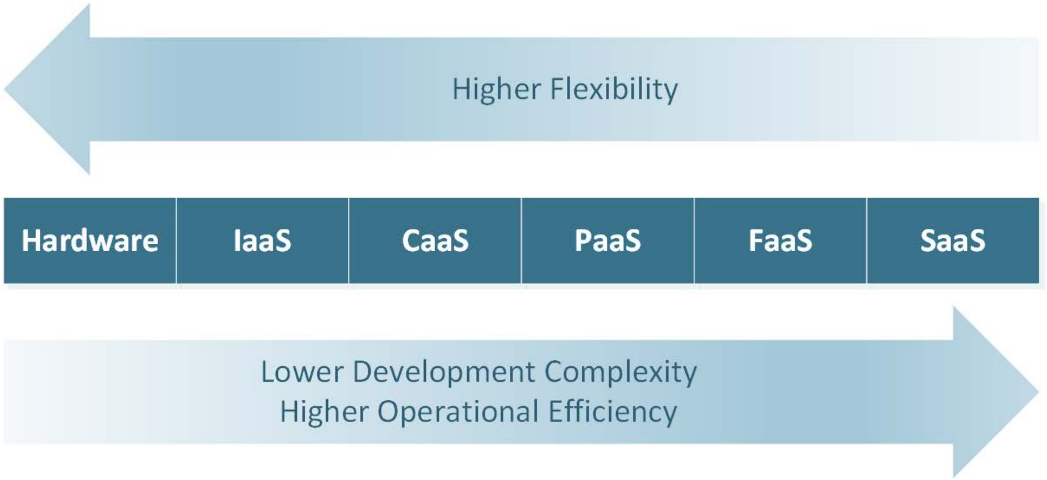


Figure 16: Operational Efficiency

The FaaS concept has made its way into the Kubernetes environment. The basic concept is to hand the FaaS some code, then the FaaS will build an image from the code and start it running on Kubernetes.

The FaaS must have these features:

1) **Build** – builds containers from source code
   a) Uses container images as the deployment unit
   b) Given source code, build the code and create a container to house it
2) **Serving** – runs the containers created by Build and automatically scales them up or down as necessary
   a) Uses CNCF Kubernetes as the underlying container orchestration layer
   b) Auto-scale up (given that the code is written to allow this)
   c) Auto-scale down all the way to zero
   d) Gradual rollouts of new versions
   e) Network routing within the cluster, and ingress connections into the cluster
3) **Eventing** – allows functions/applications to publish and subscribe to event streams, to enable loosely-coupled, event-driven systems
   a) Universal subscription, delivery, and management of events
   b) Bind events to functions or containers
   c) Trigger functions when called via and Hypertext Transfer Protocol (HTTP) requests
   d) Automatically scale from a few events per day to live streams

One popular open source product that implements FaaS for Kubernetes is Knative. Another open source product is Kubeless. The DevSecOps Software Factories must offer Knative support. They may also support Kubeless or another FaaS for Kubernetes.

6.4 Application Security Operations

This section focuses on the software application lifecycle in the production environment. *Continuous Deployment*, *Continuous Operation*, and *Continuous Monitoring* are keys to streamlined and secure operations.

6.4.1 Continuous Deployment

Continuous deployment is triggered by the successful delivery of released artifacts to the artifact repository and may be subject to control with human intervention according to the nature of the program application. The typical activities for continuous deployment include, but are not limited to, deploying a new software release to the production environment, applying necessary infrastructure and security configuration changes, running a smoke test to make sure essential functionality is working, and performing security scans. Each activity is completed by specific tools or configuration/orchestration scripts. The selection of tools and the configuration/orchestration system depends on the application and the production platform. For example, if the application is containerized and the production platform uses Kubernetes, the orchestration is done by Kubernetes. In this case, the configuration scripts would be Kubernetes Operators or Helm charts. On the other hand, if the application is VM based, the configuration/orchestration tool could be Chef, Puppet or Ansible.

Continuous deployment interacts with other DevSecOps components, such as the artifact repository for retrieving new releases, the log storage and retrieval service for logging deployment events, and the issue tracking system for recording any deployment issues. The first-time deployment may involve heavy infrastructure provisioning, dependency system configuration (such as monitoring tools, logging tools, scanning tools, backup tools, etc.), and external system connectivity (such as DoD common security services, etc.).

6.4.2 Continuous Operation

Continuous operation is an extension of continuous deployment. It is triggered by a successful deployment to the production environment, so that it operates the latest stable software release. The activities of continuous operation include, but are not limited to, system patching, compliance scanning, data backup, system recovery if failure happens, and resource optimization with load balancing and scaling. The selection of the tools that facilitate the activities are application and environment dependent. The resource optimization heavily depends on the underlying platform. A containerized application can rely on Kubernetes to automatically scale containers across cluster nodes. On the other hand, a VM-based application with no containers can rely on the underlying CSP’s scaling service.

Continuous operation interacts with the logging system, issue tracking system, and the underlying infrastructure platform.

6.4.3 Continuous Monitoring

Continuous monitoring is an extension to continuous operation. It continuously inventories all system components, monitors the performance and security of all components, and logs application and system events. Figure 20 in Section 6.4.4 illustrates the components for monitoring a containerized application deployed on Kubernetes. Figure 17 illustrates a simplified sample process of monitoring, logging, and log analysis and alerting, which also applies to deployments to non-containerized environments.

The process starts with application logging, compute resource monitoring, storage monitoring, network monitoring, security monitoring, and data monitoring at the Kubernetes pod level in the case of containerized deployment or individual subsystem level in the case of VM deployment. 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 (e.g., the Sidecar Container Security Stack) will aggregate and forward the event logs gathered from monitoring to a locally centralized aggregated logs database on the mission program platform. This should be automated within the Kubernetes cluster. The aggregated logs will be further forwarded to the Logs/Telemetry Analysis in the DoD Common Security Services after passing the program application configured log filter. The program's local log 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 back to planning.

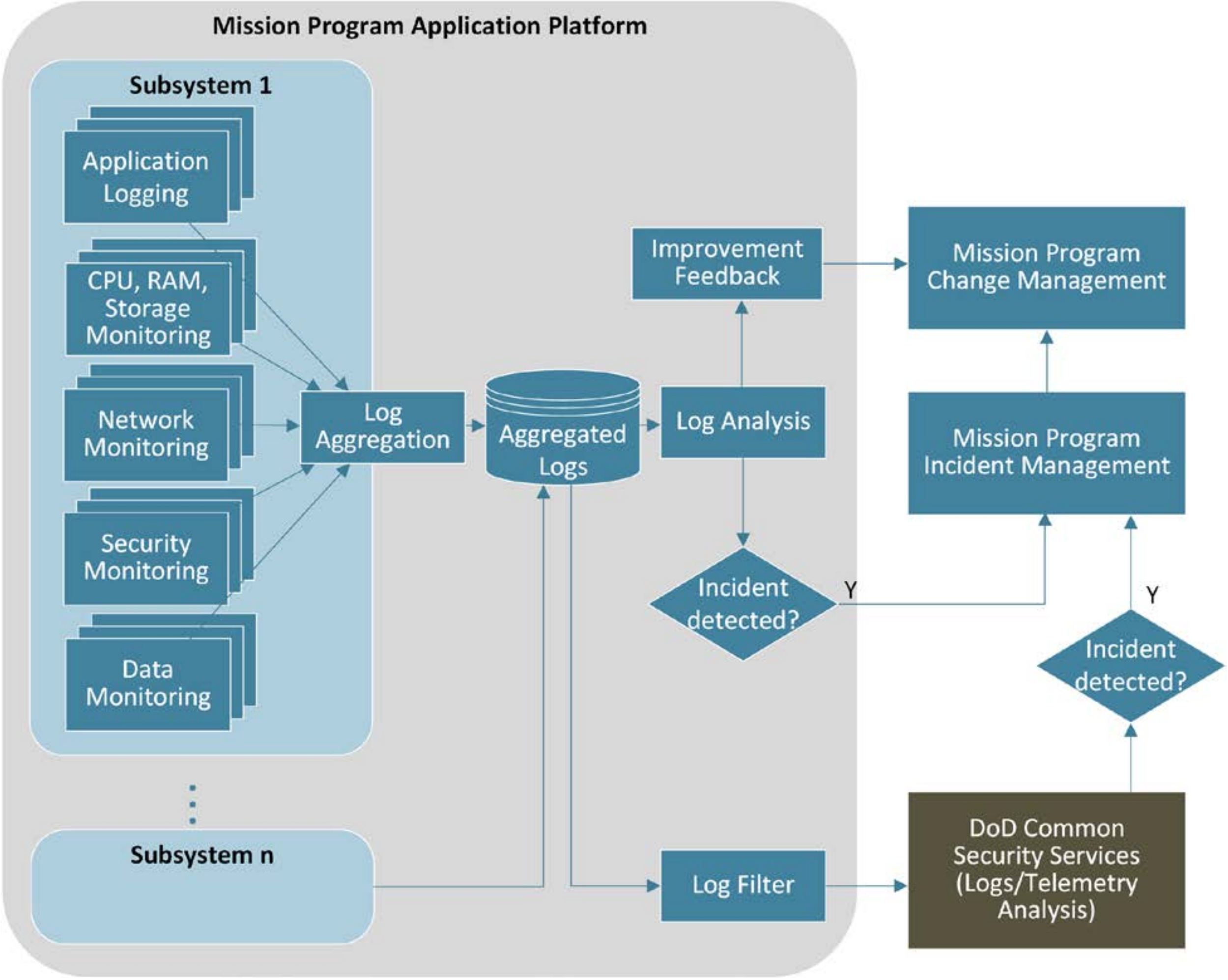


Figure 17: Logging and Log Analysis Process

6.4.4 Sidecar Container Security Stack

A new service that is enabled by DevSecOps and the container-based Kubernetes runtime environment is the Sidecar Container Security Stack (SCSS). This security stack enables: correlated and centralized logs, container security, east/west traffic management, a zero-trust model, a whitelist, Role-Based Access Control (RBAC), continuous monitoring, signature-based continuous scanning using Common Vulnerabilities and Exposures (CVEs), runtime behavior analysis, and container policy enforcement.

One advantage of using the SCSS is that Kubernetes can inject the sidecar automatically, without the team having to do anything, once it's configured.

The sidecar pattern is depicted in Figure 18. A **container group** or **pod** is a set of containers that are deployed together. A sidecar is a container running inside a pod alongside an application container. If there is one application container and one sidecar container in the container group, then we have the sidecar pattern as depicted in Figure 18.

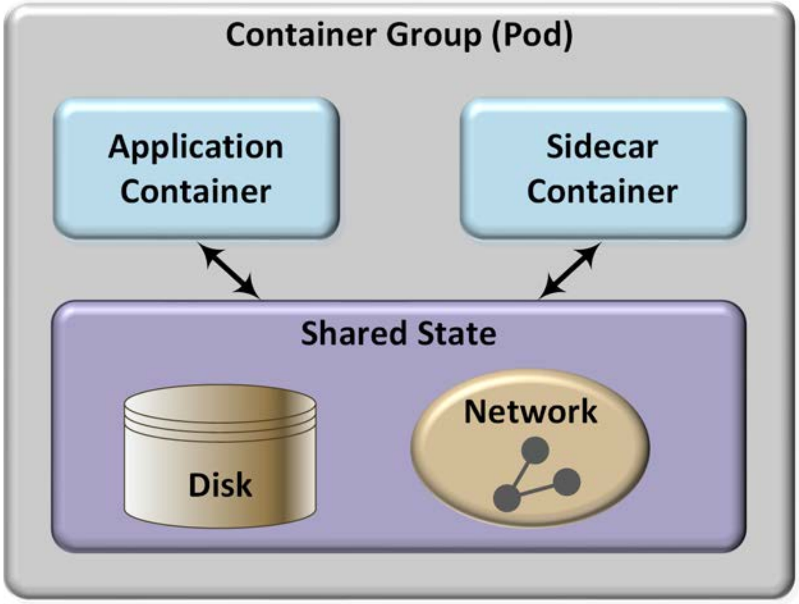


Figure 18: Sidecar Pattern

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 isolated from one another.

A sidecar is a general container pattern. The Sidecar Container Security Stack has a sidecar container that contains a security stack, along with some supporting services that run in the hosting environment, such as a logging service. The security stack in the security sidecar container will include:

1. A logging agent to push logs to a platform centralized logging service.
2. Container policy enforcement. This includes ensuring container hardening from DCAR containers are preserved and complies with the NIST 800-190 requirements [12].
3. Runtime Defense, this can perform both signature-based and behavior-based detection. This can also be used to send notifications when there is anomalous behavior.
4. Vulnerability Management
5. A service mesh proxy to connect to the service mesh
6. Zero Trust down to the container level. Zero trust requires strict controls, never trust anything by default and always verify. Key aspects of zero trust at the container level include mutual Transport Layer Security authentication (mTLS), an encrypted communication tunnel between containers, strong identities per Pod using certificates, and whitelisting rather than blacklisting.

In addition to the components in the sidecar, there are a few services that support the security sidecar. These include:

1. Program-specific Log Storage and Retrieval Service
2. Service Mesh
3. Program-specific artifact repository
4. Runtime Behavior Analysis Artificial Intelligence (AI) service
5. DCAR for the hardened containers
6. Common Vulnerabilities and Exposures (CVE)Service / host-based security to provide CVEs for the security sidecar container

The interaction of these services with the sidecar components is depicted in Figure 19. The arrows show the direction of the data flow. The items in purple are services provided by the DoD, while blue indicates components that are provided by the DoD but instantiated and operated by the program.

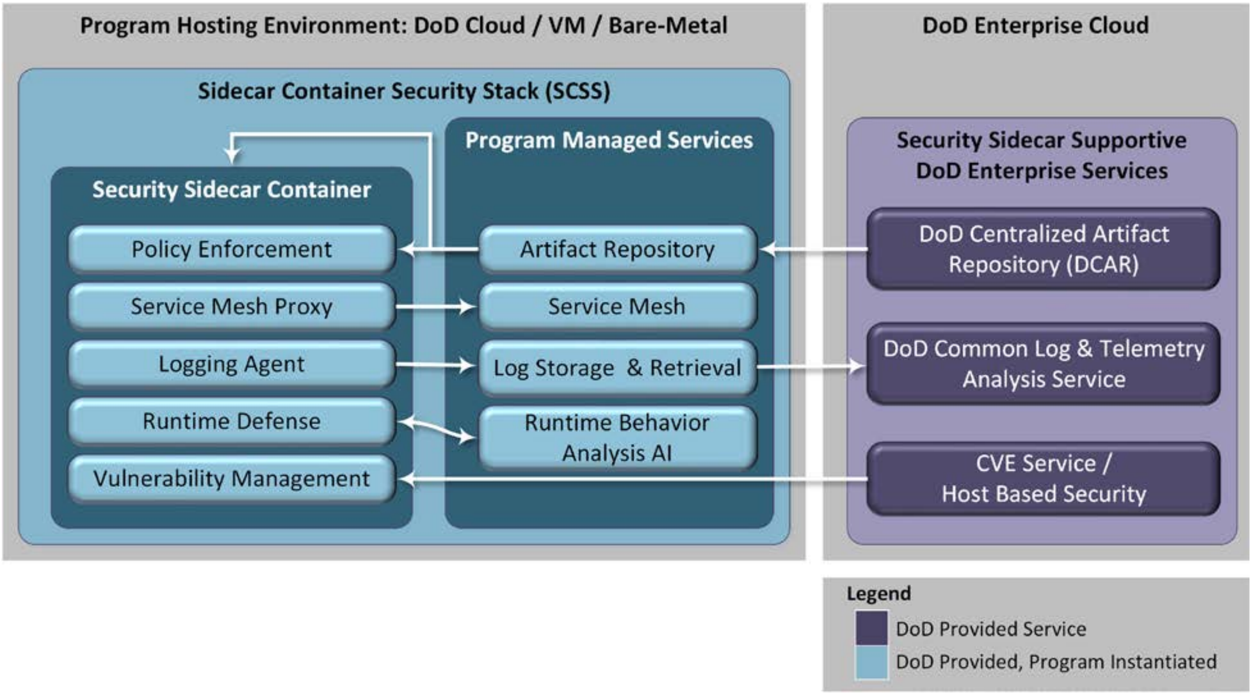

Program Hosting Environment: DoD Cloud / VM / Bare-Metal
Sidecar Container Security Stack (SCSS)
Security Sidecar Container
Policy Enforcement
Service Mesh Proxy
Logging Agent
Runtime Defense
Vulnerability Management
Program Managed Services
Artifact Repository
Service Mesh
Log Storage & Retrieval
Runtime Behavior Analysis AI
DoD Enterprise Cloud
Security Sidecar Supportive DoD Enterprise Services
DoD Centralized Artifact Repository (DCAR)
DoD Common Log & Telemetry Analysis Service
CVE Service / Host Based Security
Legend
🟪 DoD Provided Service
🟦 DoD Provided, Program Instantiated

Figure 19: Sidecar Components

Table 23: Sidecar Container Security Stack Components

Tool	Features	Benefits	Baseline
Logging agent	Send logs to a logging service	Standardize log collection to a central location. This can also be used to send notifications when there is anomalous behavior.	MVP
Logging Storage and Retrieval Service	Stores logs and allows searching logs	Place to store logs	MVP

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

Figure 20 depicts another view of the sidecar, along with some of the other DevSecOps components. Again, the arrows show the direction of the data flow, and not all interactions between the depicted components are indicated. The program dashboard displays information about the application. The dashboard is built partly using visualizations from the log visualization service. The items in purple and blue are either services or components that are provide by the DoD. But those in blue will be stood up by the program. For example, the Service Mesh is provided as a hardened container. Similarly, the Sidecar Container Security Stack is provided as a hardened container that the program installs in each pod (container group); this is injected by Kubernetes automatically without application developer involvement.

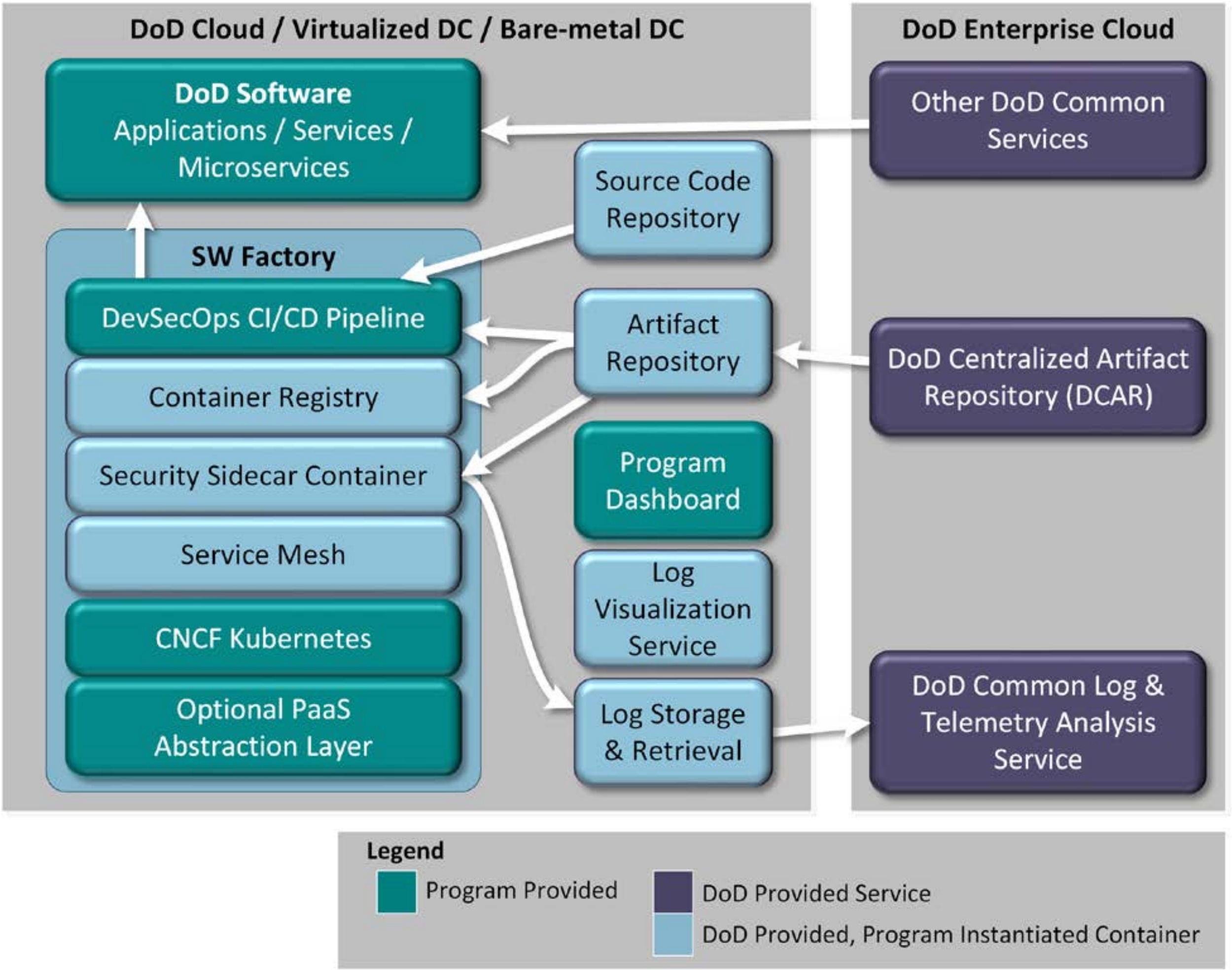

DoD Cloud / Virtualized DC / Bare-metal DC
DoD Enterprise Cloud
DoD Software
Applications / Services / Microservices
Other DoD Common Services
Source Code Repository
SW Factory
DevSecOps CI/CD Pipeline
Artifact Repository
DoD Centralized Artifact Repository (DCAR)
Container Registry
Security Sidecar Container
Program Dashboard
Service Mesh
Log Visualization Service
CNCF Kubernetes
Optional PaaS Abstraction Layer
Log Storage & Retrieval
DoD Common Log & Telemetry Analysis Service
Legend
Program Provided
DoD Provided Service
DoD Provided, Program Instantiated Container

Figure 20: Sidecar Container Security Stack Interactions

7 Conclusion

We have introduced key DevSecOps concepts, described the DevSecOps Ecosystem, including the Software Factory and the Sidecar Container Security Stack, and indicated how the ecosystem should be set up and used. More detail on the components described here, such as DCAR, can be found in other DoD CIO documents.

Moving to DevSecOps improves agility and speeds new capabilities into the field. But it also requires new policies, processes and culture change. More information on DevSecOps culture, metrics, and the maturity model can be found in the DoD DevSecOps Playbook [7].

Appendix A Acronym Table

Acronym	Definition
A&A	Assessment and Authorization
A&S	Acquisition and Sustainment
AI	Artificial Intelligence
AO	Authorizing Official
API	Application Programming Interface
AQ	Acquisition
ASW	Attack Sensing and Warning
ATC	Approval to Connect (ATC)
ATO	Authority to Operate (ATO)
AVM	Assurance Vulnerability Management
BOM	Bill of Materials
CaaS	Containers as a Service
CCB	Change Control Board
CD	Continuous Delivery
CI	Continuous Integration
CIO	Chief Information Officer
CM	Configuration Management
CMDB	Configuration Management Data Base
CNCF	Cloud Native Computing Foundation
CNSS	Committee on National Security Systems
CNSSI	Committee on National Security Systems Instruction
COTS	Commercial Off The Shelf
CPCON	Cyber Protection Condition
CPU	Central Processing Unit
CSP	Cloud Service Provider
CSSP	Cybersecurity Service Provider
CVE	Common Vulnerabilities and Exposures
DAST	Dynamic Application Security Test
DCAR	DoD Centralized Artifact Repository
DCCSCR	DoD Centralized Container Source Code Repository
DCIO	Deputy Chief Information Officer
DBaaS	Database as a Service
DDOS	Distributed Denial of Service
DevSecOps	Development, Security, and Operations
DISA	Defense Information Systems Agency

Acronym	Definition
DNS	Domain Name Service
DoD	Department of Defense
DoDI	DoD Instruction
DTRA	Defense Threat Reduction Agency
EO	Executive Order
FaaS	Function as a Service
FMA	Forensic Media Analysis
GB	Gigabyte
GEP	Global Enterprise Partners
GOTS	Government Off The Shelf
HTTP	Hypertext Transfer Protocol
IA	Information Assurance
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAST	Interactive Application Security Test
ICAM	Identity, Credential, and Access Management
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IE	Information Enterprise
IHR	Incident Handling Response
INFOCON	Information Operations Condition
IO	Input/Output
IPS	Intrusion Prevention System
IR	Incident Reporting
ISCM	Information Security Continuous Monitoring
IT	Information Technology
JVM	Java Virtual Machine
KPI	Key Performance Indicator
MB	Megabyte
MilDep	Military Department
MNP	Malware Notification Protection
mTLS	mutual Transport Layer Security authentication
MVP	Minimum Viable Product
NGG	Next Generation Governance
NIST	National Institute of Standards and Technology
NoSQL	Non SQL
NSM	Network Security Monitoring

Acronym	Definition
OCI	Open Container Initiative
OS	Operating System
PA	Provisional Authorization
PaaS	Platform as a Service
PEO	Program executive Officer
POA&M	Plan of Action and Milestones
QA	Quality Assurance
RBAC	Role-Based Access Control
RMF	Risk Management Framework
ROI	Return on Investment
SaaS	Software as a Service
SaC	Security as Code
SAF	Secretary of the Air Force
SAST	Static Application Security Test
SCAP	Security Content Automation Protocol
SCCA	Secure Cloud Computing Architecture
SCSS	Sidecar Container Security Stack
SLA	Service Level Agreement
SRG	Security Requirements Guide
SQL	Structured Query Language
SSH	Secure Shell
STIG	Security Technical Implementation Guide
TRB	Technical Review Board
VM	Virtual Machine

Appendix B Glossary of Key Terms

Following are the key terms used in describing the reference design in this document.

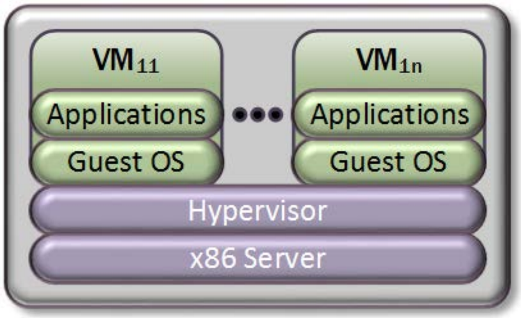
Term	Definition
Artifact **Software Artifact**	An artifact is a consumable piece of software produced during the software development process. Except for interpreted languages, the artifact is or contains compiled software. Important examples of artifacts include container images, virtual machine images, binary executables, jar files, test scripts, test results, security scan results, configuration scripts, Infrastructure as a Code, documentation, etc. Artifacts are usually accompanied by metadata, such as an id, version, name, license, dependencies, build date and time, etc. Note that items such as source code, test scripts, configuration scripts, build scripts, and Infrastructure as Code are checked into the source code repository, not the artifact repository, and are not considered artifacts.
Artifact Repository	An artifact repository is a system for storage, retrieval, and management of artifacts and their associated metadata. Note that programs may have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the content types.
Bare Metal **Bare Metal Server**	A bare metal or bare metal server refers to a traditional physical computer server that is dedicated to a single tenant and which does not run a hypervisor. This term is used to distinguish physical compute resources from modern forms of virtualization and cloud hosting.
Binary **Binary File**	Binary refers to a data file or computer executable file that is stored in binary format (as opposed to text), which is computer-readable, but not human-readable. Examples include images, audio/video files, exe files, and jar/war/ear files.
Build **Software Build**	The process of creating a set of executable code that is produced by compiling source code and linking binary code.

Build tools **Software Build Tools**	Used to retrieve software source code, build software, and generate artifacts.
CI/CD Orchestrator	CI/CD orchestrator is a tool that enables fully or semi-automated short duration software development cycles through integration of build, test, secure, store artifacts tools. CI/CD orchestrator is the central automation engine of the CI/CD pipeline
CI/CD Pipeline	CI/CD pipeline is the set of tools and the associated process workflows to achieve continuous integration and continuous delivery with build, test, security, and release delivery activities, which are steered by a CI/CD orchestrator and automated as much as practice allows.
CI/CD Pipeline Instance	CI/CD pipeline instance is a single process workflow and the tools to execute the workflow for a specific software language and application type for a project. As much of the pipeline process is automated as is practicable.
Cloud Native Computing Foundation (CNCF)	“CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.” - https://www.cncf.io/
CNCF-certified Kubernetes	CNCF has created a Certified Kubernetes Conformance Program. Software conformance ensures that every vendor’s version of Kubernetes supports the required APIs. Conformance guarantees interoperability between Kubernetes from different vendors. Most of the world’s leading vendors and cloud computing providers have CNCF certified Kubernetes offerings.
Code	Software instructions for a computer, written in a programming language. These instructions may be in the form of either human-readable source code, or machine code, which is source code that has been compiled into machine executable instructions.

Configuration Management	Capability to establish and maintain a specific configuration within operating systems and applications.
Container	A standard unit of software that packages up code and all its dependencies, down to, but not including the OS. It is a lightweight, standalone, executable package of software that includes everything needed to run an application except the OS: code, runtime, system tools, system libraries and settings.
Continuous Build	Continuous build is an automated process to compile and build software source code into artifacts. The common activities in the continuous build process include compiling code, running static code analysis such as code style checking, binary linking (in the case of languages such as C++), and executing unit tests. The outputs from continuous build process are build results, build reports (e.g., the unit test report, and a static code analysis report), and artifacts stored into Artifact Repository. The trigger to this process could be a developer code commit or a code merge of a branch into the main trunk.
Continuous Delivery	Continuous delivery is an extension of continuous integration to ensure that a team can release the software changes to production quickly and in a sustainable way. The additional activities involved in continuous integration include release control gate validation and storing the artifacts in the artifact repository, which may be different than the build artifact repository. The trigger to these additional activities is successful integration, which means all automation tests and security scans have been passed. The human input from the manual test and security activities should be included in the release control gate. The outputs of continuous delivery are a release go/no-go decision and released artifacts, if the decision is to release.
Continuous Deployment	Continuous deployment is an extension of continuous delivery. It is triggered by a successful delivery of released artifacts to the artifact repository.

	The additional activities for continuous deployment include, but are not limited to, deploying a new release to the production environment, running a smoke test to make sure essential functionality is working, and a security scan. The output of continuous deployment includes the deployment status. In the case of a successful deployment, it also provides a new software release running in production. On the other hand, a failed deployment causes a rollback to the previous release.
Continuous Integration	Continuous integration goes one step further than continuous build. It extends continuous build with more automated tests and security scans. Any test or security activities that require human intervention can be managed by separate process flows. The automated tests include, but are not limited to, integration tests, a system test, and regression tests. The security scans include, but are not limited to, dynamic code analysis, test coverage, dependency/BOM checking, and compliance checking. The outputs from continuous integration include the continuous build outputs, plus automation test results and security scan results. The trigger to the automated tests and security scan is a successful build.
Continuous monitoring	Continuous monitoring is an extension to continuous operation. It continuously monitors and inventories all system components, monitors the performance and security of all the components, and audits & logs the system events.
Continuous Operation	Continuous operation is an extension to continuous deployment. It is triggered by a successful deployment. The production environment operates continuously with the latest stable software release. The activities of continuous operation include, but are not limited to: system patching, compliance scanning, data backup, and resource optimization with load balancing and scaling (both horizontal and vertical).
Cybersecurity, **Software Cybersecurity**	The preventative methods used to protect software from threats, weaknesses and vulnerabilities.

DoD Centralized Artifact Repository (DCAR)	Holds the hardened container images of DevSecOps components that DoD mission software teams can utilize to instantiate their own DevSecOps pipeline. It also holds the hardened containers for base operating systems, web servers, application servers, databases, API gateways, message busses for use by DoD mission software teams as a mission system deployment baseline. These hardened containers, along with security accreditation reciprocity, greatly simplifies and speeds the process of obtaining an Approval to Connect (ATC) or Authority to Operate (ATO).
Delivery	The process by which a released software is placed into a artifact repository that operational environment can download.
Deployment	The process by which the released software is downloaded and deployed to the production environment.
DevSecOps	DevSecOps is a software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of software development: plan, develop, build, test, release, deliver, deploy, operate, and monitor.
DevSecOps Ecosystem	A collection of tools and process workflows created and executed on the tools to support all the activities throughout the full DevSecOps lifecycle. The process workflows may be fully automated, semi-automated, or manual.
DevSecOps Pipeline	DevSecOps pipeline is a collection of DevSecOps tools, upon which the DevSecOps process workflows can be created and executed.
DevSecOps phase	The software development, security, and operation activities in the software lifecycle are divided into phases. Each phase completes a part of related activities using tools.
Environment	Sets a runtime boundary for the software component to be deployed and executed. Typical environments include development, integration, test, pre-production, and production.

Factory, **Software Factory**	A software assembly plant that contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The software factory supports multi-tenancy.
Software Factory Artifact Repository	Stores artifacts pulled from DCAR as well as locally developed artifacts to be used in DevSecOps processes. The artifacts include, but are not limited to, VM images, container images, binary executables, archives, and documentation. It supports multi-tenancy. Note that program could have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the contents.
Hypervisor	A hypervisor is a kind of low-level software that creates and runs virtual machines (VMs). Each VM has its own Operating System (OS). Several VMs can run on one physical machine.  VM$_{11}$ VM$_{1n}$ Applications Applications Guest OS Guest OS Hypervisor x86 Server Figure 21: Hypervisor with Virtual Machines
Image Management, **Software Image Management,** **Binary Image Management,** **Container Image Management,** **VM Image Management**	The process of centralizing, organizing, distributing, and tracking of software artifacts.
Immutable infrastructure	An infrastructure paradigm in which servers are never modified after they're deployed. If something needs to be updated, fixed, or

	modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they're validated, they're put into use and the old ones are decommissioned. The benefits of an immutable infrastructure include more consistency and reliability in your infrastructure and a simpler, more predictable deployment process. (from: https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure)
Infrastructure as Code	The management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning that the DevSecOps team uses for source code. Infrastructure as Code evolved to solve the problem of environment drift in the release pipeline.
Kubernetes	An open-source system for automating deployment, scaling, and management of containerized applications. It was originally designed by Google and is now maintained by the CNCF. Many vendors also provide their own branded Kubernetes. It works with a range of container runtimes. Many cloud services offer a Kubernetes-based platform as a service.
Lockdown	The closing or removal of weaknesses and vulnerabilities from software.
Microservices	Microservices are both an architecture and an approach to software development in which a monolith application is broken down into a suite of loosely coupled independent services that can be altered, updated, or taken down without affecting the rest of the application.
Mission Application Platform	The mission application platform is the underlying hosting environment resources and capabilities, plus any mission program enhanced capabilities that form the base upon which the mission software application operates.
Monitoring **Security Monitoring**	The regular observation, recording, and presentation of activities.

Node **Cluster node**	A node is a worker machine in CNCF Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components, including the node controller.
OCI	"An open governance structure for the express purpose of creating open industry standards around container formats and runtime" - https://www.opencontainers.org/
OCI Compliant Container	The image of the OCI compliant container must conform with the OCI Image Specification.
OCI Compliant Container Runtime	A container runtime is software that executes containers and manages container images on a node. OCI compliant container runtime must conform with the OCI Runtime Specification.
Orchestration	Automated configuration, coordination, and management.
Platform	A platform is a group of resources and capabilities that form a base upon which other capabilities or services are built and operated.
Pod	A group of containers that run on the same CNCF Kubernetes worker node and share libraries and OS resources.
Provisioning	Instantiation, configuration, and management of software or the environments that host or contain software.
Reporting	An account or statement describing an event.
Repository	A central place in which data is aggregated and maintained in an organized way.
Resource	CPU, Memory, Disk, Networking
Scanning, **Security Scanning**	The evaluation of software for cybersecurity weaknesses and vulnerabilities.
Sidecar Container Security Stack	A sidecar container security stack is a stack of sidecar containers aimed to enhance the security capabilities of the main containers in the same Pod.

Sidecar	A sidecar is a container used to extend or enhance the functionality of an application container without strong coupling between two. When using CNCF Kubernetes, a pod is composed of one or more containers. A sidecar is a utility container in the pod and its purpose is to support the main application container or containers inside the same pod. For more information, see Section 6.4.4 and Kubernetes documentation.
Telemetry	Capability to take measurements and collect and distribute the data.
Test Coverage, **Code Coverage**	Test coverage is a measure used to describe what percentage of application code is exercised when a test suite runs. A higher percentage indicates more source code executed during testing, which suggests a lower chance of containing undetected bugs.
Virtual Machine (VM)	Emulates a physical computer in software. Several VMs can run on the same physical device.
Virtual Network	Networks constructed of software-defined devices.
Virtual Storage	Storage constructed of software-defined devices.

Appendix C References

The following references are utilized in the development of the DevSecOps reference design document:

[1] Department of Defense, "DoD Cloud Computing Strategy," December 2018.

[2] DISA, "Department Of Defense Cloud Computing Security Requirements Guide, V1R2," 18 March, 2016.

[3] DISA, "DoD Secure Cloud Computing Architecture (SCCA) Functional Requirements," January 31, 2017.

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

[5] National Institute of Standards and Technology, Framework for Improving Critical Infrastructure Cybersecurity, 2018.

[6] DISA, "Department Of Defense Container Hardening Security Requirements Guide (Draft)," 2019.

[7] Office of the DoD CIO, "DoD DevSecOps Playbook," 2019. [Online]. Available: https://www.milsuite.mil/book/groups/dod-enterprise-devsecops.

[8] Puppet Labs, "Puppet State of DevOps Report 2018," Puppet Labs, 2018.

[9] Defense Threat Reduction Agency (DTRA), "Next-Generation Technology Governance," 2018.

[10] DoD, "DoDI 8510.01: Risk Management Framework for DoD Information Technology," 24 May 2016.

[11] E. Ries, "The Lean Startup," [Online]. Available: http://theleanstartup.com/principles. [Accessed 15 July 2019].

[12] NIST, "NIST Special Publication 800-190, Application Container Security Guide," September 2017.

[13] N. M. Chaillan, "DoD Enterprise DevSecOps Initiative Hardening Containers," DRAFT, 2019.

[14] NIST, "Security and Privacy Controls for Federal Information Systems and Organizations," NIST SP 800-53 Revision 4, 2013.

[15] NIST, "Risk Management Framework for Information Systems and Organizations," NIST SP 800-37 Revision 2, 2018.

[16] Committee on National Security Systems (CNSS) , "Committee on National Security Systems Instruction (CNSSI) 4009: Committee on National Security Systems (CNSS) Glossary," 2015.

[17] M. Fowler, "Strangler Fig Application," 29 June 2004. [Online]. Available: https://martinfowler.com/bliki/StranglerFigApplication.html. [Accessed 12 July 2019].

[18] P. Hammant, "Legacy applicaion Strangulation: Case Studies," 14 July 2013. [Online]. Available: https://paulhammant.com/2013/07/14/legacy-application-strangulation-case-studies/. [Accessed 12 July 2019].