DoD Enterprise DevSecOps Fundamentals

October 2024
Version 2.5

This page is intentionally left blank.

Document Approval

Approved by the DoD Software Modernization Senior Steering Group on October 16, 2024.

Trademark Information

Foreword

The Department of Defense (DoD) is embracing innovative software approaches to *deliver resilient software capability at the speed of relevance*. A shift toward Agile and DevSecOps and away from traditional waterfall approaches has been underway and is rapidly accelerating. This shift is crucial for maintaining our lethal force and building flexibility, security, and resiliency in our software delivery practices.

Agile software methodology is the foundation of modern software development. Software developers frustrated with the pace and unpredictability of legacy processes realized there was a better way to deliver software faster and more predictably by breaking down silos, developing software incrementally, creating feedback loops, and quickly delivering a “minimum viable product” for fast feedback. The Agile revolution transformed commercial software development and led to a burst of innovation in automation and tools.

The acceleration of software development led to the discipline of DevOps that focused on the need to both develop (Dev) and transition software into operations (Ops).

DoD’s unique mission compels programs to raise security to an equal footing with development and operations. **DevSecOps** is a combination of software engineering methodologies, practices, and tools that unifies software development (Dev), security (Sec), and operations (Ops). It emphasizes collaboration across these disciplines, along with automation and continuous monitoring to support the delivery of secure, high-quality software. DevSecOps integrates security tools and practices into the development pipeline, emphasizes the automation of processes, and fosters a culture of shared responsibility for performance, security, and operational integrity throughout the entire software lifecycle, from development to deployment and beyond. The concepts build upon the modern technology trends of the past two decades:

- The shift from waterfall software development methodology to Agile
- The transition from tightly coupled monolithic systems to loosely coupled modular services
- Integration of security across the lifecycle of technology
- Incorporation of testing throughout the software lifecycle
- Evolution from traditional data centers to cloud

The **DoD Enterprise DevSecOps Fundamentals**, along with other supporting guidance available at https://dodcio.defense.gov/library/, provides education, best practices, and implementation and operational guidance for information technology (IT) capability providers, IT capability consumers, product teams, and Authorizing Officials (AO). It is intended to build a community that understands the realm of the possible and is motivated to pursue the possible to enable a warfighting force strengthened by software.

Contents

Figures

1 Introduction

The Department of Defense is on a multi-year journey to fundamentally transform how software is built, used, and managed across all aspects of the DOD enterprise. This includes business systems, weapons systems, embedded software, and essential command and control and warfighting support systems. This transformation is essential to maintain the technology advantage that underpins US military capability. The DoD Enterprise DevSecOps Fundamentals recognizes the importance of software. This document is an educational compendium of DevSecOps concepts and is intended to promote the adoption of modern software practices across the Department. Additional information is captured in corresponding guidance available at https://dodcio.defense.gov/library/.

Each DoD organization is expected to tailor its culture and align DevSecOps practices to their own unique processes, products, security requirements, and operational procedures, while leveraging DoD Enterprise resources as a first preference, and making reasonable efforts to ensure developed solutions are re-usable across DoD organizations. Embracing DevSecOps requires organizations to shift their culture, evolve existing processes, adopt new technologies, and strengthen governance.

The intended audience of this document includes novice and intermediate staff who have recently adopted or anticipate adopting DevSecOps as well as DoD managers and leadership who need to understand the software transformation underway and how it will impact their mission. Expert practitioners may find value in this material as a refresher.

The document is organized in the following manner:

Section 1.0: Introduction	Provides a brief introduction and includes assumptions related to the concepts of DevSecOps.
Section 2.0: Defining DevSecOps	Provides the definition of DevSecOps and a description of the DevSecOps lifecycle and phases.
Section 3.0: Implementing DevSecOps	Provides in-depth information regarding the components of DevSecOps to include DevSecOps integration with other processes.
Section 4.0: Institutionalizing DevSecOps	Provides guidance regarding a DevSecOps culture and metrics.
Section 5.0: Getting Started	Provides next steps and identifies additional resources to support a DevSecOps journey.

1.1 Assumptions

The DoD Enterprise DevSecOps Fundamentals makes the following assumptions:

- Agile/iterative software development practices have been implemented. The concepts in this document take those practices to the next step in maturity by implementing DevSecOps.
- Organizations deploying new business solutions or modernizing existing software systems will use an approved or provisionally authorized (PA) cloud environment as their preferred solution.

For weapons systems, the environment will leverage cloud along with on-premises capabilities that facilitate hardware-in-the-loop (HWIL) testing for embedded systems.

- Cybersecurity elements will leverage cloud service provider (CSP) managed service capabilities where practicable. Teams will aggressively seek to integrate automated feedback, patching, alerting, and other authorized network security measures.

- Rapidly changing technology dictates designing DevSecOps pipelines and patterns for flexibility as new development capabilities enter/exit the commercial product market.

- DoD Components acknowledge a lock-in posture and recognize that vendor lock-in as well as product, version, architecture, platform, skills, legal, and mental lock-in also exist. Avoiding vendor lock-in without considering other types of lock-in is ill-advised.

- DevSecOps may be leveraged for any type of operational requirement needing software capabilities.

2 Defining DevSecOps

2.1 Definition

DevSecOps is a combination of software engineering methodologies, practices, and tools that unifies software development (Dev), security (Sec), and operations (Ops).

DevSecOps emphasizes collaboration across these disciplines, along with automation and continuous monitoring to support the delivery of secure, high-quality software. DevSecOps integrates security tools and practices into the development pipeline, emphasizes the automation of processes, and fosters a culture of shared responsibility for performance, security, and operational integrity throughout the entire software lifecycle, from development to deployment and beyond. Figure 1 visually depicts the DevSecOps lifecycle as an iterative infinity loop divided into ten distinct phases.

DevSecOps is iterative by design, recognizing that *software is never done*. The serial-style delivery of the waterfall process is replaced with small, frequent deliveries that make it 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 continuous deployment. This lifecycle is adaptable and includes numerous feedback loops that drive continuous process improvements.

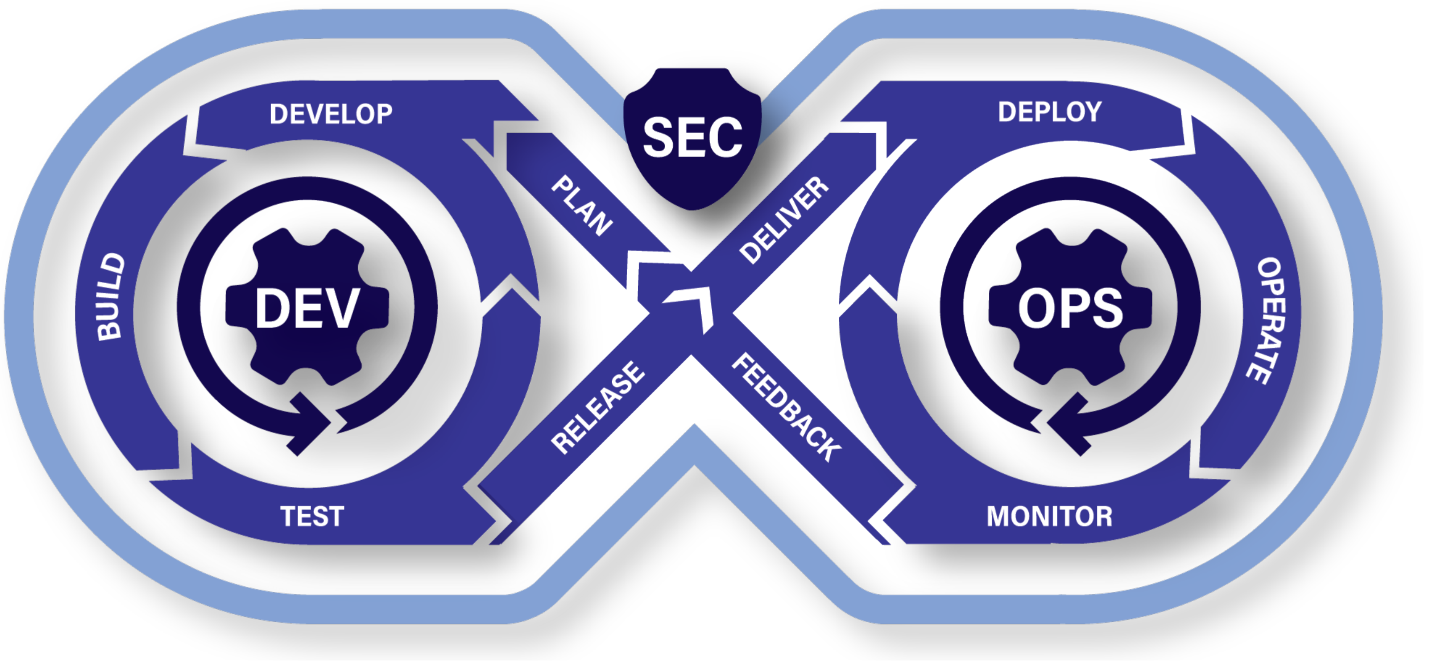


Figure 1: DevSecOps Lifecycle Phases (Infinity Loop)

2.2 DevSecOps Lifecycle

The DevSecOps lifecycle consists of ten phases. The ten phases group common activities and any quality gates that occur at that point in the lifecycle and proceed in a cyclical manner with the result of a cycle being a software product release. A software product release is an iteration of the product that includes new functionality, performance enhancements, and/or security improvements. Each cycle builds upon the results of previous cycles. The duration of the cycle is determined by the mission owner in response to the needs of the stakeholders. Some phase activities may start in one phase and continue throughout additional phases of the lifecycle. The ten phases are defined below with specific activities for each phase enumerated in the **DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools**.[1]

1. **Plan** – Define the requirements and objectives of the product, with the greatest focus on the contents of the next release or version.
2. **Develop** – Create the elements of the product based upon the requirements and objectives identified in the Plan phase.
3. **Build** – Compile and/or integrate the new elements with any existing elements of the product.
4. **Test** – Verify that the new elements meet requirements and objectives.
5. **Release** – Package the product and create required documentation.
6. **Deliver** – Make the product available to the operational environment.
7. **Deploy** – Install and/or configure the product within the operational environment.
8. **Operate** – Use the product in the operational environment.
9. **Monitor** – Observe, measure, and monitor the product as it is used.
10. **Feedback** – Transmit observed behavior and desired changes for consideration in the next iteration of the software product.

2.3 Key Concepts

There are fundamental concepts associated with adopting DevSecOps. These concepts are described in Table 1 and are detailed in the sections that follow.

Table 1: DevSecOps Key Concepts

Concept	Description
Software Supply Chain	A software supply chain is a collection of steps that create, transform, and assess the quality and policy conformance of software artifacts. These steps are often carried out by different actors who use and consume artifacts to produce new artifacts. (NIST SP 800-204D)
Software Factory	In the DoD, a software factory is defined as a collection of people, tools, and processes that enables teams to continuously deliver value by deploying software to meet the needs of a specific

[1] DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools, May 2023. Available: https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d.

	community of end users. It leverages automation to replace manual processes.
DevSecOps Platform	A DevSecOps platform is defined as a group of resources and capabilities that form a base upon which other capabilities or services are built and operated within the same technical framework. Use of a DevSecOps platform is necessary to accelerate development, delivery, and cybersecurity accreditation.
Continuous Integration / Continuous Deployment (CI/CD) Pipeline	A CI/CD pipeline is the process workflows and associated tools to achieve the continuous integration and continuous delivery of software with maximum use of automation.
Continuous Authorization to Operate (cATO)	cATO is the state achieved when the organization that develops, secures, and operates a system has demonstrated sufficient maturity in their ability to maintain a resilient cybersecurity posture that traditional risk assessments and authorizations become redundant. This organization must have implemented robust information security continuous monitoring capabilities, active cyber defense, and secure software supply chain requirements to enable continuous delivery of capabilities without adversely impacting the system’s cyber posture.

3 Implementing DevSecOps

3.1 DevSecOps Implementation Components

There are several critical components in implementing DevSecOps. These components include the software supply chain, software factories, DevSecOps platforms, CI/CD pipelines, and Infrastructure as Code (IaC). The software supply chain provides the full context for software delivery, setting the stage for the other components. A software factory encompasses the entire set of software capabilities required to deliver resilient software capability at speed. The DevSecOps platform consists of those software capabilities that are common across all software factories and provides a standardized and secure foundation for software development. CI/CD pipelines include the tools, process workflows, scripts, and environments to produce a set of software deployable artifacts with minimal human intervention. IaC consists of code baselines that automatically establish common infrastructure or other service capability for faster, more consistent implementation. These components are described in the following sections.

Note that a DevSecOps implementation does not require a specific architecture, containers, or even explicit use of cloud computing. However, the use of these components is strongly recommended, and in some cases mandated by specific DoD reference designs. Teams are encouraged to start small and build up their capabilities progressively, striving for continuous process improvement at each of the ten lifecycle phases.

3.1.1 Software Supply Chain

The software supply chain is a logistical pathway that covers the entirety of all the hardware, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and tools and practices that are brought together to deliver specific software capabilities. The software supply chain matters because the end software supporting the warfighter, from embedded software on the bridge of a Naval vessel to electronic warfare algorithms in an aircraft, is only possible due to the people, processes, and tools that create the end result. For example, a compiler is unlikely to be deployed onto a physical vessel, but without the compiler there would be no guidance system. For this reason, the software supply chain must be recognized, understood, secured, and monitored to ensure mission success.

Software supply chains exist for business systems, weapons systems, and everywhere software is developed and deployed. It is easy, but naïve and incorrect, to dismiss an embedded system in a projectile as “isolated” and disconnected. The projectile includes embedded software that was compiled leveraging 3rd party libraries and links to hardware drivers and relies upon features of embedded firmware. Additionally, the very performance of the software was likely tested using modeling and simulation software executing within a high-performance computing cloud.

DevSecOps methodologies span multiple links of the software supply chain. DevSecOps cannot exist without this logistical supply chain – Integrated Development Environments (IDEs), build tools, code repositories, artifact repositories, testing software suites, and many other pieces of software must work together in unison to effectively execute a DevSecOps powered software factory. The totality of these environments must be considered when evaluating the software supply chain. For example, the cybersecurity and risk postures of a specific artifact or application would be calculated using the product rule across the entirety of the software supply chain. If the compiler is 90% secure, the code repository is 90% secure, the artifact repository is 90% secure, and the container orchestrator is 90% secure – the overall system is **not** 90% secure. The cybersecurity level of the end-to-end ecosystem is actually .9 * .9 *.9 * .9, or roughly 65% secure.

DevSecOps aims to harness the collective expertise and knowledge across the *entire* software supply chain to mitigate risk at each step. Only then can the overall cyber survivability of the ecosystem significantly increase. To further illustrate this point, if a DevSecOps team only increases security 5%, raising each level from 90% to 95%, then overall cyber survivability security jumps from 65% to 81%.

3.1.2 Software Factory

A software factory is a collection of people, tools, and processes that enables teams to continuously deliver value by deploying software to meet the needs of a specific community of end users. It leverages automation to replace manual processes. Software factories are strongly linked to one or more specific software supply chains but the software factory itself is not an entire software supply chain.

A software factory provides a structured and repeatable approach to software development, enabling organizations to streamline their processes, improve efficiency, and ensure consistent quality across products. It consists of the full set of software capabilities depicted in Figure 2, which organizes the software capabilities under Infrastructure, Digital Platform, and Applications.

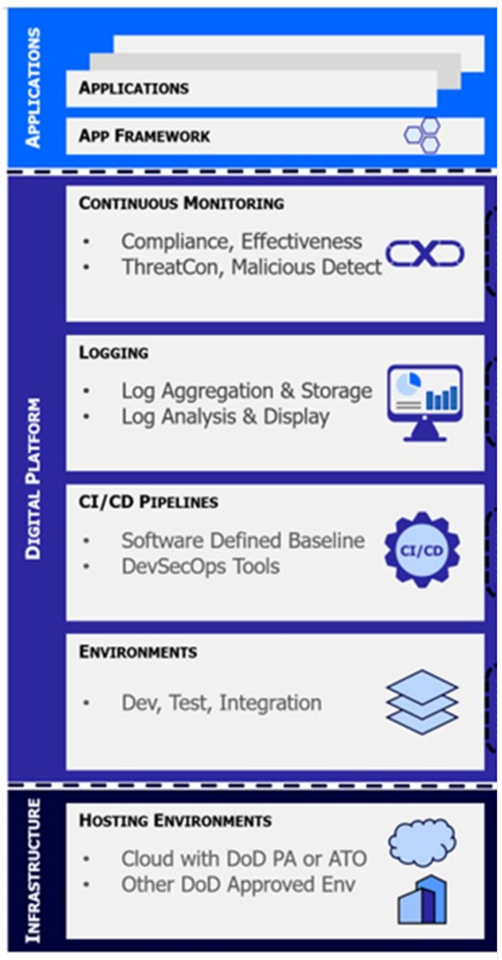


Figure 2: Software Capabilities of a Software Factory

<u>Infrastructure Capabilities</u> supply the hosting environment for the software factory, explicitly providing compute, storage, network resources, and additional CSP managed services to enable function,

cybersecurity, and non-functional capabilities. Typically, this is either an approved or DoD provisionally authorized environment provided by a CSP but is not limited to a CSP.

Digital Platform Capabilities include the distinct development environments of the software factory, its CI/CD pipelines, a clearly implemented log aggregation and analysis strategy, and continuous monitoring operations. These capabilities should support multi-tenancy, enforce separation of duties for privileged users, and be considered part of the cyber survivability supply chain of the final software artifacts produced.

The set of environments within this capability set rely upon CI/CD pipelines, each equipped with a purpose-driven set of tools and process workflows. The environmental boundaries are heavily automated with strict gates controlling promotion of software artifacts from development to test, and from test to integration. These capabilities also encompass planning and backlog functionality, configuration management (CM) repositories, and local and released artifact repositories. Access control for privileged users is expected to follow an environment-wide *least privilege* access model. Continuous monitoring assesses the state of compliance for all resources and services evaluated against NIST SP 800-53 controls. Reference NIST Special Publication 800-137, Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations and (CUI) DoD Information Security Continuous Monitoring Strategy for more information.

Application Capabilities include application frameworks, data stores such as relational or NoSQL databases and object stores, and other middleware unique to the application and outside the realm of the CI/CD pipeline.

Software factories may contain multiple assembly lines, or in software parlance, *CI/CD pipelines*. Each pipeline contains and defines a complete set of tools, process workflows, scripts, and environments that co-exist to produce a set of production quality software artifacts with minimal human intervention. Figure 3 illustrates a generic software factory with multiple pipelines where the production environment is integrated within its defined security boundaries (e.g., software developed and delivered to a production environment within the boundaries of their DevSecOps platform). Figure 4 illustrates a second software factory use case where the production environment is external to the software factory security boundaries as defined in their Authorization to Operate (ATO) (e.g., software developed and delivered to a weapons system or afloat asset).

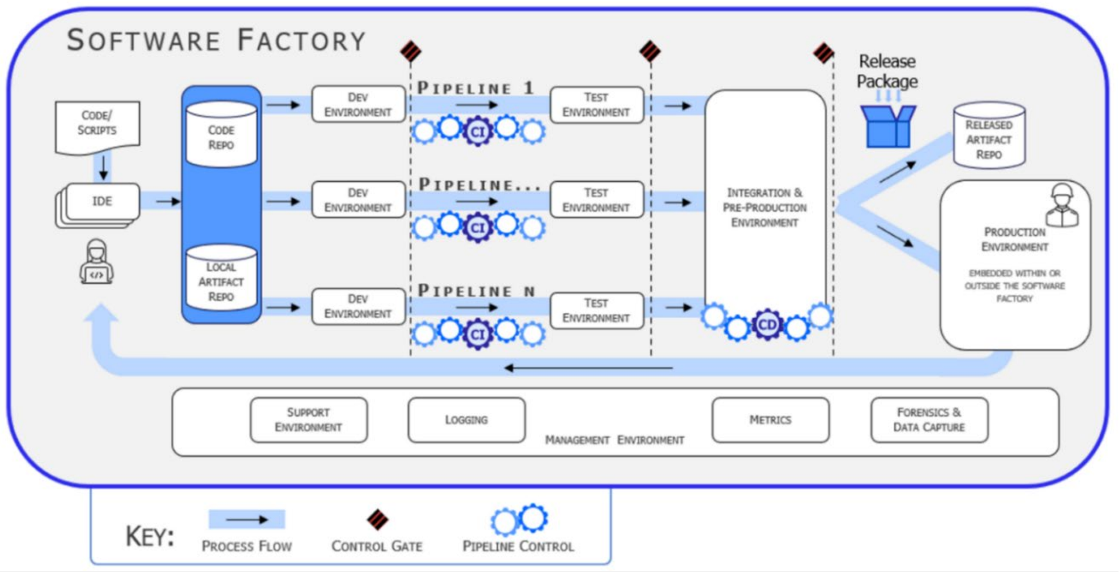
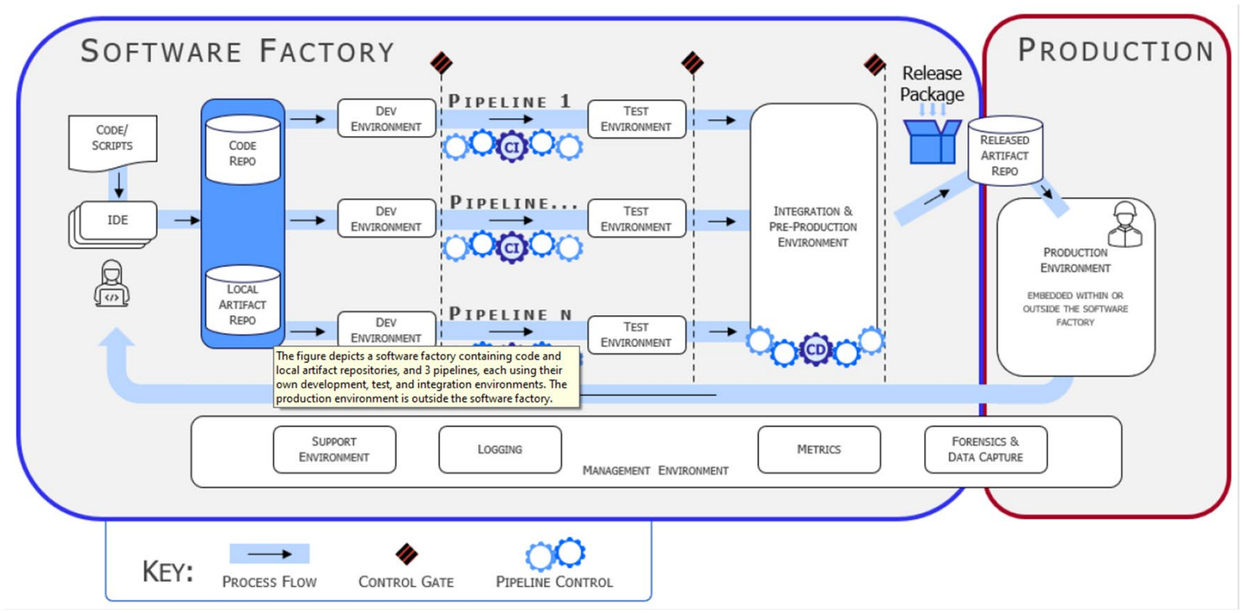


Figure 3: Generic Software Factory with Integrated Production Environment



The figure depicts a software factory containing code and local artifact repositories, and 3 pipelines, each using their own development, test, and integration environments. The production environment is outside the software factory.

Figure 4: Generic Software Factory with a Separate Production Environment

In an ideal or very mature software factory, Free and Open Source Software (FOSS), Commercial off the Shelf (COTS), Government off the Shelf (GOTS), and/or newly developed code and supporting scripts and configuration files are committed into the factory's local artifact or code repository, respectively. Assembly line-like automation moves software changes from one phase to the next. When starting, a best practice is to migrate into a software factory while sustaining legacy capability.

The **development environment** contains the rawest form of source code. When a developer looks to merge their completed work into the main branch of the code repository, they encounter a **control gate**. As an example of this, if code is successfully compiled, it will forward with a pull/merge request for peer review, which includes a critical security step that is the software code equivalent of *two-person integrity*. If the peer review identifies security flaws, architectural concerns, a lack of appropriate documentation within the code itself, or other problems, the merge request can be rejected, and the code sent back to the software developer for rework. Once the merge request is approved and any other activities such as unit tests are completed, the continuous integration step is triggered.

Continuous integration (CI) executes automated tests such as unit tests and Static Application Security Test (SAST), verifying the integrity of the work in the broader context of the artifact or application. The CI assembly line is solely responsible at this point for guiding the pipeline, including but not limited to dependency tracking, regression tests, code standards compliance, and pulling dependencies from the local artifact repository, as necessary. When the CI completes, the artifact is automatically promoted to the **test environment**.

Usually, the test environment is where a more in-depth set of tests is executed. For example, hardware-in-the-loop (HWIL) tests or software-in-the-loop (SWIL) tests may occur at this point. In addition, the test environment pipeline performs additional or more in-depth testing variants of static code analysis, functional tests, interface tests, and dynamic code analysis. If these tests complete without error, then the artifact is poised to pass through another control gate into the **integration environment** or be sent back to the development team to fix any issues discovered during the automated testing.

Once the code and artifact(s) reach the integration environment, the continuous deployment (CD) assembly line is triggered. More tests and security scans are performed in this environment, including operational and performance tests, user acceptance tests, and additional security compliance scans.

Once all tests complete without issue, the CD assembly line releases and delivers the final product package to the released artifact repository.

Released is never equivalent to Deployed! This is a source of confusion for many. A released artifact is *available* for deployment. Deployment may or may not occur instantly. A laptop that is powered off when a security patch is pushed into production will not immediately receive the artifact. Larger updates or out-of-cycle refreshes like anti-virus definition refreshes often require the user to initiate. The *deployment* occurs later. While this is a trivialized example, it effectively illustrates that released is never equivalent to deployed.

The CD acronym is often ambiguously used to mean either *Continuous Delivery* or *Continuous Deployment,* covered next. These are related but *different* concepts. This document will use CD to mean continuous deployment. In this document, Continuous Delivery is a software development practice that allows frequent releases of value to staging or various test environments once verified by automated testing. Continuous Deployment relies on a manual decision to deploy to production, though the deployment process itself should be automated.

DoD needs multiple software factories tuned for specific types of software systems, such as web applications or embedded systems that may include significant amounts of HWIL for automated testing. It also requires software factories operating at varying classification levels in both cloud and on-premises or disconnected environments.

Under the shift to a cATO, each software factory will have its processes, teams, and storage reviewed, certified, and continuously monitored to allow them to deploy applications into a continuously monitored system. This shift greatly lessens the initial burden of achieving an ATO for each piece of software, as the process and roll out are certified and fed into a continuous monitoring architecture.

An ideal DevSecOps software factory does the following:

1. **Standardization***:* Establishes standardized practices, processes, and tools for software development. It promotes consistency and reduces variability, allowing product teams to work more efficiently and effectively.

2. **Automation:** Automates various activities of the software development lifecycle, including code compilation, testing, and deployment. This automation reduces manual effort, minimizes errors, and accelerates the delivery of software updates.

3. **CI/CD:** Provides continuous integration and deployment, allowing developers to frequently integrate their code changes and deliver software updates in an automated and reliable manner. It moves Developmental and Operational Test and Evaluation activities earlier in the CI/CD pipelines instead of bolted on at the end, facilitating more rapid feedback to the development teams. This promotes agility, collaboration, and faster time-to-market.

4. **Scalability and Flexibility:** Scales and adapts to the needs of the organization. It must be designed for multi-tenancy and accommodate different types of applications, technologies, and deployment environments, including on-premises, cloud, and hybrid infrastructures.

5. **Security and Compliance:** Incorporates security practices throughout the development and delivery process. It provides a dynamically scalable set of pipelines with three distinct cyber survivability control gates and integrates security scanning tools, vulnerability assessments, and

compliance checks to identify and address security issues early on. This helps ensure that software products meet security standards and regulatory requirements and provides assurance as an AO that functional, security, integration, operational, and all other tests are reliably performed and passed prior to formal release and delivery.

6. **Collaboration and Communication:** Fosters collaboration and communication among teams, enabling them to work together seamlessly. It provides shared repositories, issue tracking systems, and collaboration tools to facilitate teamwork and information sharing.

7. **Continuous Improvement:** Promotes a culture of continuous improvement by collecting metrics, analyzing performance, and identifying areas for enhancement. It enables teams to learn from past experiences, iterate on processes, and drive ongoing optimization.

Every DoD organization is encouraged to seek out an existing managed PaaS to learn about and begin applying DevSecOps.

DoD DevSecOps Platform and Software Factory Inventory (CAC-enabled)

https://go.intelink.gov/ab7U5ad

3.1.3 DevSecOps Platform

A DevSecOps platform is defined as a group of resources and capabilities that form a base upon which other capabilities or services are built and operated within the same technical framework. It provides a comprehensive set of common tools, services, and infrastructure to support the implementation and execution of DevSecOps practices within an organization and serves as a centralized hub for managing and automating various phases of the software development lifecycle, with a strong focus on security. It can be a multi-tenant environment that brings together a significant portion of a software supply chain, operating under cATO or a provisional ATO. Each DevSecOps platform may be composed of multiple software factories, multiple environments, multiple tools, and numerous cyber resiliency tools and techniques.

Figure 5 illustrates a DevSecOps platform in support of a software factory. It identifies reference design interconnects which represent a unique set of tools and activities that exist within and/or at the boundaries between the digital platform capabilities. These unique toolsets or configurations to include proprietary tooling or specific architectural constructs connect various aspects of the DevSecOps platform together. Well-defined reference design interconnects as provided in DoD approved DevSecOps reference designs enable tailoring of the software factory design, while ensuring that core capabilities of the software factory remain intact.

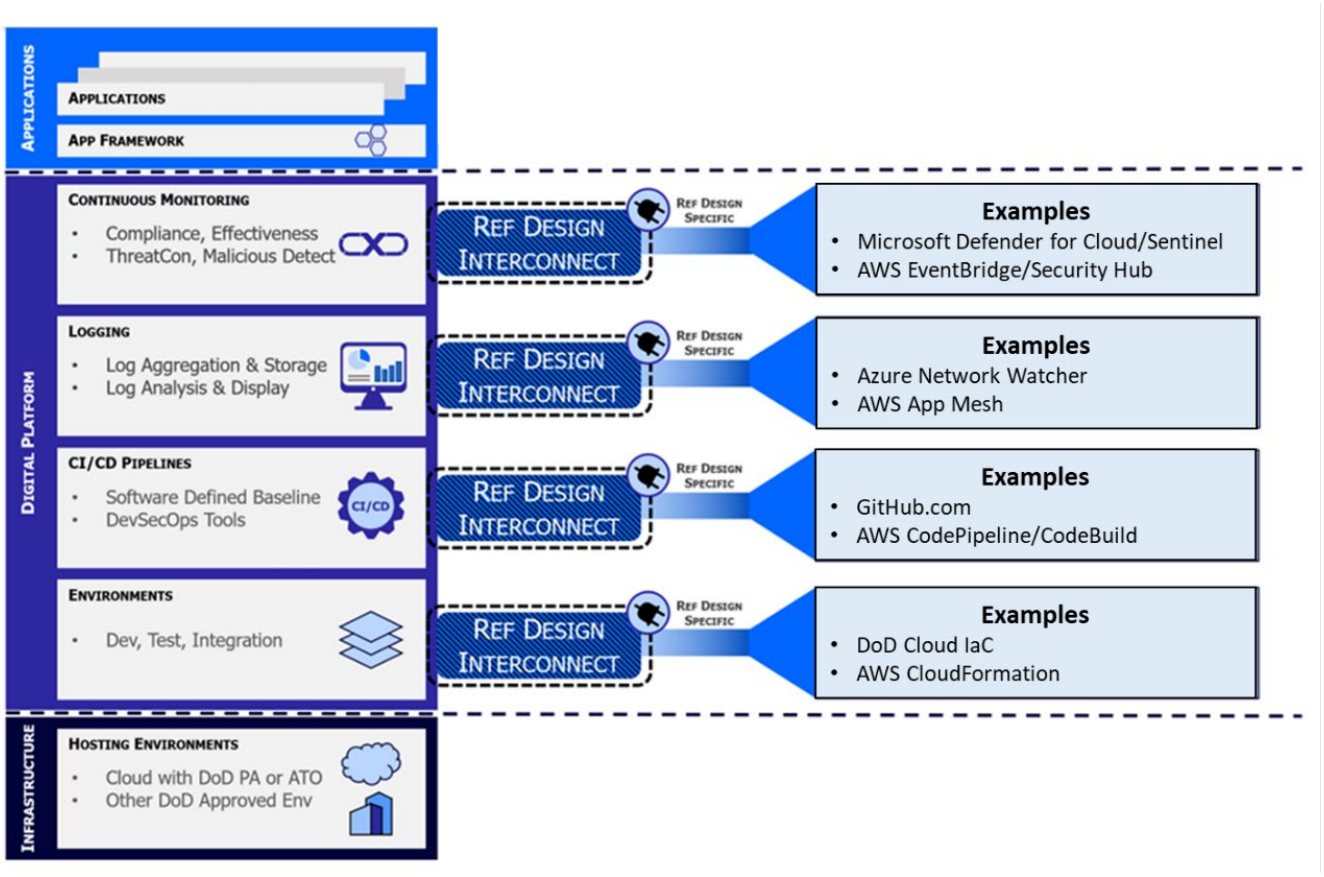


Figure 5: DevSecOps Platform

DevSecOps platforms provide rapid access to common software development infrastructure capabilities for use by a DevSecOps team. DevSecOps platforms employ automation at multiple levels and across multiple activities in the develop, build, test, release, and deliver phases of the DevSecOps lifecycle. Each environment in the process is automated to the maximum extent that is safely and securely possible, rehydrated using **Infrastructure as Code (IaC)** and **Configuration as Code (CaC)** that run on various tools. DevSecOps platforms are expected to be instantiated from hardened IaC code and scripts or DoD hardened containers from a DoD artifact repository like Iron Bank, a hardened container image repository and one of Platform One's value streams.

Platforms are necessary to implement software delivery through DevSecOps. DevSecOps platforms represent a prime opportunity for the DoD software community to leverage reuse in the underlying infrastructure. DoD organizations are encouraged to leverage government-managed and curated platforms, such as Platform One's tailorable Big Bang platform, and contribute modifications back to the baseline to provide the DoD-enterprise with a growing set of pre-configured integrations.

3.1.4 CI/CD Pipelines

A critical part of a software factory are the Continuous Integration/Continuous Delivery (CI/CD) pipelines. The adoption of CI/CD pipelines reduce risk through the practice of making many small, incremental changes to software instead of one gigantic change. The incremental changes can be reviewed quickly. Mistakes that are introduced are easier to capture and isolate when only a few things have changed.

As described previously, a software factory may contain multiple CI/CD 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. Different pipelines are needed for different types of software such as web applications, business systems, command and control systems, embedded systems, or AI/ML.

It is within each CI/CD pipeline that the software supply chain is actualized. Per NIST SP 800-204D, “Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines,” a software supply chain must not be reduced to merely a set of artifacts. Every software supply chain must explicitly address every link that exists between writing source code through production deployment:

“While software composition (e.g., dependency management) is under the purview of software supply chain activities, other often overlooked activities are central to the software supply chain. This includes writing source code; building, packaging, and delivering an application; and repackaging and containerization.”

Activities in the CI/CD pipeline empower stronger communication between developers and information security personnel. The creation of shared context by and between development teams and information security teams enables security controls to be discussed and expectations established for each individual link across the totality of the CI/CD pipeline. Figure 6 illustrates a generic CI/CD pipeline.

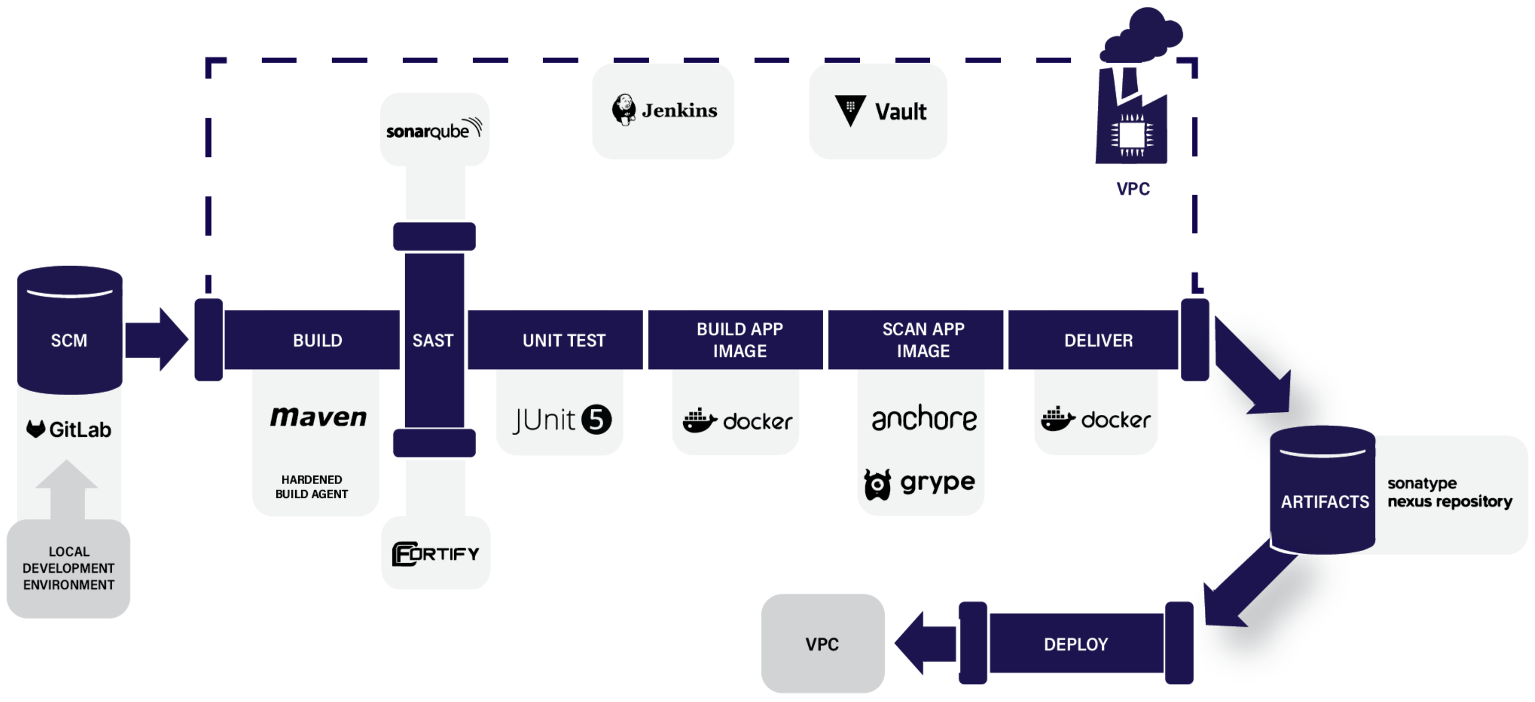


Figure 6: Example CI/CD Pipeline

As an actor initiates a series of steps within a CI/CD pipeline, resources and input artifacts are consumed, and one or more final artifacts result. It is crucial to monitor each step, or link, in the software supply chain across every segment of the CI/CD pipeline. This monitoring involves collecting and analyzing comprehensive metadata at each stage, providing visibility into the entire process. Effective observability allows teams to track the flow of data, identify potential bottlenecks, and ensure that each step is executed correctly and securely.

Observability goes beyond simple monitoring; it enables teams to gain deep insights into the performance, security, and reliability of the pipeline. By aggregating logs, metrics, and traces from each step, teams can quickly diagnose issues, ensure compliance with security policies, and optimize the pipeline for efficiency. This level of observability is especially important in environments that have received a cATO, where continuous monitoring capabilities are in place to maintain the security and integrity of the system.

3.1.5 Infrastructure as Code (IaC)

The shift to immutable infrastructure using IaC provides security and value in a number of ways. First, it removes lethargic and error-prone step-by-step deployment and configuration guides performed manually. In a legacy, manual driven approach, it is too easy to inadvertently skip a step or mistype a configuration command. Second, it confirms that the command will execute as expected, mitigating the risk of a change without any type of peer review before execution. Third, by providing a standard deployment model, a standard set of outputs can be auto-ingested into Defensive Cyber Operations (DCO) platforms and data collection/visualization mediums. This allows DCO to begin instantaneously and provides data analytics to identify necessary next innovations.

IaC plays a critical role in automation for DevSecOps platforms. For Platform teams, IaC streamlines infrastructure deployment, authorization, and security for customers leveraging cloud, shortening the time to stand up the typical infrastructure by months. IaC consists of baselines that automatically establish cloud environments in hours. It accelerates the authorization process with inheritable common controls and the use of PaaS services, which reduces the need for Security Technical Implementation Guides (STIGs), Assured Compliance Assessment Solution (ACAS), and Host-Based Security System (HBSS). If implemented correctly, it can shorten the deployment of networking, identity, and security policies for security compliance from the standard 30 weeks to hours.

DoD provides approved IaC templates for use across DoD Components at https://www.hacc.mil/Portfolio/DOD-Cloud-IaC/. DoD IaCs leverage automation to generate preconfigured, preauthorized PaaS focused environments. These baselines exist in the form of IaC templates that organizations can deploy to establish their own decentralized cloud platform. Baselines significantly reduce mission owner security responsibilities by leveraging security control inheritance from CSP PaaS managed services, where host and middleware security is the responsibility of the CSP including hardening and patching. Each baseline documents its associated inheritable controls to expedite the Assessment and Authorization (A&A) process. DoD IaC baselines can be built into DevSecOps pipelines to rapidly deploy the entire environment and mission applications.

3.1.6 Common Tools

Every DevSecOps software factory and platform must include a minimal set of common tooling. The use of the word *common* is indicative of a class of tooling; it does not, nor should it be construed that the *same* tool must be used across every implementation.

The *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools* provides a clean set of tables that captures both required and preferred tooling across DevSecOps lifecycle phase activities. DoD approved reference designs augment the *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools*, adding in its environmentally specific required and preferred tooling. Reference designs do not remove a *required* tool or activity, only augment. The *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools* and DoD approved reference designs are available at https://dodcio.defense.gov/Library/.

3.2 DevSecOps Lifecycle Process

Figure 7 visually depicts the DevSecOps phases, feedback loops, and control gates. The lifecycle is built around a series of *iterations*, with each iteration covering the Plan, Develop, Build, Test, Release, Deliver, Deploy, Operate, Monitor, and Feedback phases. While some may view this graphic as a waterfall process, this graphic contains the identical set of steps depicted previously in Figure 1 as an infinite loop but is "unfolded" to effectively illustrate the multiplicity of continuous feedback loops. The unfolded loop better illustrates the relationship between the lifecycle phases and the continuous feedback loops used to drive continuous process improvement.

Feedback loops are critical mechanisms that overlap with specific DevSecOps lifecycle phases. Each feedback loop is built upon transparency and speed. As an illustration, when a software developer commits code to a branch, a build is automatically triggered to confirm the code still builds correctly, and if it doesn't, the developer is immediately notified of the problem.

The following sections address the importance of the iteration as well as each feedback loop and the value it adds to the software factory.

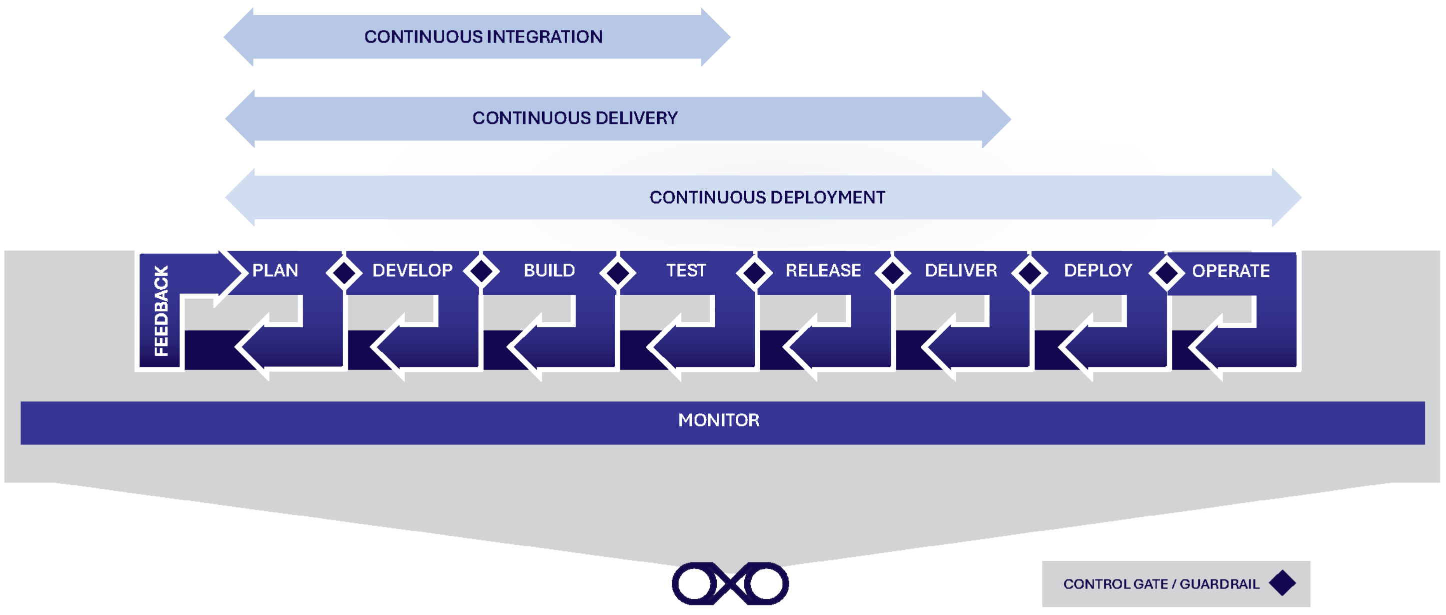


Figure 7: DevSecOps Lifecycle Phases, Continuous Feedback Loops, and Control Gates

3.2.1 Plan

The Plan Phase involves activities that help the team manage time, cost, quality, risk, and issues within the DevSecOps lifecycle. These activities may include business-need assessment, plan creation, and may further include at the story, feature, task, or epic level any combination of feasibility analysis, risk analysis, requirements updating, business process creation, system design, threat modeling, mission-based cyber risk assessments, technical debt management, software factory modification, ecosystem expansion and determining the definition of done. The concept of an iteration is supported in the Adaptive Acquisition Framework, DoDI 5000.87, Operation of the Software Acquisition Pathway. DoDI 5000.87 provides that the program manager (PM) and sponsor define a minimum viable product (MVP) using iterative, human-centered design processes. The PM and the sponsor should also *define a minimum viable capability release (MVCR) if the MVP does not have sufficient capability or performance to deploy into operations.*[2]

The plan phase repeats ahead of each iteration.

3.2.2 Continuous Integration

The CI feedback loop iterates across the *Develop*, *Build*, and *Test* phases of the DevSecOps lifecycle, depicted in Figure 7. If build doesn't complete successfully, then the commit must be sent back to the submitting engineer to fix. Without a successful build, further steps are both illogical and impossible to complete. Common types of feedback between *Develop* and *Build* include a successful build by the build tool (because a broken build shouldn't be merged into the main branch) and a pull request that creates the software equivalency of two-person integrity. The pull request performed is intended to evaluate the architecture and software structure, identify technical debt that the original engineer may (inadvertently) introduce if this commit is merged into the main branch, and most importantly, identify any glaring security risks and confusing code. Once the pull request is merged into the main branch, a complete series of automated tests are executed, including a full set of integration tests.

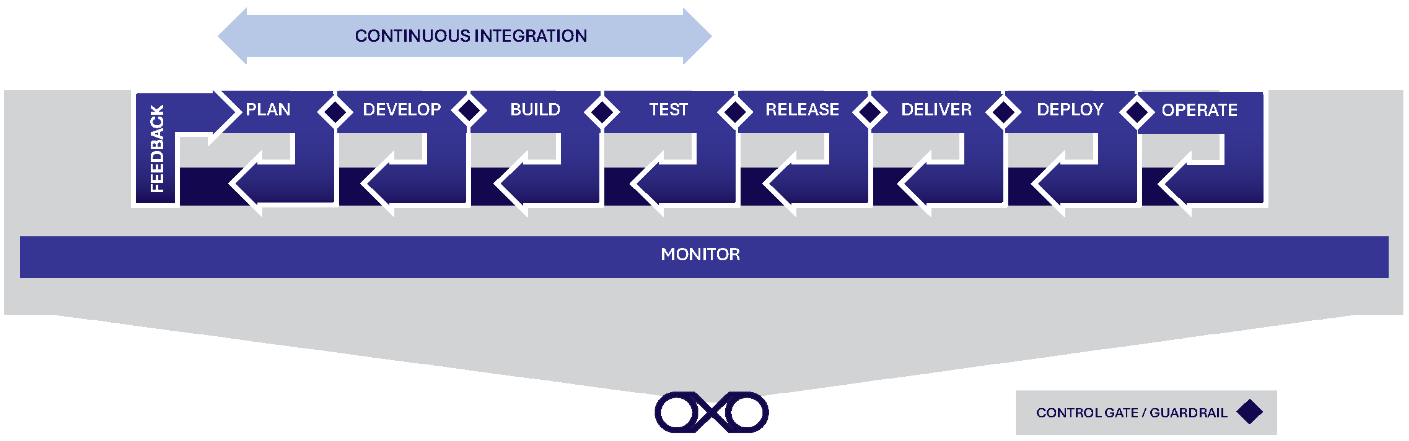


Figure 7: Continuous Integration Feedback Loop

According to Martin Fowler, CI practices occur when members of a team integrate their work frequently, usually each person integrates minimally daily, leading to multiple integrations per day verified by an

[2] DoDI 5000.87, "Operation of the Software Acquisition Pathway", https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF, October 2, 2020

automated build (including test) to detect integration errors as quickly as possible.[3] If multiple teams (with possibly different contractors) are working on a larger, unified system, this means that the whole system is integrated frequently, ideally at least daily, avoiding long integration efforts after most development is complete.

Execution of the automated test suite enhances software quality because it quickly identifies if/when a specific merge into the main branch fails to produce the excepted outcome, creates a regression, or breaks an API.

3.2.3 Continuous Delivery

The Continuous Delivery feedback loop iterates across the *Release, Deliver, and Deploy* phases of the DevSecOps lifecycle depicted in Figure 8.

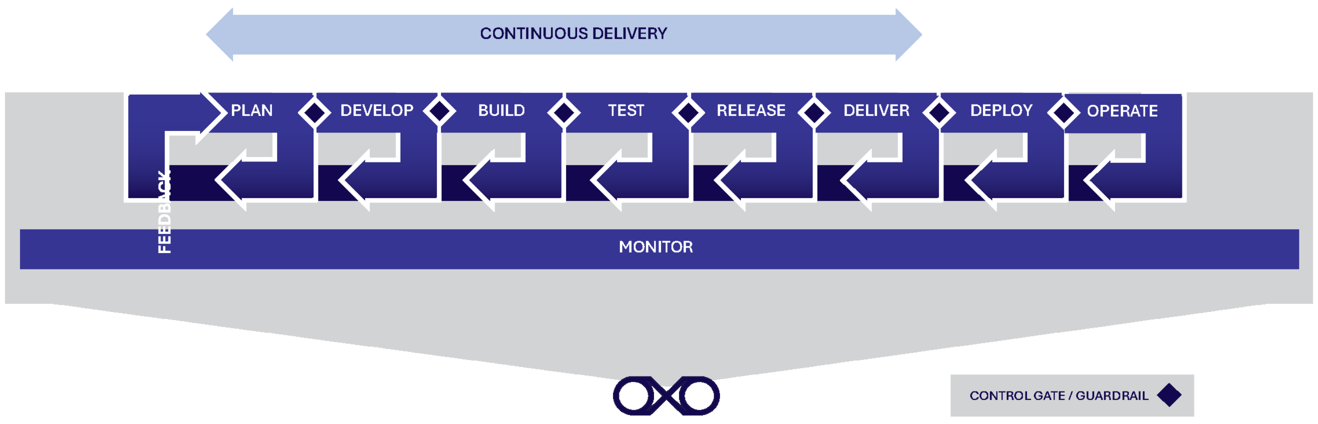


Figure 8: Continuous Delivery Feedback Loop

The most pertinent thing to understand is that *release* and *deliver* do not mean *pushed into production*. Continuous delivery acknowledges that a feature meets the Agile definition of "done." The code has been written, peer reviewed, merged into the main branch, successfully passed its complete set of automated tests, and finally tagged with a version within the source code configuration management tool and deployed into an artifact repository.

At this point, the feature and its related artifacts *could be deployed* but deployment is not mandatory. It is common to group together a series of features and deploy them into production as a unit, for example.

[3] M. Fowler, "Continuous Integration." May 1, 2006, [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html.

3.2.4 Continuous Deployment

The Continuous Deployment feedback loop iterates across the *Plan*, *Develop*, *Build*, *Test*, *Release, Deliver*, *Deploy, and Operate* phases of the DevSecOps lifecycle depicted in Figure 9. Deployment is formally the act of pushing one or more features into production *in an automated fashion*. This is the first *additional* control gate outside of the control gates depicted in the software CI/CD pipeline, visualized in Figure 9.

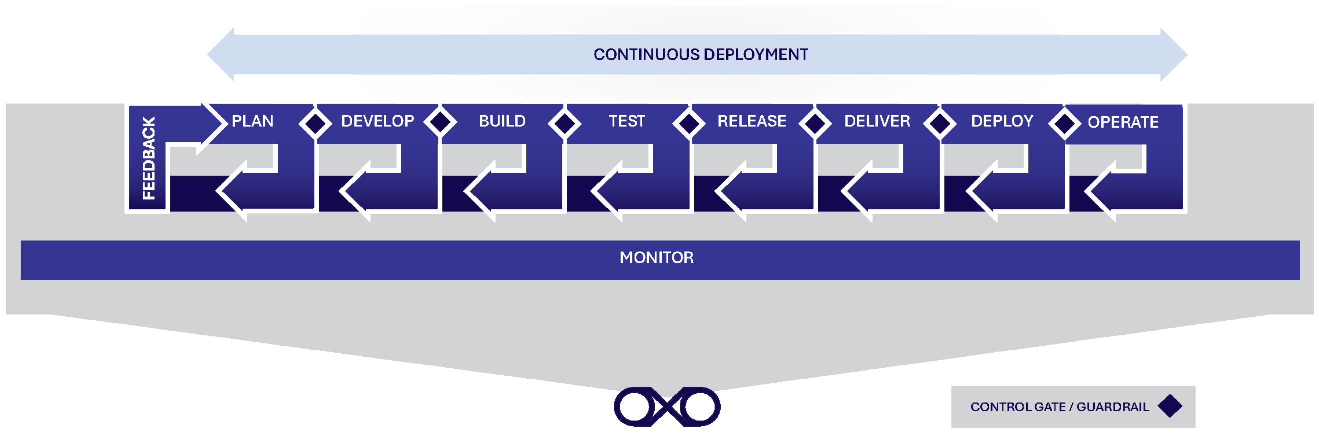


Figure 9: Continuous Deployment Feedback Loop

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

The use of the word continuous here is contextual and situational. First, in some programs, continuous deployment may occur automatically when a new feature is released and delivered. For example, during a cloud based microservice continuous deployment, it is possible to automate the deployment. Alternatively, if this artifact is destined for an underwater resource, it may be several orders of magnitude harder to automatically push a 750MB release of software to a submersed vehicle operating at 300 feet below the surface of the ocean. This scenario further illustrates the separation between *continuous delivery* and *continuous deployment*.

3.2.5 Monitor/Feedback

The Monitor and Feedback phases must bring together a deep, rich set of real-time performance metrics and supporting data to continuously evaluate the totality of the software environment. These phases serve two main functions: cybersecurity monitoring to ensure events and incidents are handled in accordance with DoD mandates and policies and live data feedback and interaction between network defenders, end users, and developers. For cybersecurity, the antiquated snapshot view of network security is replaced with real time feeds, allowing security actions to be taken by local defenders, monitoring teams (Cybersecurity Service Providers (CSSPs)), incident response teams (Cyber Protection Teams (CPTs)), and Command and Control (C2) elements of U.S. Cyber Command/Joint Force Headquarters – DoD Information Network (JFHQ-DoDIN). For customer satisfaction, developers receive feedback regarding the operational effectiveness of the software release and its impact in meeting mission requirements. Some other monitoring activities include fault detection, log auditing, and value assessment.

3.3 Integration with Other DoD Processes

A software factory consists of the technology and processes focused on the development and delivery of software capability and is only one of many processes in the Department that impact speed to delivery. Figure 10 depicts other DoD processes (not all inclusive) that must be integrated with the DevSecOps lifecycle to deliver better software faster to the warfighter. These processes include security, risk management framework, test and evaluation, and acquisition.

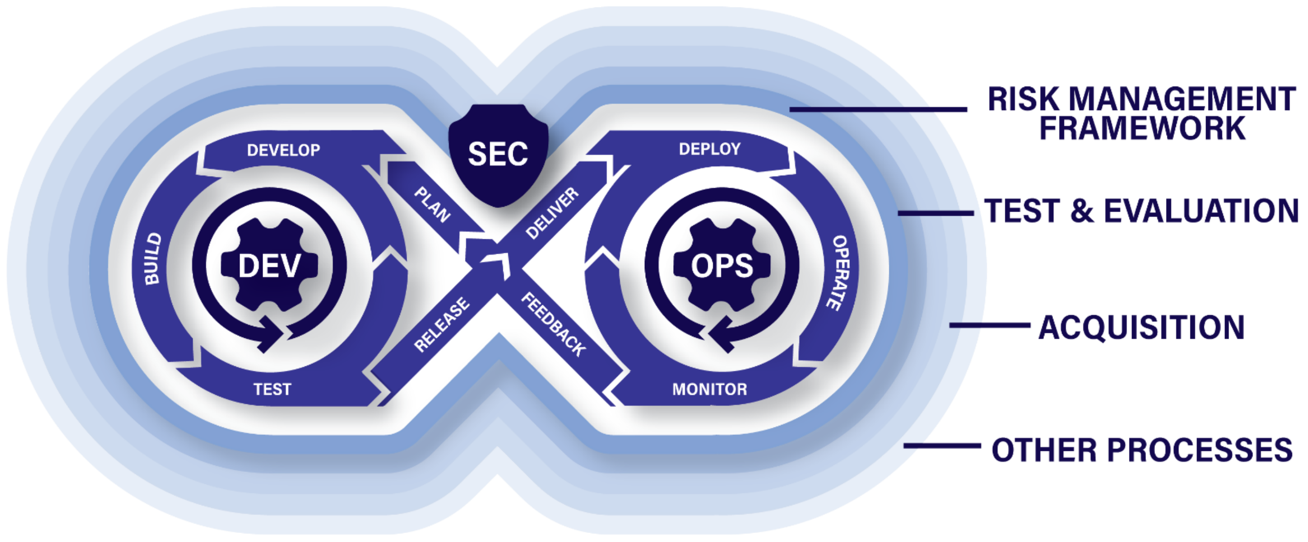


Figure 10: DevSecOps Lifecycle Integration with Other DoD Processes

3.3.1 Security

The security inherent within DevSecOps is achieved through a fundamental change in the culture and approach to cybersecurity and testing. Security is continuously *shifted left* and integrated throughout the software artifacts from the beginning. As software is developed, the development teams use secure coding techniques and practices while the pipeline includes control gates and guardrails that evaluate code for discrepancies and vulnerabilities before sending it to production. Additionally, security practices continue through deployment and into production. Repositories are scanned and software updated when new vulnerabilities are found, and continuous monitoring in production helps identify operational vulnerabilities and insider threat incidents. One can see that security spans the full lifecycle, whether during development, deployment, or production. Assessments, testing, documentation, detection, mitigation, and remediation are also required to provide the most secure operational capability possible.

Figure 11 identifies security activities that occur throughout the DevSecOps lifecycle. These are not all inclusive. Each shield surrounding the DevSecOps lifecycle represents a distinct category of cybersecurity testing and activities. This blanket of protection is intentionally depicted surrounding the ten phases because these tests must permeate throughout the lifecycle to achieve benefits. Failure to weave security and functional testing into just one of the ten phases can create a risk of exposure, or worse, compromise, in the final product.

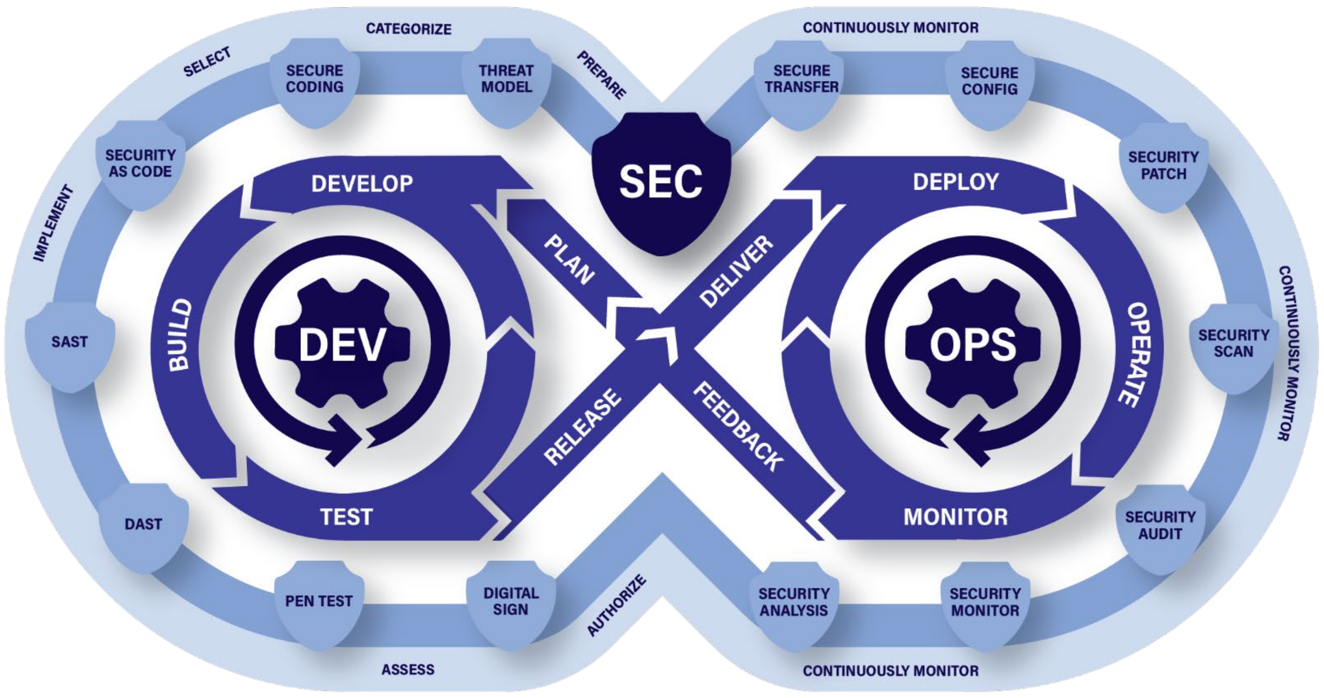


Figure 11: DevSecOps Integrated with Security

This approach differs from the view that cybersecurity can simply be a bolt-on activity after the software is built and deployed into production. When security problems are identified in production software, they almost always require the software development team to (re-)write code to fix the problem. DevSecOps is only fully realized when security and functional capabilities are built, tested, and monitored at each step of the lifecycle, preventing security and functional problems from reaching production in the first place.

3.3.1.1 Zero Trust

DevSecOps software factories and platforms must adopt zero trust as the target security model for cybersecurity. Teams should consistently strive to bake in as many of the zero trust principles as are supported by their architecture across each of the ten phases of the DevSecOps lifecycle. Further, DevSecOps teams must fully consider security from both the end user perspective and all non-person entities (NPEs). To illustrate several of these concepts in a notional list, these NPEs include servers, the mutual transport layer security (mTLS) between well-defined services relying on Federal Information Processing Standard (FIPS) compliant cryptography, adoption of *deny by default* postures, and an understanding of how all traffic, both north-south and east-west, is protected throughout the system's architecture.

DevSecOps inherently addresses the principles of zero trust by integrating development, security, and operations teams, and addressing security throughout each phase. It includes control gates and guardrails prior to deployment and relies on constant monitoring and feedback to address any vulnerabilities or incidents.

For more information on zero trust, refer to the DoD Zero Trust Strategy.[4]

3.3.1.2 Risk Management Framework

The Risk Management Framework (RMF) is the framework DoD uses to ensure all IT systems and applications are “cyber” secure. RMF provides a disciplined and structured, yet flexible process for managing security. The six-step process is designed to develop strong cybersecurity through proper categorization, vulnerability identification and mitigation, assessment, and monitoring of systems and software. RMF includes activities to prepare organizations to execute the framework at organizationally defined risk management levels. RMF is the key component of the process used to authorize systems, applications, and networks. An AO will review the output from a risk management evaluation of security controls and system configuration information to determine the residual risk associated with a system, application, or network. For DevSecOps software factories and platforms, the AO must include the people, processes, and technology used to develop software in their analysis. Most authorizations will provide a one-, two-, or three-year authorization based on the level of residual risk. A reauthorization must take place at the end of the original authorization.

Continuous Authorization to Operate: A cATO is the state achieved when the organization(s) that develops, secures, and operates a system has demonstrated sufficient maturity in their ability to maintain a resilient cybersecurity posture that traditional risk assessments and authorizations become redundant. This organization must have implemented robust information security continuous monitoring capabilities, active cyber defense, and secure software supply chain requirements to enable continuous delivery of capabilities without adversely impacting the system’s cyber posture.

- **Continuous Monitoring:** Per NIST SP 800-137.26, “Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations,” ISCM is defined as maintaining ongoing awareness of information security, vulnerabilities, and threats to support organizational risk management decisions.

- **Active Cyber Defense:** ISCM, as a component of active cyber defense, includes an integrated auditing and incident response capability with predefined triggers and thresholds that can be displayed on an interactive dashboard. Continuously monitoring a DevSecOps platform and its software factories requires the ability to continually evaluate and assess security controls, gates, guardrails, and operating conditions for software either in development or operations. The outcome of this monitoring ensures the effectiveness of security control implementation, verifies compliance with organizational and federal security requirements, and provides the AO information on the residual risk associated with the development and operations of the platform and software. This is a critical component to maintaining a cATO.

- **Secure Software Supply Chain:** NIST 800-218, “Secure Software Development Framework (SSDF),” describes a set of fundamental, secure software development practices that can be integrated into each software development lifecycle implementation. The software development practices are organized into four groups:
    - Prepare the Organization
    - Protect the Software
    - Produce Well-Secured Software

[4] DoD CIO Cyber Security, Zero Trust Strategy, https://dodcio.defense.gov/Portals/0/Documents/Library/DoD-ZTStrategy.pdf, October 21, 2022.

- Respond to Vulnerabilities

Each group consists of numerous activities/practices that software factories should follow to ensure the software supply chain is secure and software delivered is safe. Following such practices should help software producers reduce the number of vulnerabilities in released software, reduce the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences.

A DevSecOps platform and its software factories must complete an authorization. Once an authorization is approved, the DevSecOps platform and its software factories can expedite future authorization activities and processes by obtaining a cATO. cATO moves away from a control assessment point-in-time document-based approach (though some documents are still required) towards a continuous risk determination and authorization concept by continuously assessing, monitoring, and managing risk. cATO raises the security standard over a traditional Authorization to Operate (ATO) and provides the ability to deploy updated software more rapidly while improving security. cATO includes the need for a Secure Software Supply Chain (SSSC) and requires a Software Bill of Materials (SBOM). Further information about cATO is available in the DevSecOps Continuous Authorization Implementation Guide.[5]

3.3.2 Test and Evaluation

While the DevSecOps lifecycle includes a phase named "Test," there are testing activities throughout the lifecycle, whether by securing the software supply chain or by adopting practices in alignment with NIST SP 800-218. These testing activities are identified in Figure 12 and were developed in coordination with the Operational Test and Evaluation (OT&E) and the Developmental Test and Evaluation (DT&E) organizations and the Joint Interoperability Test Command (JITC). At a high level, the two categories of testing are Functional Testing and Security Testing. The *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools* contains more details about the different tests to include throughout the lifecycle and maps the tests to the SSDF.

Functional Testing: Functional testing is essential to DevSecOps, as it ensures that each product increment functions as intended. Integrated into the CI/CD pipeline, functional testing validates the behavior of individual components, modules, and the entire system, simulating real-world scenarios to confirm correct operation under various conditions. Tailoring the *DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools* to each mission could result in additional testing activities.

Security Testing: There is no "one size fits all" solution for security testing design. Each product team has its own unique requirements and constraints. However, the software artifact promotion control gates are a best practice of the software factory**.** Different pipelines within the software factory may define different collections of tests to maximize the effectiveness of a control gate.

[5] DoD CIO, "DevSecOps Continuous Authorization Implementation Guide," April 11, 2024.

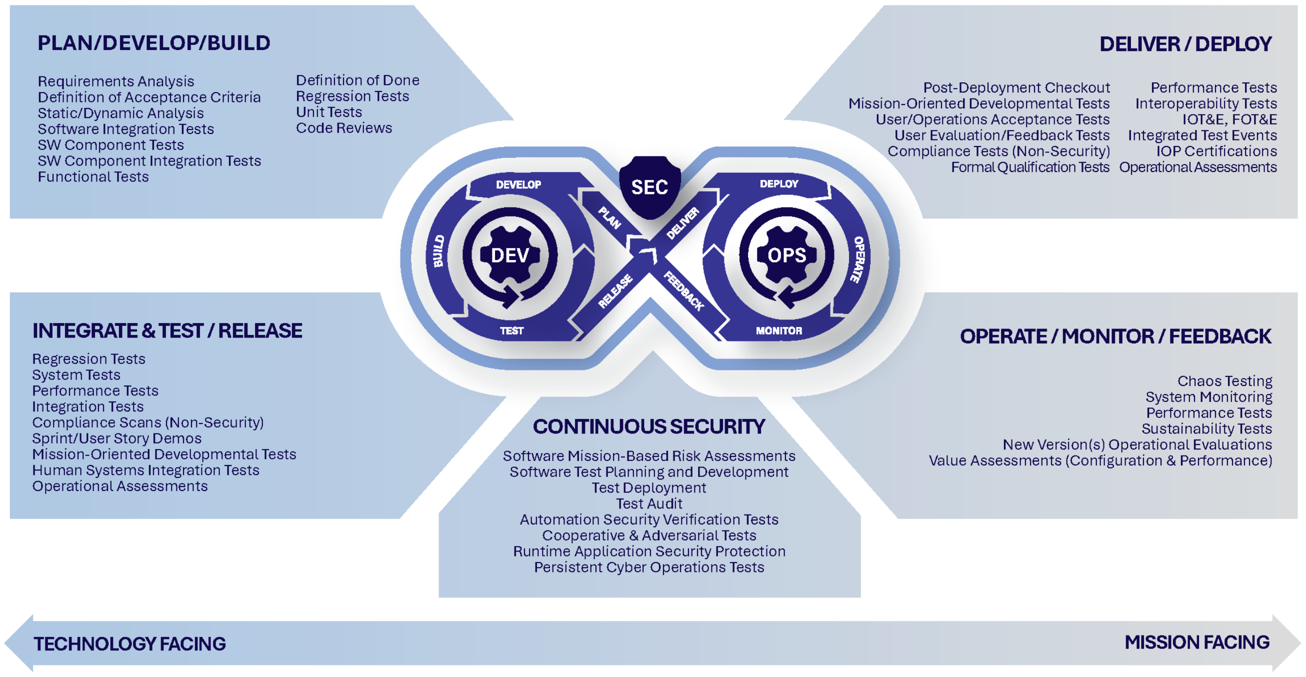

PLAN/DEVELOP/BUILD
Requirements Analysis
Definition of Acceptance Criteria
Static/Dynamic Analysis
Software Integration Tests
SW Component Tests
SW Component Integration Tests
Functional Tests
Definition of Done
Regression Tests
Unit Tests
Code Reviews
DELIVER / DEPLOY
Post-Deployment Checkout
Mission-Oriented Developmental Tests
User/Operations Acceptance Tests
User Evaluation/Feedback Tests
Compliance Tests (Non-Security)
Formal Qualification Tests
Performance Tests
Interoperability Tests
IOT&E, FOT&E
Integrated Test Events
IOP Certifications
Operational Assessments
SEC
DEVELOP
DEV
BUILD
TEST
PLAN
RELEASE
DELIVER
FEEDBACK
DEPLOY
OPS
OPERATE
MONITOR
INTEGRATE & TEST / RELEASE
Regression Tests
System Tests
Performance Tests
Integration Tests
Compliance Scans (Non-Security)
Sprint/User Story Demos
Mission-Oriented Developmental Tests
Human Systems Integration Tests
Operational Assessments
OPERATE / MONITOR / FEEDBACK
Chaos Testing
System Monitoring
Performance Tests
Sustainability Tests
New Version(s) Operational Evaluations
Value Assessments (Configuration & Performance)
CONTINUOUS SECURITY
Software Mission-Based Risk Assessments
Software Test Planning and Development
Test Deployment
Test Audit
Automation Security Verification Tests
Cooperative & Adversarial Tests
Runtime Application Security Protection
Persistent Cyber Operations Tests
TECHNOLOGY FACING
MISSION FACING

Figure 12: Testing Activities for Integration with the DevSecOps Lifecycle

3.3.3 Acquisition

Traditionally, the DoD acquisition process for software applications was lengthy and expensive, taking years and millions of dollars to deploy systems. With the adoption of Agile and DevSecOps and the use of the DoD Software Acquisition Pathway (SWP), the acquisition process can be streamlined and accelerated. This alignment between SWP and DevSecOps provides *for the efficient and effective acquisition, development, integration, and timely delivery of secure software.*[6] Refer to DoDI 5000.87 for the steps in the SWP.

[6] DoDI 5000.87, "Operation of the Software Acquisition Pathway", https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF, October 2, 2020

4 Institutionalizing DevSecOps

4.1 DevSecOps Culture and Philosophy

DORA's 2023 State of DevOps Report established that a healthy, generative culture is instrumental in driving not only organizational performance but also software delivery and operational efficiency. Key to this is the cultivation of an environment where information flows freely, reducing knowledge silos and enhancing the effectiveness of task execution.

Their findings identified that culture drives organizational performance in teams with a generative culture delivering a 30% higher organization performance than teams without. They found that "the relationship between culture and technical capabilities is reciprocal: culture emerges from practices, and practices emerge from culture." Other findings include the following:

- **User-Centered Approach:** A significant revelation is the impact of a user-centered approach in software development. By prioritizing user needs and feedback, organizations witness increased user satisfaction and a better user experience.

- **Work Distribution and Team Performance:** The report finds that equitable work distribution correlates positively with team and organizational performance. However, it suggests that overly formalized processes in work distribution might hinder software delivery performance, indicating a need for balanced and flexible work management practices within DevSecOps frameworks.

- **Organizational Stability vs. Agility:** An interesting observation is the slight decrease in software delivery performance in more stable, established organizations. This observation could imply a need for established organizations to foster a culture of agility and innovation to keep pace with rapidly evolving cybersecurity and operational demands.

- **Information Sharing and Operational Performance:** The findings highlight the importance of open information sharing, which is directly linked to improved software delivery and operational performance. In DevSecOps, this translates to transparent communication and collaboration across all stages of the development lifecycle.

- **Flexibility in Work Arrangements:** The report underscores the benefits of flexible work arrangements across all performance metrics, particularly in software delivery. As organizations navigate remote work policies, maintaining a degree of flexibility can be advantageous.

- **Employee Well-being:** Lastly, the report emphasizes that a healthy culture is foundational to employee well-being, reducing burnout, and increasing job satisfaction and productivity. In the context of DevSecOps, investing in a positive culture is not just beneficial but essential for the overall health and success of the organization.

There are several tenets for a successful transition to a DevSecOps culture:

- Continuous delivery of small incremental changes.

- Include security from the beginning rather than bolt it on at the end.

- Value open source software.

- Engage users early and often.

- Prefer user centered and warfighter focus and design.

- Value automating repeated manual processes to the maximum extent possible.
- Fail fast, learn fast, but don't fail twice for the same reason.
- Fail responsibly; fail forward.
- Treat every API as a first-class citizen.
- Good code always has documentation as close to the code as possible.
- Recognize the strategic value of data; ensure its potential is not unintentionally compromised.

4.2 A Whole-of-DoD Approach

Driving true transformational change in the DoD towards Agile and DevSecOps practices for software development and delivery requires holistic organizational change encompassing people, processes, and technology. To ensure success, we must radically shift our organizational culture away from traditional waterfall methodologies and processes and break down silos to foster collaboration. These efforts are the foundation needed for critical advancements such as artificial intelligence and machine learning.

A whole-of-DoD approach involves a comprehensive and coordinated effort across the Department to embrace and integrate modern software development practices such as Agile and DevSecOps.

Leadership and Culture:

- Leadership commitment: Senior leaders within the DoD must demonstrate a strong commitment to modern software practices and actively promote their adoption throughout the organization.
- Organizational patience: DevSecOps has been proven on programs across the Department, but there have also been DevSecOps programs that have not been successful. This is not a reflection of the technological approach, but rather the complexity of building software in the Department with many authorities and decision makers that determine the fate of a program.
- Cultural shift: There needs to be a cultural shift towards embracing agility, collaboration, and continuous improvement. This includes breaking down silos, fostering cross-functional teams, and promoting a culture of innovation and learning.

Workforce Development:

- Training and education: Provide training programs and resources to upskill the workforce in modern software practices, including Agile methodologies, DevSecOps principles, and relevant technologies.
- Career paths: Establish clear career paths and development opportunities for software professionals, including roles specific to Agile and DevSecOps, to attract and retain top talent.

Management and Governance:

- "Top-down" and "bottom-up:" The management objective of DevSecOps must be both "top-down" and "bottom-up" to balance the larger strategic goals of software modernization across the DoD. Senior leader buy-in is crucial for success, though buy-in at the staff level is equally important. This engenders a sense of ownership, which encourages the appropriate implementation of processes related to governance and enables team members to support continuous process improvement.

- Governance frameworks: Develop governance frameworks that align with modern software practices, ensuring compliance, risk management, and accountability while enabling agility and innovation.

- Inherent Accountability: Push down or delegate responsibility to the lowest level for more effective and efficient decision-making.

Collaboration and Partnerships:

- Cross-functional collaboration: Foster collaboration and communication between development, security, operations, end users, and other relevant stakeholders to break down silos and promote shared responsibility for software development, security, and capability delivery.

- Public-private partnerships: Establish partnerships with industry, academia, and other government agencies to leverage best practices, share knowledge, and drive innovation in software development.

Continuous Improvement and Measurement:

- Metrics and feedback loops: Define and track key performance indicators (KPIs) to measure the effectiveness of modern software practices and identify areas for improvement. Establish feedback loops to gather insights from users, stakeholders, and the development teams to drive continuous improvement.

By adopting a whole-of-DoD approach, the Department can create an environment that supports the successful implementation of modern software practices. This approach ensures alignment across the organization, promotes collaboration, and enables the DoD to deliver secure, resilient, and innovative software capabilities at the speed of relevance.

4.3 Measuring Success with Performance Metrics

Metrics are crucial to software development, delivery, and operations as they provide objective and quantifiable measures of performance, progress, and quality. By tracking and analyzing metrics, organizations can gain valuable insights into the efficiency, effectiveness, and health of their software development processes. Metrics help identify bottlenecks, measure productivity, and assess the impact of process improvements. They enable teams to make data-driven decisions, set realistic goals, and continuously improve their practices. Metrics also facilitate communication and transparency, allowing stakeholders to understand the status of projects, identify risks, and make informed decisions. Ultimately, metrics empower organizations to optimize their software development and delivery processes, enhance customer satisfaction, and achieve better outcomes.

Aggregation of performance metrics is accomplished by querying data from multiple tools across the software factory and is most often accomplished using existing tool APIs. Most git repositories have documented APIs to query how often an artifact is pulled from a main branch, for example. If you have successfully automated deployment, you can pull the deployment metrics from your automation stack.

4.3.1 DORA and Google Metrics

The 2023 State of DevOps Report, compiled by DevOps Research and Assessment (DORA) and Google Cloud, investigates the capabilities and measures of high-performing technology-driven organizations. The DORA research for 2023 looked at the following:

- Organizational performance – The organization should produce not only revenue, but value for customers, as well as for the extended community.
- Team performance – The ability for an application or service team to create value, innovate, and collaborate.
- Employee well-being – The strategies an organization or team adopts should benefit the employees – reduce burnout, foster a satisfying job experience, and increase people's ability to produce valuable outputs.

They also researched performance measures for the following:

- Software delivery performance – Teams can safely, quickly, and efficiently change their technology systems. The following four metrics measure the speed and stability of software delivery:
    - Deployment Frequency – how frequently changes are pushed to production.
    - Change Lead Time – how long it takes a change to go from committed to deployed.
    - Change Failure Rate – how frequently a deployment introduces a failure that requires immediate intervention.
    - Failed Deployment Recovery Time – how long it takes to recover from a failed deployment.
- Operational performance – The service provides a reliable experience for its users. This is the extent to which a service can meet the expectations of its users and includes measures like availability and performance.

  Another set of metrics that have value for measuring day to day operations are *Google's Four Golden Signals*.[7] These metrics have a higher value for platform operators than for developers or security team members. However, they are very useful for developers to measure application performance and for security teams to identify abnormalities that may point to a concern.
    - Latency – the time it takes to service a request.
    - Traffic – how much demand is being placed on your system.
    - Errors – rate of requests that fail.
    - Saturation – how "full" is your service (storage, compute, and bandwidth).

[7] The Four Golden Signals, [Online]. Available: https://sre.google/sre-book/monitoring-distributed-systems/#xref_monitoring_golden-signals.

- Reliability targets – The extent to which a service meets its stated goals for measures like availability, performance, and correctness.

OUSD(A&S) has defined the DoD4 which is an expansion of the DORA 4 and includes the following:

- Delivery Speed and Cadence: Lead Time and Deployment Frequency
- Stability & Reliability: MTTR, Deployment Failure Rate
- Value/ROI: Value produced (and associated costs) – tailored to mission threads/stakeholders
- Cyber Resilience: Time to detect/resolve cyber event and average time to achieve ATO

Additional metrics guidance is available from OUSD(A&S) at https://aaf.dau.edu/wp-content/uploads/2022/08/Agile-Metrics-Guide.pdf. Software Acquisition Pathway metrics are available at https://aaf.dau.edu/aaf/software/metrics-and-reporting/.

4.3.2 Objectives and Key Results (OKRs)

OKR stands for Objectives and Key Results. OKRs are a methodology for settings goals and tracking results. It started with Peter Drucker who introduced Management by Objectives in 1954. It was improved upon by Andy Grove at Intel in the 1970s by adding the Key Results. Google adopted OKRs in 1999. The goal of OKRs is to focus on what matters. A current DoD example using OKRs is Platform One. Some metrics from Platform One include the following:

- Objective: Expand the delivery of Platform One products and services to the warfighter.
    - Result: 100% automatic Big Bang edge instance deployments and upgrades via cloud distribution.
- Objective: Win the hearts and minds of current customers through enhanced product experiences.
    - Result: Reduce Big Bang release cycle from 2 weeks to 24 hours with upgrade paths avoiding customer downtime.
    - Result: Consolidate CNAP clusters 50% to increase maintainability and decrease attack surface.

4.3.3 Flow Metrics

Flow metrics are a set of quantitative measures used to assess the flow of work through a software development or delivery process. They provide insights into the efficiency and effectiveness of the process, helping teams identify bottlenecks, optimize workflow, and improve overall productivity. Flow metrics focus on the movement of work items from one stage to another, tracking the time taken at each stage and the rate at which work items progress through the process.

Some common flow metrics include the following:

1. Cycle Time: Cycle time measures the time taken for a work item to move from the start to the end of the process. It provides an indication of how quickly work is completed and can help identify areas where delays occur.

2. Throughput: Throughput measures the rate at which work items are completed within a given time period. It provides an understanding of the team's capacity to deliver work and can help identify potential bottlenecks or constraints.

3. Work in Progress (WIP): WIP measures the number of work items that are actively being worked on at any given time. It helps teams understand the amount of work they have in progress and can be used to manage and balance workloads.

By tracking and analyzing flow metrics, teams can identify areas for improvement, optimize their processes, and make data-driven decisions to enhance productivity and delivery speed. Flow metrics are often used in conjunction with other metrics and practices, such as Kanban or Lean methodologies, to create a more efficient and streamlined software development or delivery process.

4.4 Workforce

A critical component in institutionalizing software practices like DevSecOps is people. There must be a skilled workforce to power software factories and DevSecOps platforms.

A software product team typically consists of a group of individuals who work together to develop and deliver a specific product or feature. The team is typically cross-functional, meaning that each member brings a unique skillset or expertise. This allows for efficient collaboration, quick decision-making, and the ability to deliver a product that meets customer needs within a shorter timeframe. A small team size promotes a sense of ownership, accountability, and close collaboration among team members, leading to a more agile and responsive development process (see Figure 13). On highly functional teams, a given individual might execute in multiple roles, and that the nature of a cross-functional team means that this is not only allowable, but to be encouraged.

1. **Product Manager:** The product manager is responsible for defining the product vision, strategy, and roadmap. They gather requirements, prioritize features, and ensure that the product aligns with customer needs and business goals.

2. **Software Developer:** Software Developers are responsible for writing code and developing the software product. They work closely with other team members to implement features, fix bugs, and ensure the product meets quality standards.

3. **UX/UI Designer:** The UI designer focuses on creating a user-friendly and visually appealing interface for the product. The UX designer manages the user experience of a product focused on human factors by making products intuitive and maximizing usability, accessibility, and simplicity.

4. **Security Engineer:** Security Engineers are responsible for analysis and development of systems/software security throughout the product lifecycle to include integration, operations, and maintenance.

5. **Test Engineer**: Test Engineers plan, prepare, and perform testing, evaluation, verification, and validation of software to evaluate results against specifications, requirements, and operational need.

6. **Data Scientist/Engineer:** Data Scientists/Engineers gather and analyze data to provide insights into user behavior, product performance, and market trends. They help inform decision-making and drive data-driven improvements to the product.

7. **DevSecOps Specialist/Site Reliability Engineer:** DevSecOps Specialists/Site Reliability Engineers focus on the integration of development and operations processes. They select/deploy/maintain the set of CI/CD tools and processes used by the development team and/or maintain the deployed software product and ensure observability and security across the lifecycle.

8. **Software/Cloud Architect:** Software/Cloud Architects manage and identify product high-level technical specifications, which may include application design, cloud computing strategy and adoption, and integration of software applications into a function system to meet requirements.

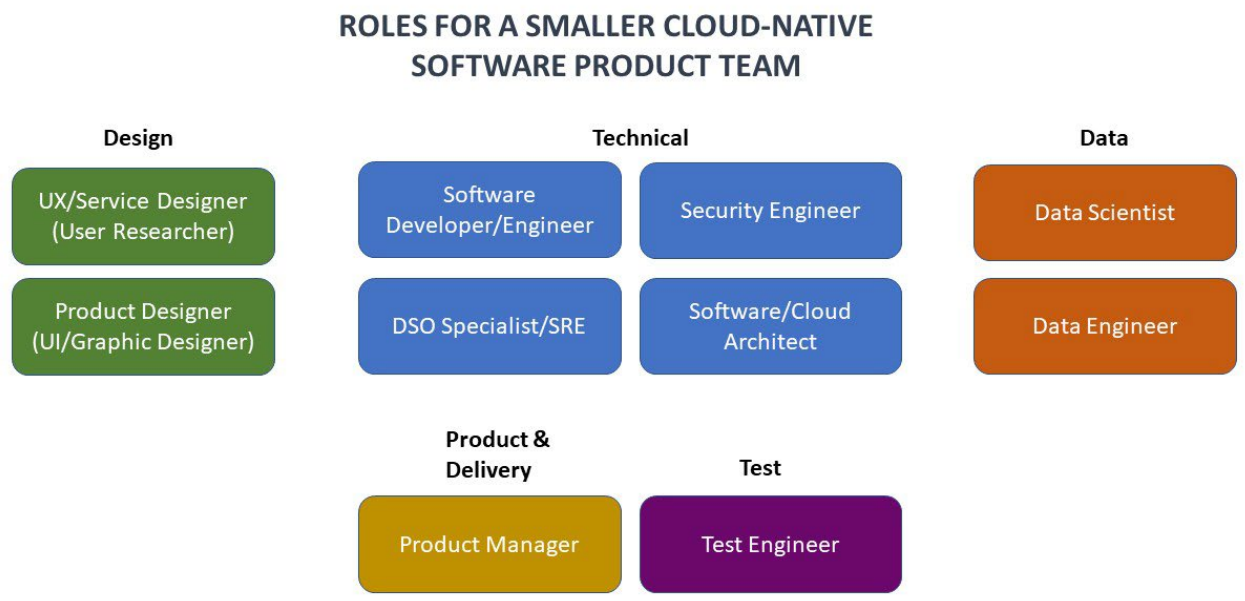

ROLES FOR A SMALLER CLOUD-NATIVE SOFTWARE PRODUCT TEAM
Design
UX/Service Designer (User Researcher)
Product Designer (UI/Graphic Designer)
Technical
Software Developer/Engineer
Security Engineer
DSO Specialist/SRE
Software/Cloud Architect
Data
Data Scientist
Data Engineer
Product & Delivery
Product Manager
Test
Test Engineer

Figure 13: Roles for a Smaller Product Team

Depending on the specific needs of the product, the team may also include and is not limited to other roles such as an Agile Coach, business analyst, technical writer, or customer support representative.

There are additional roles that may impact the software product team. They include the Warfighter/User, Mission Owner, Authorizing Official, Program Executive Officer (PEO), Program Manager, Sponsor, Financial Manager, Cost Analyst, Contracting Officer, Human Resources Specialist, Product Support Manager, and Intel/Threat Analyst.

4.5 DevSecOps Community

In April 2019, DoD conducted the first monthly DoD Enterprise DevSecOps Community of Practice (DSO CoP). The DSO CoP is a monthly meeting where practitioners from across DoD, academia, and industry gather to learn, exchange success stories and innovative ideas, educate members, raise issues to the greater DevSecOps community, and collaborate to resolve issues. The community reduces silos and enhances knowledge building across the Department by encouraging participation and partnerships and by staying connected to centers of excellence and pioneer implementers. To become a member of the DSO CoP, contact: osd.mc-alex.dod-cio.mbx.devsecops@mail.mil.

5 Getting Started

This document introduces DevSecOps – its definition, implementation, and institutionalization.

Adoption of DevSecOps and its methodologies and technical tools must be likened to a marathon, *not a sprint*. In more direct terms, comprehension of this document's material is the starting point, not the destination. Practitioners must demonstrate an insatiable thirst for knowledge to accept constructive criticism from their teammates, and indirectly from their individual and team performance metrics. Continuous improvement is just that – *continuous*. Keep your journey going and review these materials next at https://dodcio.defense.gov/library/

- DevSecOps Fundamentals Guidebook: DevSecOps Activities & Tools
- DoD Approved Reference Designs.

Acronyms

Acronym	Definition
A&A	Assessment and Authorization
ACAS	Assured Compliance Assessment Solution
AI	Artificial Intelligence
AO	Authorizing Official
API	Application Programming Interface
ATO	Authorization to Operate (ATO)
C2	Command and Control
CaC	Configuration as Code
cATO	continuous Authorization to Operate
CD	Continuous Deployment
CI	Continuous Integration
CIO	Chief Information Officer
CM	Configuration Management
CNAP	Cloud Native Access Point
COTS	Commercial Off the Shelf
CPT	Cyber Protection Team
CSP	Cloud Service Provider
CSSP	Cybersecurity Service Provider
DCIO	Deputy Chief Information Officer
DCO	Defensive Cyber Operations
DevSecOps	Development, Security, and Operations
DISA	Defense Information Systems Agency
DoD	Department of Defense
DoDI	DoD Instruction
DORA	DevOps Research and Assessments
DSO CoP	DevSecOps Community of Practice
DT&E	Developmental Test and Evaluation
FIPS	Federal Information Processing Standard
FOSS	Free and Open-Source Software
GOTS	Government Off the Shelf
HBSS	Host Based Security System
HWIL	Hardware-in-the-Loop
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IDE	Integrated Development Environment

IE	Information Enterprise
IL	Impact Level
IO	Input/Output
ISCM	Information Security Continuous Monitoring
IT	Information Technology
JFHQ-DODIN	Joint Force Headquarters – DoD Information Network
JTIC	Joint Integration Test Center
ML	Machine Learning
mTLS	mutual Transport Layer Security authentication
MVCR	Minimum Viable Capabilities Release
MVP	Minimum Viable Product
NIST	National Institute of Standards and Technology
NPE	Non-Person Entity
OKR	Objectives and Key Results
OT&E	Operational Test and Evaluation
PA	Provisional Authorization
PaaS	Platform as a Service
PM	Program Manager
QA	Quality Assurance
RMF	Risk Management Framework
SaaS	Software as a Service
SAST	Static Application Security Test
SP	Special Publication
SSDF	Secure Software Development Framework
STIG	Security Technical Implementation Guide
SWE	Software Engineering
SWIL	Software-in-the-Loop
T&E	Testing and Evaluation
UI	User Interface
UX	User Experience

Glossary of Key Terms

Following are the key terms used in this document.

Term	Definition
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.
CI/CD Pipeline	The process workflows and associated tools to achieve the continuous integration and continuous deployment of software with maximum use of automation.
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 Authorization to Operate	cATO is the state achieved when the organization that develops, secures, and operates a system has demonstrated sufficient maturity in their ability to maintain a resilient cybersecurity posture that traditional risk assessments and authorizations become redundant. This organization must have implemented robust information security continuous monitoring capabilities, active cyber defense, and secure software supply chain requirements to enable continuous delivery of capabilities without adversely impacting the system's cyber posture
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).
Control Gate	A control gate is a point in the software lifecycle process when the code is evaluated, and a decision is made to proceed or stop progress. Control gates can contain automated and manual checks.
Delivery	The process by which a released software is placed into an 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 combination of software engineering methodologies, practices, and tools that unifies software development (Dev), security (Sec), and operations (Ops). It emphasizes collaboration across these disciplines, along with automation and continuous monitoring to support the delivery of secure, high-quality software. DevSecOps integrates security tools and practices into the development pipeline, emphasizes the automation of processes, and fosters a culture of shared responsibility for performance, security, and operational integrity throughout the entire software lifecycle, from development to deployment and beyond.
DevSecOps Platform	A DevSecOps platform is defined as a group of resources and capabilities that form a base upon which other capabilities or services are built and operated within the same technical framework. Use of a DevSecOps platform is encouraged to accelerate development, delivery, and cybersecurity accreditation
Embedded Software	Software with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints, or software applications embedded in a platform (e.g., air vehicle, ground vehicle, or ship). DoDI 5000.87
Immutable infrastructure	Immutable infrastructure refers to computer infrastructure (virtual machines, containers, network appliances) that cannot be changed once deployed. Containers are a good example because persistent changes to containers can only be made by creating a new version of the container or recreating the existing container from its image. (from: https://glossary.cncf.io/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.
Iron Bank	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).
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.
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.
Repository / repo	A central place in which data is aggregated and maintained in an organized way.
Software Artifact	A software artifact is a tangible machine-readable document created during software development. (ISO/IEC/IEEE 24765:2017)
Software Factory	A software factory is defined as a collection of people, tools, and processes that enables teams to continuously deliver value by deploying software to meet the needs of a specific community of end users. It leverages automation to replace manual processes.
Software Supply Chain	The software supply chain is a collection of steps that create, transform, and assess the quality and policy conformance of software artifacts. (NIST SP 800-204D)