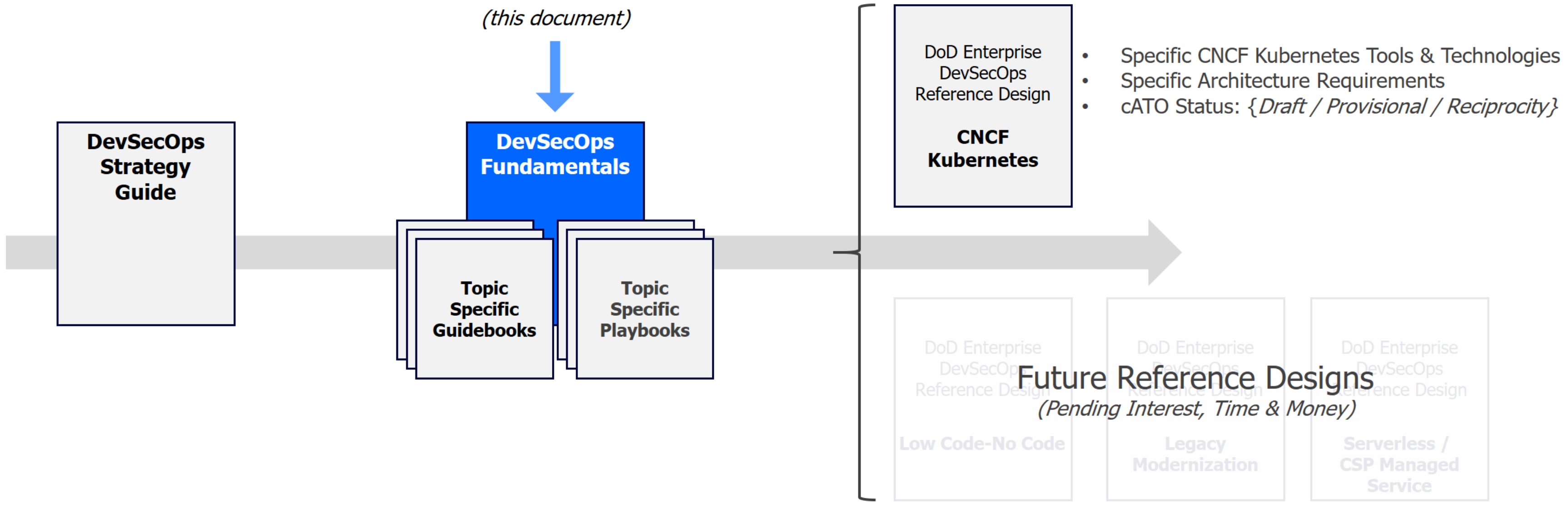
DoD Enterprise DevSecOps Fundamentals

March 2021

Version 2.0

Document Set Reference


(this document)
DevSecOps Strategy Guide
DevSecOps Fundamentals
Topic Specific Guidebooks
Topic Specific Playbooks
DoD Enterprise DevSecOps Reference Design
CNCF Kubernetes
Specific CNCF Kubernetes Tools & Technologies
Specific Architecture Requirements
cATO Status: {*Draft / Provisional / Reciprocity*}
DoD Enterprise DevSecOps Reference Design
Low Code-No Code
DoD Enterprise DevSecOps Reference Design
Legacy Modernization
DoD Enterprise DevSecOps Reference Design
Serverless / CSP Managed Service
Future Reference Designs
(Pending Interest, Time & Money)

Document Approvals

Approved by:

[illegible]

John B. Sherman

Chief Information Officer of the Department of Defense (Acting)

Approved by:

[illegible]

Stacy A. Cummings

Principal Deputy Assistant Secretary of Defense (Acquisition)

Performing the Duties of Under Secretary of Defense for Acquisition and Sustainment

Trademark Information

Contents

Document Approvals....3
Trademark Information....4
Introduction....7
Agile....8
The Agile Manifesto....8
Psychological Safety....8
Software Supply Chains....9
Value of a Software Factory....9
Software Supply Chain Imperatives....13
Development Imperatives....13
Security Imperatives....13
Operations Imperatives....14
DevSecOps....16
Overview....16
DevSecOps Culture & Philosophy....18
DevSecOps Cultural Progression....19
Zero Trust in DevSecOps....20
Behavior Monitoring in DevSecOps....20
DevSecOps Lifecycle....21
Cybersecurity Testing at Each Phase....21
Importance of the Plan Phase....23
Clear and Identifiable Continuous Feedback Loops....24
Continuous Build....24
Continuous Integration....24
Continuous Delivery....25
Continuous Deployment....25
Continuous Operations....26
Continuous Monitoring....26
DevSecOps Platform....28
DevSecOps Platform Conceptual Model....30
Current and Envisioned DevSecOps Software Factory Reference Designs....31
CNCF Kubernetes Architectures....31
Low Code/No Code and RPA Architectures....31

Serverless Architectures 31
CSP Managed Service Provider Architectures 31
Minimal DevSecOps Tools Map 32
Architecture Agnostic Minimal Common Tooling 32
Measuring Success with Performance Metrics 34
DevSecOps Next Steps 36
Appendix A Acronym Table 37
Appendix B Glossary of Key Terms 40

Figures

Figure 1 Notional Software Supply Chain 10
Figure 2 Normative Software Factory Construct 11
Figure 3 Maturation of Software Development Best Practices 16
Figure 4 DevSecOps Distinct Lifecycle Phases and Philosophies 17
Figure 5 "Unfolded" DevSecOps Lifecycle Phases 21
Figure 6 Notional expansion of a single DevSecOps software factory pipeline 22
Figure 7 Control Gate Goals 23
Figure 8 Continuous Build Feedback Loop 24
Figure 9 Continuous Integration Feedback Loop 24
Figure 10 Continuous Delivery Feedback Loop 25
Figure 11 Continuous Deployment Feedback Loop 25
Figure 12 Continuous Operations Feedback Loop 26
Figure 13 Continuous Monitoring Phase and Feedback Loop 27
Figure 14 Notional DevSecOps Platform with Reference Design Interconnects Identified 29
Figure 15 DevSecOps Conceptual Model with Cardinalities Defined 30
Figure 16 Notional Hypervisor Construct 44

Introduction

This document is intended as an educational compendium of universal concepts related to DevSecOps, including normalized definitions of DevSecOps concepts. Other pertinent information is captured in corresponding topic-specific guidebooks or playbooks. Guidebooks are intended to provide deep knowledge and industry best practices with respect to a specific topic area. Playbooks consist of one-page *plays*, structured to consist of a best practice introduction, salient points, and finally a checklist or call-to-action.

The intended audience of this document includes novice and intermediate staff who have recently adopted or anticipate adopting DevSecOps. The associated guidebooks and playbooks provide additional education and insight. Expert practitioners may find value in this material as a refresher.

This document and its topic-specific guidebooks/playbooks are intended to be educational.

Section 1: Agile principle adoption across the DoD continues to grow, but it is not ubiquitous by any measure. This document presents an informative review of *Agile* and agile principles.

Section 2: Includes a review of *software supply chains*, focusing on the role of the software factory within the supply chain, as well as the adoption and application of DevSecOps cultural and philosophical norms within this ecosystem. Development, security, and operational imperatives are also captured here.

Section 3: Building on the material covered in sections 1 and 2, this section includes an in-depth explanation of DevSecOps and the DevSecOps lifecycle to include each phase and related continuous process improvement feedback loops.

Section 4: Includes current and potential DoD Enterprise DevSecOps Reference Designs. Each reference design is fully captured in its own separate document. The minimum set of material required to define a DevSecOps Reference Design is also defined in this section.

Section 5: Performance metrics are a vital part of both the software factory and DevSecOps. Specific metrics are as-yet undefined, but pilot programs are presently underway to evaluate what metrics make the most sense for the DoD to aggregate and track across its enormous portfolio of software activities. This section introduces a number of well-known industry metrics for tracking the performance of DevSecOps pipelines to create familiarity.

Agile

The Agile Manifesto

The Agile Manifesto captures core competencies that define the functional relationship and what a DevSecOps team should value most:[1]

- Individuals and interactions *over* processes and tools
- Working software *over* comprehensive documentation
- Customer collaboration *over* contract negotiation
- Responding to change *over* following a plan

The use of the phrase *over* is vital to understand. The manifesto is not stating that there is no value in processes and tools, documentation, etc. It is, however, stating that these things should not be emphasized to a level that penalizes the other.

The first principle regarding *Individuals and interactions over processes and tools* explicitly speaks to DevSecOps. The ability of a cross-functional team of individuals to collaborate together is a stronger indicator of success than the selection of specific tooling or processes. This ideal is further strengthened by the 12 principles of agile software, particularly the principle that reinforces the priority for early and often customer engagement.[2]

Psychological Safety

New concepts inherently come with a degree of skepticism and uncertainty. Within the DoD, DevSecOps is a new concept, and the entire span of our workforce, from engineering talent, to acquisition professionals, through our leadership have many questions on this topic. The success of the commercial industry in using these practices has been widely documented.[3] There are leaders who want DevSecOps, but cannot tell if they are already practicing DevSecOps, or how to effectively communicate their practices if they do. Acquisition professionals routinely struggle to understand how to effectively buy services predicated upon DevSecOps due to the perception that it is hard to put tangible frames around and a price tag on something seemingly conceptual. Skepticism and uncertainty can also drive undesirable actions and reactions across the DoD, such as bias and fear. It is human nature to instinctively fall back on life experiences in an attempt to bring experiential knowledge to an unfamiliar situation. When this happens, we unknowingly insert bias into decision making processes and understanding. When this happens, this must be recognized and corrected.

Taxpayers reasonably expect an evaluation of investments, specifically if an appropriate level of value will be created given the investment of time, resources, and money spent. Across the Department the status quo is too often maintained because of the *sunk cost fallacy.*[4] The

[1] Beck, K. *et. al.*, 2001. Manifesto for Agile Software Development. [Online]. Available at: https://agilemanifesto.org.
[2] Beck, K. *et. al.*, 2001. Manifesto for Agile Software Development. [Online]. Available at: https://agilemanifesto.org/principles.html.
[3] Defense Innovation Board (DIB), "Software Acquisition and Practices (SWAP) Study." May 03, 2019, [Online]. Available: https://innovation.defense.gov/software.
[4] Arkes, Hal R. & Blumer, Catherine, 1985. "The psychology of sunk cost," Organizational Behavior and Human Decision Processes, Elsevier, vol. 35(1), pages 124-140, February.

DevSecOps journey can be a positive transformational journey, but only if we are acutely aware of bias towards psychological safety when working towards critical decisions.

Software Supply Chains

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), technology force multipliers, and tools and practices that are brought together to deliver specific software capabilities. A notional software supply chain, depicted in *Figure 1*, is the recognition that software is rarely produced in isolation, and a vulnerability anywhere within the supply chain of a given piece of code could create an exposure or worse, a compromise. Hardware, infrastructure, platforms and frameworks, Software as a Service, technology force multipliers, and especially the people and processes come together to form this supply chain.

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 because of the people, processes, and tools that created 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.

Value of a Software Factory

A normative software factory construct, illustrated in *Figure 2*, contains multiple **pipelines**, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The environments that are set up in the software factory should be rehydrated using **Infrastructure as Code (IaC)** and **Configuration as Code (CaC)** that run on various tools. A software factory must be designed for multi-tenancy and automate software production for multiple products. A DoD organization may need multiple pipelines for different types of software systems, such as web applications or embedded systems.

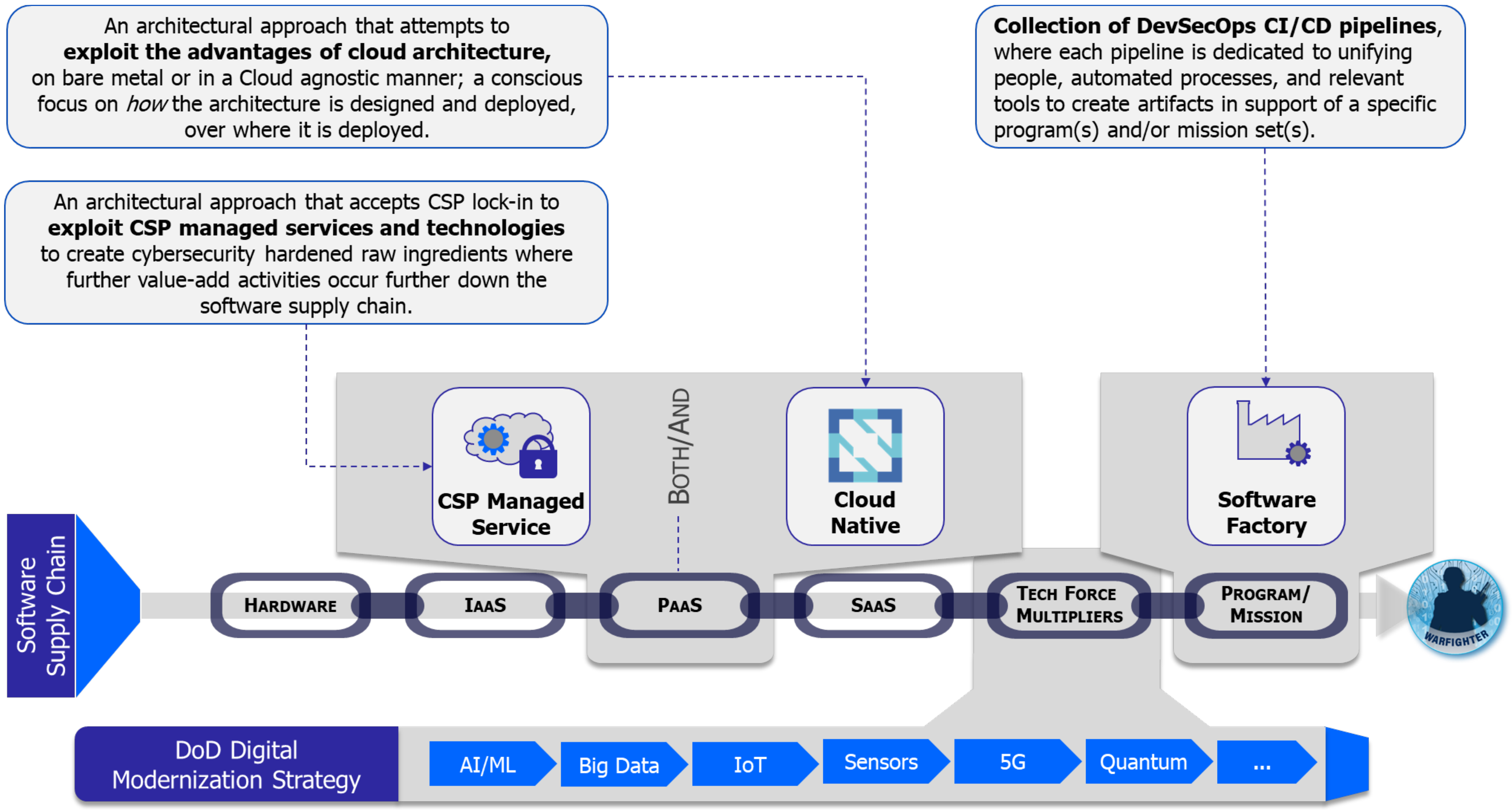

An architectural approach that attempts to **exploit the advantages of cloud architecture,** on bare metal or in a Cloud agnostic manner; a conscious focus on *how* the architecture is designed and deployed, over where it is deployed.
An architectural approach that accepts CSP lock-in to **exploit CSP managed services and technologies** to create cybersecurity hardened raw ingredients where further value-add activities occur further down the software supply chain.
Collection of DevSecOps CI/CD pipelines, where each pipeline is dedicated to unifying people, automated processes, and relevant tools to create artifacts in support of a specific program(s) and/or mission set(s).
CSP Managed Service
BOTH/AND
Cloud Native
Software Factory
Software Supply Chain
HARDWARE
IAAS
PAAS
SAAS
TECH FORCE MULTIPLIERS
PROGRAM/ MISSION
WARFIGHTER
DoD Digital Modernization Strategy
AI/ML
Big Data
IoT
Sensors
5G
Quantum
...

Figure 1 Notional Software Supply Chain

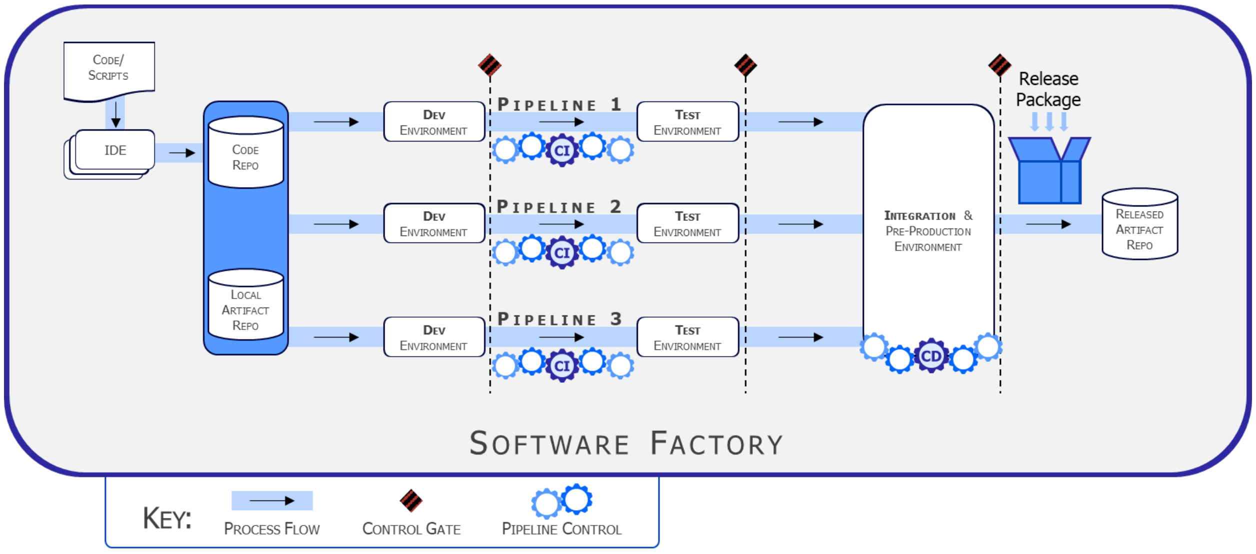


Figure 2 Normative Software Factory Construct

Each factory is expected be instantiated from hardened IaC code and scripts or DoD hardened containers from the sole DoD artifact repository, **Iron Bank**. In the case of a CNCF Certified Kubernetes powered software factory, the core services must also come from DoD hardened containers pulled from Iron Bank. Once the software factory is up and running, the developers predominately use their IDEs to begin creating their custom software artifacts, using the services offered by the specific software factory.

Every bit of free and open software source (FOSS), Commercial off the Shelf (COTS), Government off the Shelf (GOTS), and/or newly developed code and supporting scripts, configuration files, etc. are committed into the factory's local artifact repo or code repository, respectively. With each commit to the code repository, the assembly line automation kicks in. There are multiple continuous integration / continuous delivery (CI/CD) pipelines executing in parallel, representing different unique artifacts being produced within the factory.

The adoption of CI/CD pipelines reduce risk by making many small, incremental changes instead of a "big bang" change. The incremental changes of application code, infrastructure code, configuration code, compliance code, and security code can be reviewed quickly. Mistakes introduced are easier to capture and isolate when few things have changed.

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, the encounter a control gate. If the code is successfully compiled, it will forward with a pull/merge request for peer review, 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, it can reject the merge request and send the code back to the software developer for rework. Once the merge request is approved, and the merge completed, the continuous integration step is triggered.

Continuous integration executes unit tests, such as Static and Dynamic Application Security Test (SAST), verify 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 subsystem, including 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**.

The test environment usually is where a more in-depth set of tests are executed, for example, hardware-in-the-loop (HWIL) testing or software-in-the-loop (SWIL) testing may occur, especially when the hardware is too expensive or too bulky to provide to each individual developer to work against locally. In addition, the test environment performs additional or more in-depth testing variants of static code analysis, functional tests, interface tests, and dynamic code analysis. If all of 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 test, additional security compliance scans, etc. Once all of these 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.

In summary, the DevSecOps software factory provides numerous benefits, including:

- Rapid creation of hardened software development infrastructure for use by a DevSecOps team.
- A dynamically scalable set of pipelines with three distinct cyber survivability control gates.
- Operational Test & Evaluation is shifted left, moved into the CI/CD pipelines instead of bolted on the end of the process, facilitating more rapid feedback to the development teams.
- Simplified governance through the use of pre-authorized IaC scripts for the development environment itself
- Assurance as an Authorizing Official (AO) that functional, security, integration, operational, and all other tests are reliably performed and passed prior to formal release and delivery.

Software Supply Chain Imperatives

Evaluation of every software supply chain must consider a series of imperatives that span development, security, and operations – the pillars of DevSecOps. Regardless of the specific software factory reference design that is applied, there are a core set of imperatives that must always exist. These imperatives include:

- Use of agile frameworks and user-centered design practices.
- Baked-in security across the entirety of the software factory and throughout the software supply chain.
- Shifting cybersecurity left.
- Shifting both development tests and operational tests left.
- Reliance on IaC and CaC to avoid environment drifts between deployments.
- Use of a clearly identifiable CI/CD pipeline(s).
- Adoption of Zero Trust principles and a Zero Trust Architecture throughout, both north-south *and* east-west traffic.[5]
- Comprehension and transparency of lock-in decisions, with a preference for avoiding vendor lock-in.
- Comprehension and transparency of the cybersecurity stack, with a preference for decoupling it from the application workload.
- Centralized log aggregation and telemetry.
- Adoption of *at least* the DevOps Research and Assessment (DORA) performance metrics, defined in full in the section Measuring Success with Performance Metrics.

Additional imperatives across development, security, and operations should be considered.

Development Imperatives

- Favor small, incremental, and frequent updates over larger, more sporadic releases.
- Apply cross-functional skill sets of Development, Cybersecurity, and Operations throughout the software lifecycle, embracing a continuous monitoring approach in parallel instead of waiting to apply each skill set sequentially.
- With regard to legacy software modernization, lift & shift is a myth. Simply moving applications to the cloud for re-hosting by lifting the code out of one environment and shifting it to another is not a viable software modernization approach. True modernization will require applications be rebuilt to cloud-specific architectures, and DevSecOps will be fundamental in this journey.
- Continuously monitor environment configurations for unauthorized changes.
- Deployed components must always be replaced in their entirety, never update in place.

Security Imperatives

- Zero Trust principles must be adopted throughout.

[5] National Institute of Standards and Technology, "NIST Special Publication 800-207, Zero Trust Architecture." August, 2020.

- Configure control gates with explicit, transparently understood exit criteria tailored to meet the AO's specific risk tolerance.
- Ensure the log management and aggregation strategy meets the AO's specific risk tolerance.
- Support Cyber Survivability Endorsement (CSE) for the specific application and data, based on the DoD Cloud Computing Security Requirements Guide (SRG) and industry best practices.[6]

  NOTE: Teams should discuss and understand how the CSE is factored into technical design assessments, RFP source selection, and operational risk trade space decisions throughout the system's lifecycle. Early consideration of cyber survivability requirements can prevent the selection of foundationally flawed technology implementations (cost drivers) that are frequently rushed to market without incorporating best business practice development for cybersecurity and cyber resilience.

- Automate as much developmental and operational testing and evaluation (OT&E), including functional tests, security tests, and non-functional tests, as possible.
- Recognize that the components of the platform can be instantiated and hardened in multiple different ways, to include a mixture of these options:
  - Using Cloud Service Provider (CSP) managed services, providing quick implementation and deep integration with other CSP security services, but with the "cost of exit" that these services will have different APIs and capabilities on a different cloud, and are unavailable if the production runtime environment is not in the cloud (e.g., an embedded system on a weapons system platform).
  - Using hardened containers from a DoD authorized artifact repository, e.g. Iron Bank, to instantiate a CSP-agnostic solution running on a CNCF-compliant Kubernetes platform.
- Formal testing goes from testing each new environment to testing the code that instantiates the next new environment.
- Expressly define an access control strategy for privileged accounts; even if someone is privileged, that doesn't mean they need the authorization to be *turned on* 24x7.

Operations Imperatives

- Continuous monitoring is necessary and contextually related to the ThreatCon; what was a non-event last week may be a critical event this week.
- Only accept a "fail-forward" recovery. **Failing forward** recognizes that the time taken to roll out a deployment and revert to a prior version is often equivalent to the amount of time it takes for the software developer to fix the problem and push it through the automated pipeline, thus "failing forward" to a newer release that fixes the problem that existed in production.
- Recognize and adopt blue/green deployments when possible. A Blue/Green deployment exists when the existing production version continues to operate alongside the newer version being deployed, providing time for a minor subset of production traffic to be

[6] DISA, "Department of Defense Cloud Computing Security Requirements Guide, v1r3," Mar 6, 2017.

routed over to the newer version to validate the deployment, or for the development team to validate the deployment alongside security and operations peers. Once the newer version has been validated, 100% of the traffic can be routed to the new deployment, and the old deployment resources can be reclaimed.

- Recognize and adopt canary deployments for new features. A canary deployment is when a feature can be enabled or disabled via metadata. While every production instance has the capability, it is only enabled on a minor percentage of the cluster such that only a few users actually see the new capability. Through continuous monitoring of usage, the DevSecOps team can evaluate metrics and usefulness of the feature and determine if it should be made widely available, or if the feature needs to be re-worked.

DevSecOps

Overview

Software development best practices are ever-evolving as new ideas, new frameworks, new capabilities, and radical innovations become available. Over time we witness technological shifts that relegate what was once state-of-the-art to be described as legacy or deprecated. In wireless, 2.5G systems have been fully retired, 3G systems were aggressively replaced by 4G LTE with shutdown dates publicly announced, and now 4G LTE is being supplanted by the rise of 5G. Software is no different, and *Practices depicts* the *broad trends* over the last 30 years. Different programs and application teams may be more advanced in one aspect and lagging in another.

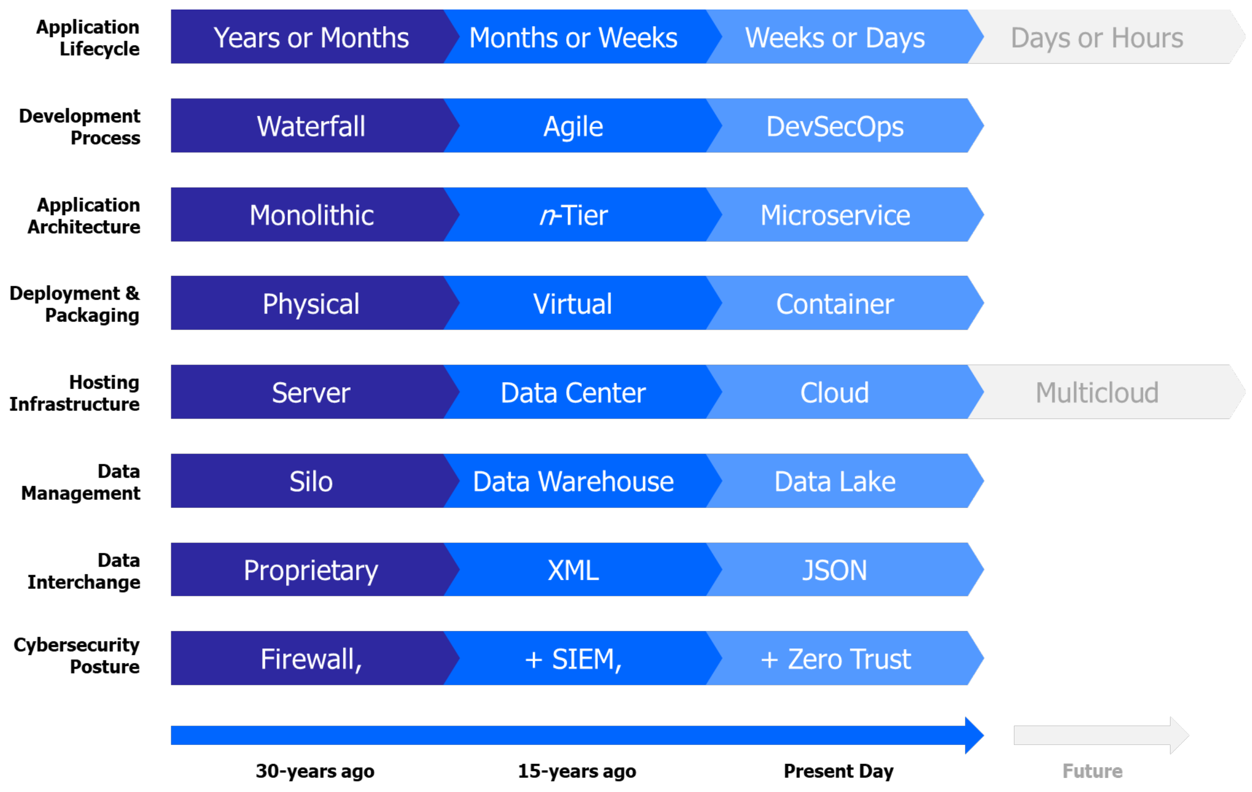


Figure 3 Maturation of Software Development Best Practices

While tightly coupled monolithic architectures were the norm, the growth of finely grained, loosely coupled microservices are now considered state-of-the-art and have evolved the Service Oriented Architecture (SOA) concept of services and modularity. Development timeframes have compressed, deployment models have shifted to smaller containerized packaging, and the cloud portends to deliver an endless supply of computing capacity as infrastructure for compute, storage, and network have shifted from physical to virtual to cloud.

The shift towards DevSecOps, microservices, containers, and Cloud necessitates a new approach to cybersecurity. Security must be an equal partner with the development of business and mission software capabilities, integrated throughout the phases between planning and production.

The alluring characteristics of DevSecOps is how it improves customer and mission outcomes through implementation of specific technologies that automate processes and aid in the *delivery of software at the speed of relevance*, a primary goal of the DoD's software modernization efforts. DevSecOps is a **culture and philosophy** that must be practiced across the organization, realized through the unification of a set of software development (Dev), security (Sec) and operations (Ops) personnel into a singular team. The DevSecOps **lifecycle** phases and philosophies, depicted in *Figure 4*, is an iterative closed loop lifecycle that spans eight distinct phases. Teams new to DevSecOps are encouraged to start small and build it up their capabilities, progressively, striving for continuous process improvement at each of the eight lifecycle phases.

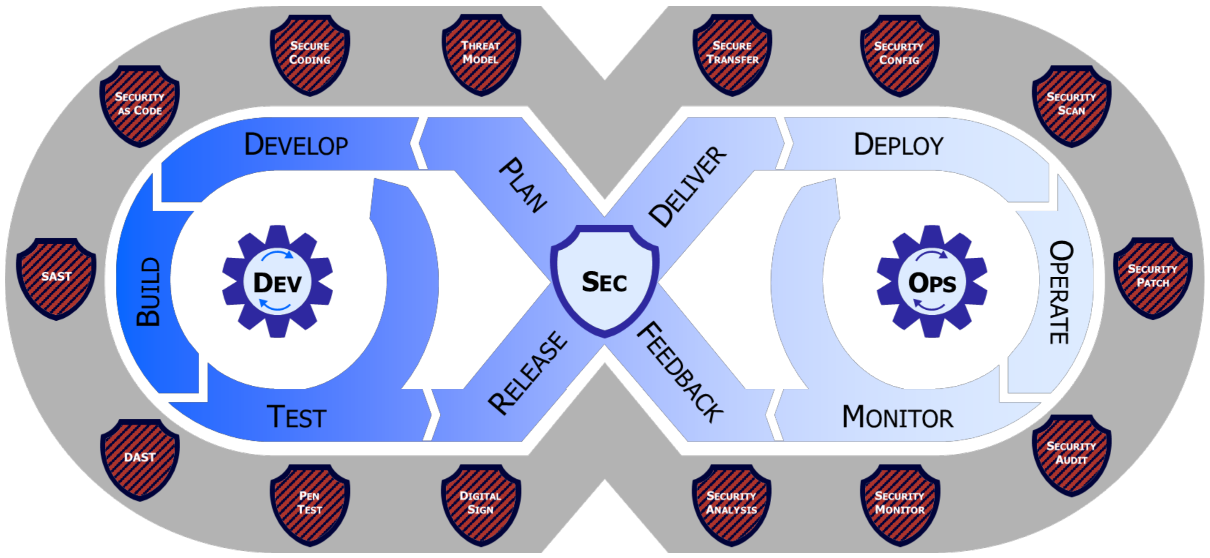


Figure 4 DevSecOps Distinct Lifecycle Phases and Philosophies

The Security value of DevSecOps is achieved through a fundamental change in the culture and approach to cybersecurity and functional testing. Security is continuously *shifted left* and integrated throughout the fabric of the software artifacts from day zero. This approach differs from the stale view that operational test and evaluation (OT&E) and cybersecurity can simply be bolt-on activities 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. The DevSecOps differentiation is realized fully only when security and functional capabilities are built, tested, and monitored at each step of the lifecycle, preventing the security and functional problems from reaching production in the first place.

Each of the shields surrounding the DevSecOps lifecycle in *Figure* 4 represents a distinct category of cybersecurity testing and activities. This blanket of protection is intentionally depicted *surrounding* the eight distinct phases of the DevSecOps lifecycle because these tests must permeate throughout the lifecycle to achieve benefits. Failure to weave security and functional testing into just one of the eight phases can create a risk of exposure, or worse, compromise, in the final production product.

Some people describe DevSecOps as emphasizing *cybersecurity over compliance*; this is recognition that *you can be compliant, but not secure*, and *you can be secure, but not compliant*. Sidestepping the debate on the fairness of this characterization of DevSecOps, the below considers the specific characteristics espoused by DevSecOps practitioners:

- Fully automated risk characterization, monitoring, and mitigation across the application lifecycle is paramount.
- Automation and the security control gates continuously evaluate the cybersecurity posture while still empowering *delivery of software at the speed of relevance*.
- Support meeting Cyber Survivability Endorsement (CSE) Cyber Survivability Attribute (CSA) number 10 – Actively Manage System's Configurations to Achieve and Maintain an Operationally Relevant Cyber Survivability Risk Posture (CSRP).[7]

There are additional benefits to the Government where specific choices in technology implementations and data collection can automate traditionally manual processes, such as risk characterization, monitoring and mitigation, across the entire application lifecycle. Another pain point felt by the Government relates to updating and patching systems, something that can be fully automated from a DevSecOps tech stack because of its emphasis on automation and control gates.

Strictly speaking, DevSecOps adoption does **not** require a specific architecture, containers, or even explicit use of a Cloud Service Provider (CSP). However, the use of these things are strongly recommended, and in some cases mandated by specific Reference Designs. The goal of software modernization and what DevSecOps must drive is the ability deliver resilient software capabilities at the speed of relevance through the release of incremental capabilities in a decoupled fashion.

DevSecOps Culture & Philosophy

DevSecOps, and its predecessor DevOps, is a culture and philosophy. DevSecOps builds upon the value proposition of DevOps by expanding its culture and philosophy to recognize that maximizing cyber survivability requires integrating cybersecurity practices throughout the entire systems development lifecycle (SDLC). DevSecOps advances the growing philosophy and sentiment that reliance upon bolt-on or standalone monolithic cybersecurity platforms is incapable of providing adequate security in today's operational environments. Cybersecurity tooling that is fully isolated from the development and operational environments are *reactive* at best, whereas integrated automated tooling with the software factory is *proactive*.

A proactive culture recognizes that it is better to detect and halt the deployment of a cybersecurity risk within the software factory pipelines instead of detecting it after the fact in production. Further the DevSecOps cybersecurity culture embraces another core Agile tenet that prefers work software over comprehensive documentation. Mounds of cybersecurity

[7] DoD, "Cybersecurity Test and Evaluation Guidebook, Version 2.0, Change 1," Feb 10, 2020

documentation do not offer an assurance that software is secure; automated tests and testing outputs continuously executed within the software factory pipelines captures meaningful, timely metrics that provide a higher level of assurance to AOs.

There are several key principles for a successful transition to a DevSecOps culture:

- Continuous delivery of small incremental changes.
- Bolt-on security is weaker than security baked into the fabric of the software artifact.
- Value open source software.
- Engage users early and often.
- Prefer user centered & 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.

DevSecOps Cultural Progression

As a program's DevSecOps culture matures, it should progress along "the three ways", as introduced in the book *The Phoenix Project*, and as described in the seminal book *The DevOps Handbook:*[8,9]

1. *First Way: Flow.* "The First Way enables fast left-to-right flow of work from Development to Operations to the customer. In order to maximize flow, we need to make work visible, reduce our batch sizes and intervals of work, build in quality by preventing defects from being passed to downstream work centers, and constantly optimize for the global goals. By speeding up flow through the technology value stream, we reduce the lead time required to fulfill internal or customer requests, especially the time required to deploy code into the production environment. By doing this, we increase the quality of work as well as our throughput, and boost our ability to out-experiment the competition. The resulting practices include continuous build, integration, test, and deployment processes; creating environments on demand; limiting work in process (WIP); and building systems and organizations that are safe to change." - [The DevOps Handbook].
2. *Second Way: Feedback.* "The Second Way enables the fast and constant flow of feedback from right to left at all stages of our value stream. It requires that we amplify feedback to prevent problems from happening again, or enable faster detection and recovery. By doing this, we create quality at the source and generate or embed knowledge where it is needed—this allows us to create ever-safer systems of work where problems are found and fixed long before a catastrophic failure occurs. By seeing problems as they occur and swarming them until effective countermeasures are in place, we continually shorten and amplify our

[8] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*, IT Revolution Press, 2013

[9] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*, IT Revolution Press, 2016

feedback loops, a core tenet of virtually all modern process improvement methodologies. This maximizes the opportunities for our organization to learn and improve." - [The DevOps Handbook].

3. *Third Way: Continual Learning and Experimentation.* "The Third Way enables the creation of a generative, high-trust culture that supports a dynamic, disciplined, and scientific approach to experimentation and risk-taking, facilitating the creation of organizational learning, both from our successes and failures. Furthermore, by continually shortening and amplifying our feedback loops, we create ever-safer systems of work and are better able to take risks and perform experiments that help us learn faster than our competition and win in the marketplace." - [The DevOps Handbook].

Zero Trust in DevSecOps

The DevSecOps ecosystem that includes the software factory and the intrinsic blending across development, security, and operational creates complexity. This complexity has outstripped legacy security methods predicated upon "bolt-on" cybersecurity tooling and perimeter defenses. Zero Trust must be the target security model for cybersecurity adopted by DevSecOps platforms and the teams that use those platforms.

There is no such as a singular product that delivers a zero trust architecture because zero trust focuses on service protection, *and* data, *and* may be expended to include the complete set of enterprise assets.[5] This means zero trust touches infrastructure components, virtual and cloud environments, mobile devices, servers, end users, and literally every part of an information technology ecosystem. To encompass all of these things, zero trust defines a series of *principles* that when thoughtfully implemented and practiced with discipline prevent data breaches and limit the internal lateral movement of a would-be attacker.

DevSecOps teams must consistently strive to bake in zero trust principles across each of the eight phases of the DevSecOps SDLC, covered in the next section. Further, DevSecOps teams must fully consider security from both the end user perspective and all non-person entities (NPEs). To illustrate several of these concept in a notional list, these NPEs include servers, the mutual transport layer security (mTLS) between well-defined services relying on FIPS compliant cryptography, adoption of *deny by default* postures, and understanding how all traffic, both north-south and east-west, is protected throughout the system's architecture.

Behavior Monitoring in DevSecOps

The software factory platform and the DevSecOps team will quickly establish a normative set of behaviors. To illustrate this point, a merge into the main branch should never occur without a pull request that includes two or more additional engineers reviewing the code for quality, purpose, and cybersecurity. There are two types of behavior monitoring that are required to support the ideals of Zero Trust, covered above, and to enhance the overall cyber survivability of the software factory platform, the software artifacts being produced within that factory, and the different environments linked to the software factory: Behavior Detection and Behavior Prevention. The idea of detection is to trigger an actionable and logged alert, possibly delivered through a ChatOps channel to the entire team that conveys *I saw something anomalous*. The idea of prevention goes a step further. It still triggers an actionable and logged alert, but it also either proactively prevents or immediately terminates anomalous behaviors and conveys *I inhibited something anomalous*. There are a multitude of technologies available to achieve behavior monitoring in a DevSecOps environment. At a minimum, teams must incorporate behavior detection, and they should aspire and drive to incorporate behavior prevention throughout the software factory and its environments.

DevSecOps Lifecycle

The DevSecOps software lifecycle is most often represented using a layout that depicts an infinity symbol, depicted previously in *Figure 4*. This representation emphasizes that the software development lifecycle is not a monolithic linear process. There are eight phases: plan, develop, build, test, release, deliver, deploy, operate, and monitor, each complimented by specific cybersecurity activities.

DevSecOps is iterative by design, recognizing that **software is never done**. The “big bang” 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 delivery. This lifecycle is adaptable and as discussed next, includes numerous feedback loops that drive continuous process improvements.

NOTE: The unfolded “infinity” DevSecOps diagram depicted in *Figure 5* is used to better illustrate the relationship between the lifecycle phases and the continuous feedback loops used to drive continuous process improvement.

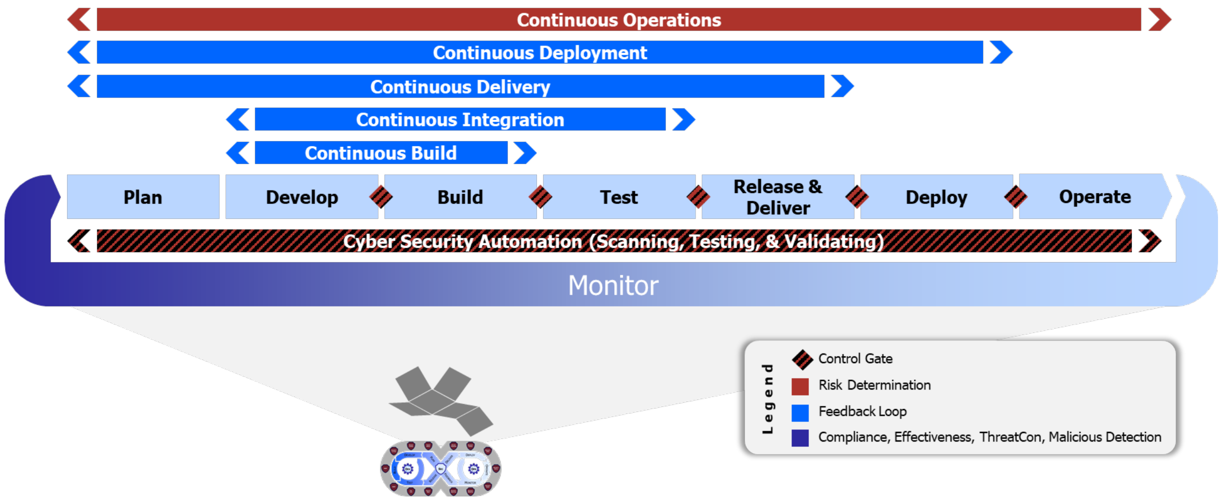


Figure 5 "Unfolded" DevSecOps Lifecycle Phases

Cybersecurity Testing at Each Phase

There is no “one size fits all” solution for cybersecurity testing design. Each software team has its own unique requirements and constraints. However, **the software artifact promotion control gates are a mandatory part of the software factory; their inclusion cannot be waivered away**. *Figure* 6 depicts where each of the mandatory control gates are within each of the software factory’s pipelines, depicted by the diamonds at the top of the graphic. The graphic depicts a notional and incomplete sampling of the types of tests at each gate; different pipelines within the software factory may define different collections of tests to maximize the effectiveness of a control gate.

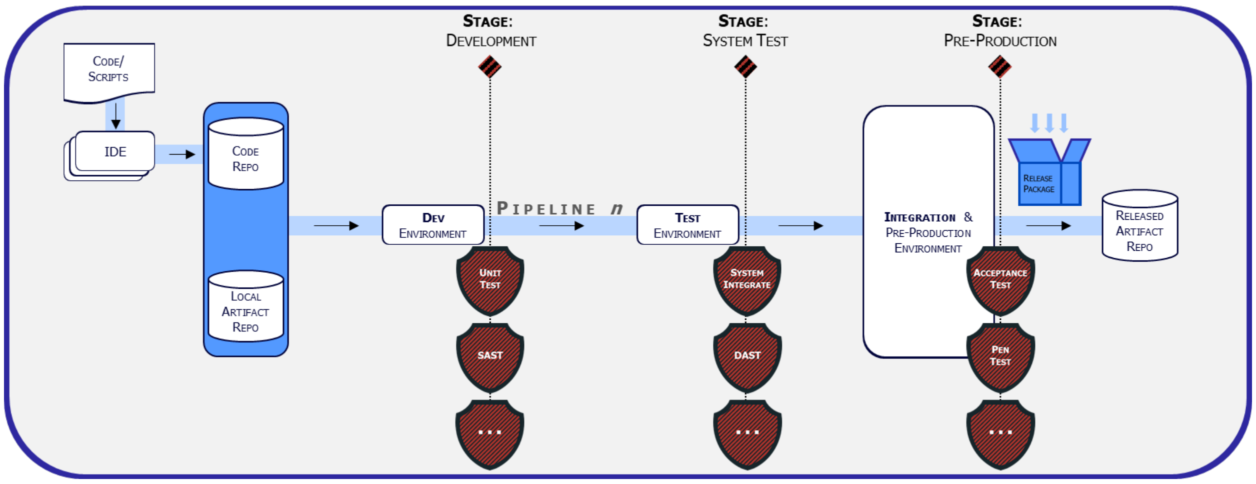


Figure 6 Notional expansion of a single DevSecOps software factory pipeline

The control gates are mandatory, but there is no expectation that they are fully automated from the moment the software factory is instantiated. On the contrary, because each program's requirements are unique, and as espoused by agile practices, it is expected that initially control gates may require human intervention. As the team matures through continuous process improvement, the team should identify repeatable actions and add automation of those actions into the team's backlog. The complete team must have strong confidence in the automation built at a control gate. To recap, as a best practice, start with more human intervention and gradually decrease human intervention in favor of repeatable automation as part of a continuous process improvement process.

One final note about the control gates; while they are described predominately as being cyber focused and preventing environmental and behavioral drift, it is vital to incorporate meaningful operational test & evaluation (OT&E) assessments when and where possible within the software factory pipelines. Shifting OT&E left into the software factory control gates is intended to accomplish the goals depicted in *Figure 7*.

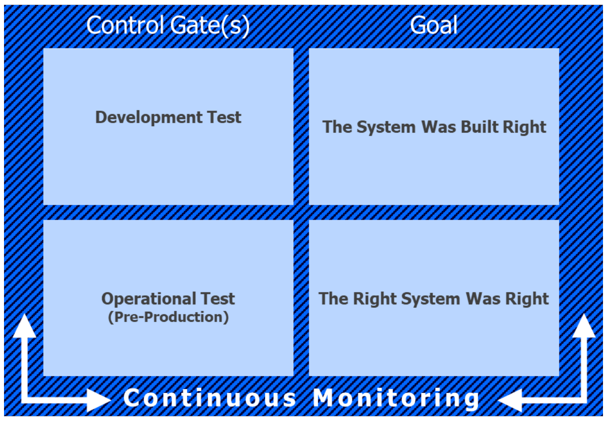


Figure 7 Control Gate Goals

Importance of the Plan Phase

The plan phase involves activities that help the team manage time, cost, quality, risk and issues. These activities include business-need assessment, project plan creation, feasibility analysis, risk analysis, business requirements gathering, business process creation, system design, DevSecOps design, and ecosystem instantiation.

The plan phase repeats ahead of each sprint iteration.

It is a best practice to develop a minimum viable product (MVP) for critical business needs first, then iterate to add features, more automation, more testing, etc. Use the continuous feedback loops, covered in the next section thoroughly, to implement continuous process improvement. This approach is recommended in the Lean Startup methodology.[10] The software factory encapsulates the DevSecOps processes and control gates, guiding the automation throughout the lifecycle as the team commits code. Rely upon the DevSecOps ecosystem tools to facilitate process automation and consistent process execution, recognizing the value in a continuous process improvement approach instead of a “big bang” approach.

The entirety of the DevSecOps team must have access to a set of communication, collaboration, project management, and change management tools. They may or may not be embedded within the software factory itself; in some cases, these tools may be procured as enterprise services. There is an explicit recognition that full automation *within* the plan phase is unrealistic, as the end users will collaborate with the team to establish a prioritized backlog. In other words, the backlog work cannot be automated. Teams must recognize the value of the planning tools in driving team interaction and collaboration, ideally increasing the team’s overall productivity during the plan phase.

[10] E. Reis, “The Lean Startup,” [Online]. Available: http://theleanstartup.com/principles. [Accessed 04 Feb 2021].

Clear and Identifiable Continuous Feedback Loops

The phases of the DevSecOps lifecycle rely upon six different continuous feedback loops. As originally presented and visualized in *Figure 5* "Unfolded" DevSecOps Lifecycle Phases, there are three control gates that exist *within* the CI/CD pipeline and two additional control gates.

Continuous Build

The Continuous Build feedback loop iterates between the *Develop* and *Build* phases of the DevSecOps lifecycle, depicted in *Figure 8*. 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, thus the importance of this feedback loop.

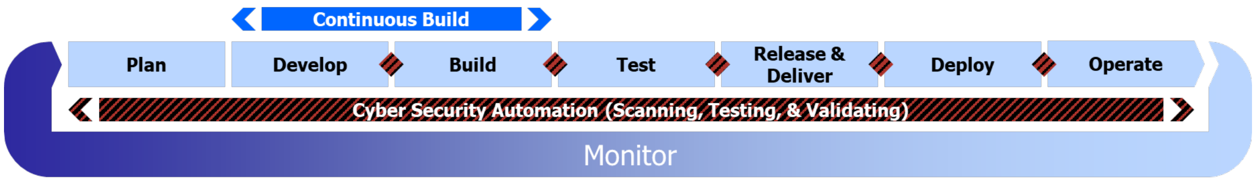


Figure 8 Continuous Build Feedback Loop

Common types of feedback in this loop 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 in this feedback loop 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.

Continuous Integration

The Continuous Integration (CI) feedback loop iterates across the *Develop*, *Build*, and *Test* phases of the DevSecOps lifecycle, depicted in *Figure 9*. Once the Continuous Build feedback loop completes and the pull request is merged into the main branch, a complete series of automated tests are executed, including a full set of integration tests.

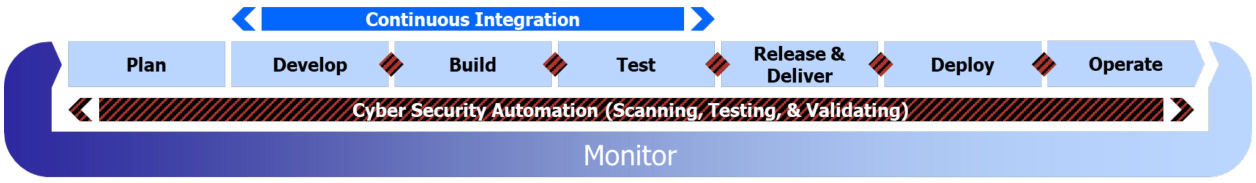


Figure 9 Continuous Integration Feedback Loop

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 automated build (including test) to detect integration errors as quickly as possible.[11] 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.

[11] M. Fowler, "Continuous Integration." May 01, 2006, [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

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, breaks an API, etc.

Continuous Delivery

The Continuous Delivery feedback loop iterates across the *Plan*, *Develop*, *Build*, *Test*, and *Release & Deliver* phases of the DevSecOps lifecycle, depicted in *Figure 10*. The most pertinent thing to understand is that *release and delivery* does not mean *pushed into production*. Continuous delivery acknowledges that a feature meets the Agile definition of "Done-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.

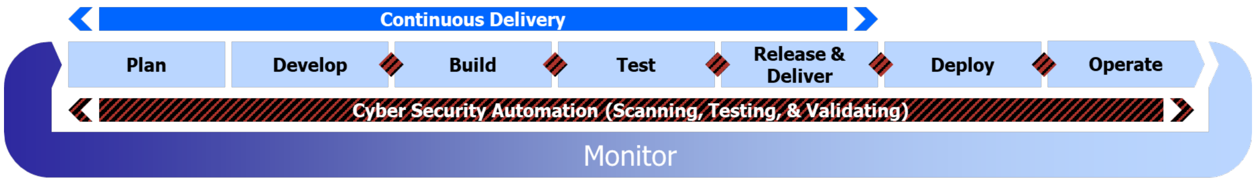


Figure 10 Continuous Delivery Feedback Loop

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.

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 delivery. In this document, CD is a software development practice that allows frequent releases of value to staging or various test environments once verified by automated testing. Continuous Delivery relies on a manual decision to deploy to production, though the deployment process itself should be automated. In contrast, continuous deployment is the automated process of deploying changes directly into production by verifying intended features and validations through automated testing.

Continuous Deployment

The Continuous Deployment feedback loop iterates across the *Plan*, *Develop*, *Build*, *Test*, *Release & Deliver*, and *Deploy* phases of the DevSecOps lifecycle, depicted in *Figure 11*. 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 factory's CI/CD pipeline, visualized in *Figure 6*.

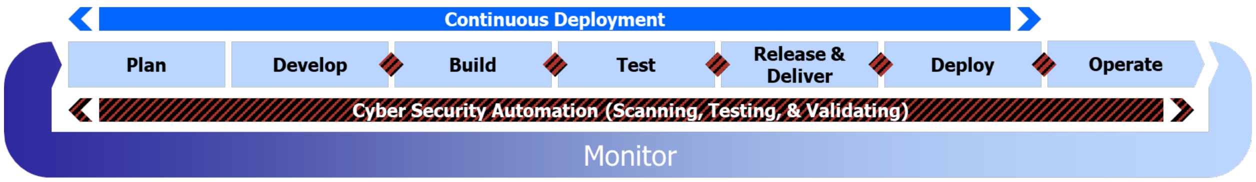


Figure 11 Continuous Deployment Feedback Loop

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 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*.

Continuous Operations

The Continuous Operations feedback loop iterates across the *Plan*, *Develop*, *Build*, *Test*, *Release & Deliver*, *Deploy*, and *Operate* phases of the DevSecOps lifecycle, depicted in *Figure 12*. Continuous operations are any activities focused on *availability, performance,* and *software operational risk*.

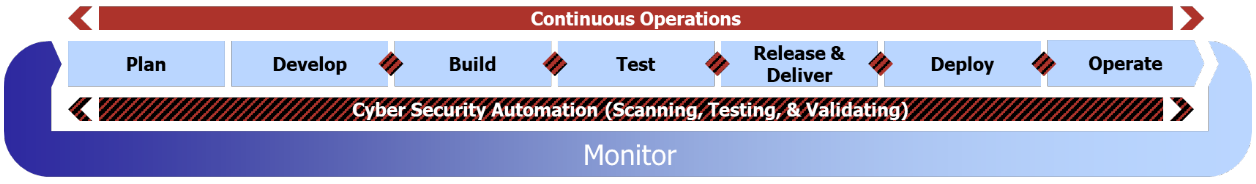


Figure 12 Continuous Operations Feedback Loop

Availability is often illustrated best by the concept of a Service Level Agreement (SLA). Today's modern applications are expected to be always available with near-zero downtime, measured in 9's. Software available 99.9% of the time can only be offline about 44 minutes per month; 99.99% availability drops to only 4 minutes per month, and so on.

This does not imply that the software is never updated. Modern microservice software architectures built on containers offer operational characteristics that enable high availability and performance scaling.

Performance of the software must respond to the normal ebb and flow of user demand. The most extreme example given is often in the retail world, where the demands during the last month of the year, e.g., the holiday season, create massive spikes in user demand as sales are announced. The instantaneous nature of these spikes requires heavy automation to ensure the software performance isn't degraded beyond usability limits.

Continuous Monitoring

> "Information security continuous monitoring (ISCM) is defined as maintaining ongoing awareness of information security, vulnerabilities, and threats to support organizational risk management decisions." – *Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations (NIST SP 800-137).*[12]

The final, all-inclusive phase and continuous feedback loop covering all phases of the DevSecOps lifecycle is Continuous Monitoring, depicted in *Figure 13*. Continuous monitoring recognizes the totality of the system must be monitored as a whole, not only as individual parts.

[12] National Institute of Standards and Technology, "Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations (SP 800-137)." Sep. 2011, [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-137/final

This approach ensures that teams are not forming inaccurate opinions about the software by only looking at a local minima or maxima. All aggregated metrics are monitored, from the flow of features from backlog into production, to the outputs of each of the control gates.

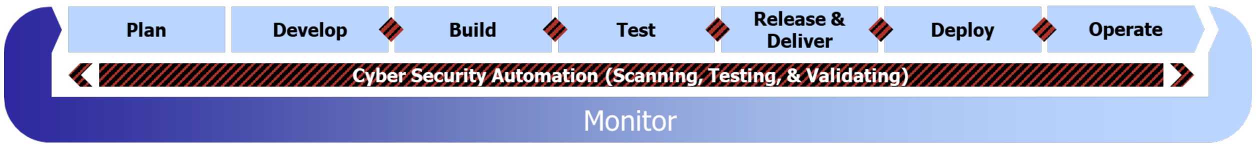


Figure 13 Continuous Monitoring Phase and Feedback Loop

Continuous monitoring constantly watches *all* system components, watches the performance and security of *all* supporting components, analyzes *all* system logging events, and considers all *external* threat conditions that may rapidly evolve. Continuous Monitoring provides insight into security control compliance, control effectiveness at mitigating a changing threat environment, and resulting analysis of the residual risk compared to the authorizing officials risk tolerance.

Specific, Measurable, Attainable, Relevant, and Time-Bound (SMART) performance metrics are also closely watched in this feedback loop. Performance metrics collected at *every* phase of the DevSecOps software lifecycle must be SMART. For example, measuring how long it takes to *type* a user story is specific, measurable, attainable and time-bound – *but is it relevant*? (The answer is *no*.) The section *Measuring Success with Metrics* later in this document explores a number of industry recognized SMART performance metrics that programs should adopt.

DevSecOps Platform

A DevSecOps Platform is defined as a multi-tenet environment that brings together a significant portion of a software supply chain, operating under cATO or a provisional ATO. The components of a DevSecOps platform can be instantiated in many ways, and each will include a mixture of options. Each reference design's unique platform configurations must be clearly defined across each of these three distinct layers: **Infrastructure, Platform/Software Factory, Applications**. These layers and their constituent components are depicted below in *Figure 14*.

The *Infrastructure Layer* supplies the hosting environment for the Platform/Software Factory layer, 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 Cloud Service Provider (CSP), but is not limited to a CSP.

At the boundary between the Infrastructure Layer and the Platform/Software Factory Layer is a **Reference Design Interconnect**. The purpose of the interconnect is to recognize that specific reference design manifestations may stipulate unique environmental requirements against the infrastructure layer. These unique requirements are often seen as either requiring proprietary IaaS tooling or specific architectural constructs to enhance the overall security of the next layer (Platform/Software Factory Layer).

The value proposition of each **Reference Design Interconnect** block depicted in *Figure 14* is found in how each reference design explicitly defines specific tooling and explicitly stipulates additional controls between the layers and the internal components within the layer. These interconnects are an acknowledgement of the need for *platform architectural designs* to support the primacy of security, stability, and quality. Each Reference Design must acknowledge and/or define each interconnect.

The *Platform/Software Factory Layer* includes the distinct development environments of the software factory, its CI/CD pipelines, a clearly implemented log aggregation and analysis strategy, and continuous monitoring operations. In between each of the architectural constructs within this layer is a Reference Design Interconnect. This layer 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 layer heavily rely upon the CI/CD pipelines, each equipped with a purpose-driven set of tools and process workflows. The environmental boundaries heavily automated, strict control gates control promotion of software artifacts from dev to test, and from test to integration. This layer also encompasses 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, and it must include log analysis and netflow analysis for event and incident detection.

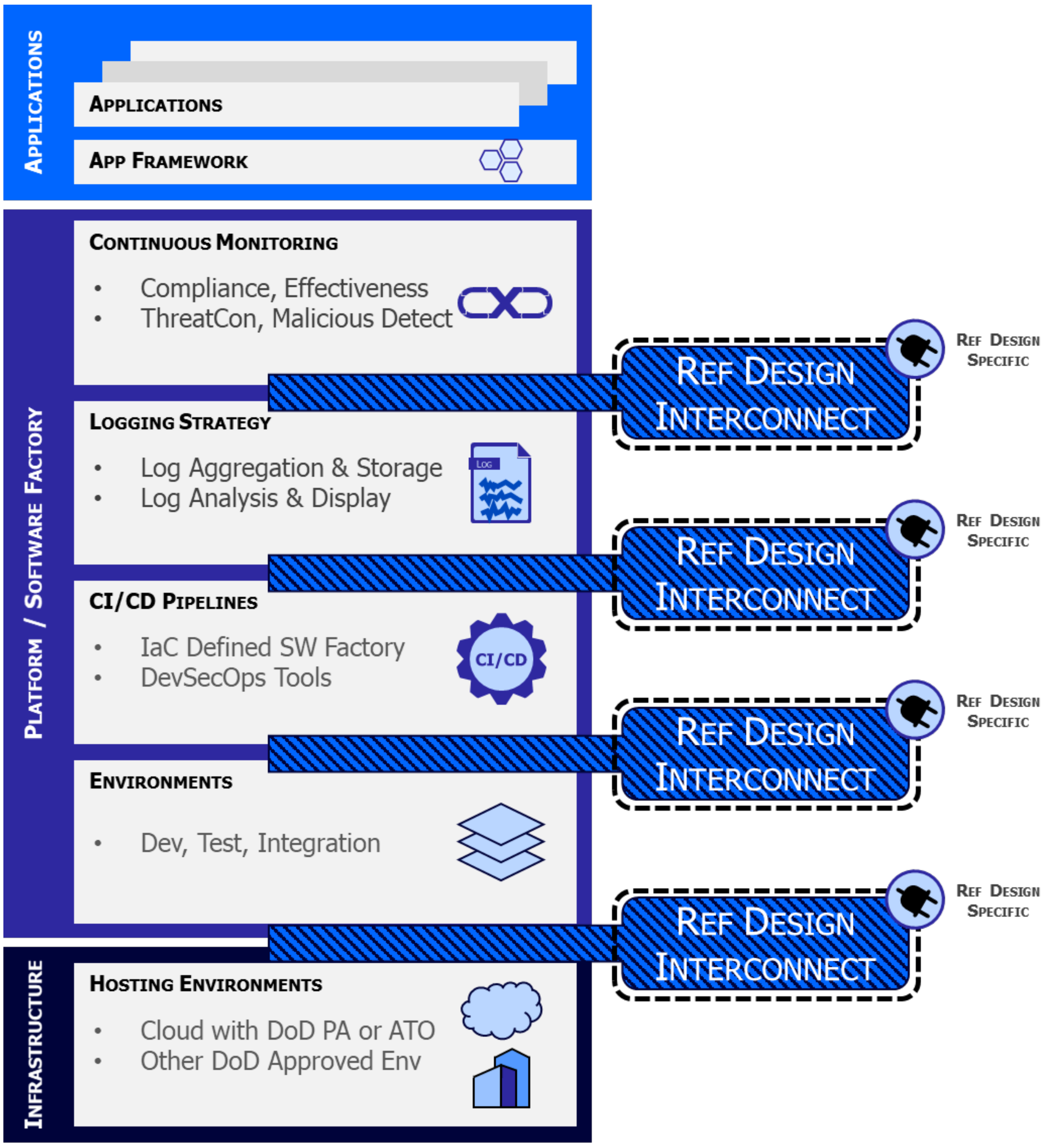


Figure 14 Notional DevSecOps Platform with Reference Design Interconnects Identified

The *Application Layer* includes 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.

DevSecOps Platform Conceptual Model

Each DevSecOps platform is composed of multiple software factories, multiple environments, multiple tools, and numerous cyber resiliency tools and techniques. The conceptual model in *Figure 15* visualizes the relationships between these and their expected cardinalities.

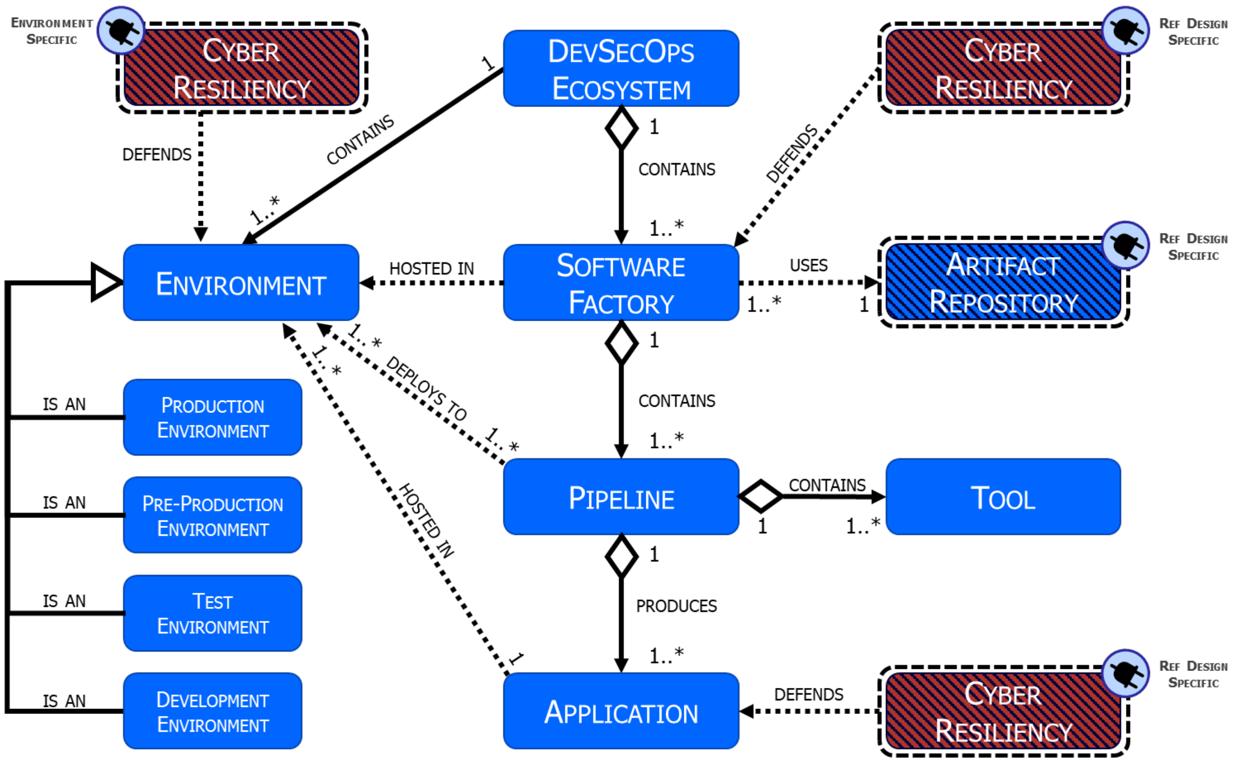


Figure 15 DevSecOps Conceptual Model with Cardinalities Defined

NOTE: When reading text along an arrow, follow the direction of the arrow. For example, a DevSecOps Ecosystem contains one or more software factories; each software factory contains one or more pipelines, etc.

Current and Envisioned DevSecOps Software Factory Reference Designs

As of this update, there remains only one approved DoD Enterprise DevSecOps Reference Design, built upon a CNCF Certified Kubernetes. A goal of this revision (v2) is to create the constructs for exploration and potential approval or provisional ATO of new types of reference designs, recognizing that industry continues to push DevSecOps culture and philosophies into new environments.

CNCF Kubernetes Architectures

The CNCF Certified Kubernetes Reference Design is predicated upon the use of a CNCF-compliant Kubernetes to orchestrate a collection of containers and its cybersecurity stack. The security stack provides Zero Trust continuous monitoring with behavior detection leveraging the reference design's mandated Sidecar Container Security Stack (SCSS). To review the complete reference design, please refer to the DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes.

NOTE: This is currently the most mature and presently the only approved DoD Enterprise DevSecOps Reference Design.

Low Code/No Code and RPA Architectures

The DevSecOps Strategy Guide explicitly defines the need to scale to any type of operational requirement needing software, to include business systems, attended and unattended bots, and beyond. There are rapid advancements taking place in low code/no code and robotic process automation (RPA) tooling and architectures. In this release (v2), there is neither an approved nor a provisionally authorized DevSecOps reference design for this type of environment. However, there has been a growing amount of discussion about how to leverage the cultural and philosophical ideals of DevSecOps in this area.

Serverless Architectures

Serverless architectures rely on fully managed and scalable hardware in a way that allows the software developer to emphasis business process over architecture. Serverless has rapidly matured, with commercial offerings across all major CSPs and open source libraries that plug-in to existing ecosystems like CNCF Certified Kubernetes stacks. In this release (v2), there is neither an approved nor a provisionally authorized **standalone** DevSecOps reference design for this type of environment.

Of note, Kubernetes supports Serverless workloads and Serverless architectures within the DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes.

CSP Managed Service Provider Architectures

The CSP ecosystem is also advancing rapidly, as CSPs look to differentiate themselves in the market by offering complete turn-key DevSecOps environments. The number of overall CSP managed services available grows each month, and many now offer full-featured IDEs through configuration management repository, build tools, artifact tagging and release management, and the continuous monitoring tooling spanning an entirely self-contained CI/CD pipeline.

The DoD Cloud IaC baselines are a service that leverage IaC automation to generate preconfigured, preauthorized, Platform as a Service (PaaS) focused environments. These baselines exist in the form of IaC templates that organizations can deploy to establish their own decentralized cloud platform. These baselines significantly reduce mission owner security responsibilities by leveraging inheritance from CSP PaaS managed services, where host and middleware security is the responsibility of the CSP, including hardening and patching (no secure technical implementation guide (STIG), no host-based security system (HBSS), and no assured compliance assessment solution (ACAS) required). Each baseline includes its associated inheritable controls in eMASS to expedite the Assessment and Authorization (A&A) process. Whenever possible, DoD Cloud IaC leverages managed security services offered by CSP over traditional data center tools for improved integration with cloud services. DoD Cloud IaC baselines can be built into DevSecOps pipelines to rapidly deploy the entire environment and mission applications. The department is presently working to determine what demand signals exist for a reference design for this style of architecture.

Lock-In: As noted in the *DevSecOps Strategy Guide*, the government must acknowledge a lock-in posture, recognizing vendor lock-in, *and* recognizing other types of lock-in like product, platform, and mental. Today's CSP offerings include many services, including CNCF Certified Kubernetes. Selection of a CSP managed service architecture creates lock-in, as does standardization on a CNCF Certified Kubernetes. What is important is to acknowledge and understand the lock-in posture of any given architecture.

Minimal DevSecOps Tools Map

The following material is a high-level summary of the *DevSecOps Tools and Activities Guidebook* included with this document set. The Guidebook provides a clean set of tables that completely captures both required and preferred tooling across each lifecycle phase's activities.

Each Reference Design is expected to augment the *DevSecOps Tools and Activities Guidebook*, adding in its environmentally specific required and preferred tooling. Reference Designs cannot remove a *required* tool or activity, only augment.

Architecture Agnostic Minimal Common Tooling

Every DevSecOps software factory 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 reference design. The following is a list of the common tools that are expected to be available within a DevSecOps software factory:

- **Team Collaboration System**: A popular example is Atlassian Confluence, a collaborative wiki that allows teams to share documentation in a way that is deeply integrated into the issue tracking system. The selected tool must support multi-tenancy.
- **Issue Tracking System:** A popular example is Atlassian Jira, a robust issue tracking system that is customizable to meet the needs of each specific team. The selected tool must support multi-tenancy.
- **Integrated Development Environment:** Many different IDEs exist, a popular example tuned for building and debugging modern cloud applications is Visual Studio Code.

- **Source Code Repository:** All DevSecOps software factories **must use a git based** repository. Popular git based repositories include GitLab and Atlassian Bitbucket.
- **Build Tool**: The build tool is an essential part of the CI/CD pipeline. A popular example of a build tools is Jenkins.
- **Artifact Repository:** The artifact repository must integrate into the build tool, providing access to both dependency artifacts required to complete the build, as well as storing artifacts that a result of the build.
- **Automated Test Development Tooling / Suite:** The specific features of the testing suite will be explicitly linked to the type of application be designed and built.
- **Test Coverage Reporting Tool:** Teams are encouraged to establish a high (but reasonable) threshold for code test coverage, and reporting of that coverage level is vital.
- **Static and Dynamic Application Security Test (SAST, and DAST, respectively) Tools:** Advanced tools can detect a range of concerns, spanning poorly implemented software algorithms to glaring (and not so glaring) security holes.
- **Log Strategy:** Log aggregation, analysis, and auditing tools must be incorporated into the operation of every software factory.
- **Continuous Monitoring:** InfoSec and operational monitoring of the entire software factory is mandatory, and the explicit tools and mechanisms to accomplish this must be clearly defined.
- **Alerting and Notification Strategy:** CI/CD pipelines must have access to an alerting and notification capability to proactively notify the team of any problems that need addressed. Alerts and notifications may arise from the issue tracking system, build tool, automated test tool(s), SAST/DAST, and monitoring tool(s).

Measuring Success with Performance Metrics

DevSecOps provides demonstrable quality and security improvements over the traditional software lifecycle. Proficiency can measured against both *tempo* and *stability* metrics, both of which are thoroughly defined through the DevOps Research and Assessment (DORA) Quick Check.[13]

As an introduction to DORA, the following key measurements are strong indicators of a team's proficiency at DevSecOps:

- Deployment Frequency – cycle Time, planning to production.
- Lead Time – the measurement between commit time to production deployment.
- Mean Time to Resolution (MTTR) – how long to get your code back up and running, if there is an incident.
- Change Failure Rate (CFR) – % changes going into production that require rework.

Performance metrics are important tools and measurement of metrics requires a considerable level of honesty in accepting where the team excels and where the team struggles. Remember that failure in agile is never a bad thing. In fact, it is encouraged to fail often and quickly to learn rapidly, and it is through those learning cycles that the team grows into a successful DevSecOps organization.

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.

DORA also publishes an annual *State of DevOps Report*. The DORA 2019 report identifies six indicators of growth across teams surveyed against the four metrics above:

1. *The industry continues to improve, particularly among the elite performers.*

   The proportion of our highest performers has almost tripled, now comprising 20% of all teams. This shows that excellence is possible—those that execute on key capabilities see the benefits.

2. *Delivering software quickly, reliably, and safely is at the heart of technology transformation and organizational performance.*

   We see continued evidence that software speed, stability, and availability contribute to organizational performance (including profitability, productivity, and customer satisfaction). Our highest performers are twice as likely to meet or exceed their organizational performance goals.

[13] DevOps Research and Assessment Quick Check, [Online]. Available: https://www.devops-research.com/quickcheck.html.

3. *The best strategies for scaling DevOps in organizations focus on structural solutions that build community.*

   High performers favor strategies that create community structures at both low and high levels in the organization, including Communities of Practice and supported Proofs of Concept, likely making them more sustainable and resilient to reorgs and product changes.

4. *Cloud continues to be a differentiator for elite performers and drives high performance.*

   The use of cloud—as defined by NIST Special Publication 800-145— is predictive of software delivery performance and availability. The highest performing teams were 24 times more likely than low performers to execute on all five capabilities of cloud computing.

5. *Productivity can drive improvements in work/life balance and reductions in burnout, and organizations can make smart investments to support it.*

   To support productivity, organizations can foster a culture of psychological safety and make smart investments in tooling, information search, and reducing technical debt through flexible, extensible, and viewable systems.

6. *There is a right way to handle the change approval process, and it leads to improvements in speed and stability and reductions in burnout.*

   Heavyweight change approval processes, such as change approval boards, negatively impact speed and stability. In contrast, having a clearly understood process for changes drives speed and stability, as well as reductions in burnout.

The findings from the DORA 2019 *State of DevOps Report* is insightful in understanding what the department can expect to gain from successful cultural adoption and technology maturity, as well as industry based insights for indicators on where teams may need to improve on adoption and utilization of cloud resources.

Another set of metrics that have value for measuring day to day operations are *Google's Four Golden Signals.*[14] These metrics have a higher value for platform operators than for developers or security team members, but are very useful for developers to measure application performance, and the same platform metrics aid 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).

These metrics are best measured by understanding the specifics of the platform and of the application performance; necessary monitoring tools will need to be employed to collect data.

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

DevSecOps Next Steps

This document has provided an introduction to Agile, DevSecOps, Software Supply Chains, Software Factories, and continuous feedback loops, as well as an overview of the minimum set of DevSecOps tools and activities, and current and aspirational DevSecOps Reference Designs. Adoption of DevSecOps and its related cultural, philosophical, 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 and learn 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:

- DevSecOps Tools and Activities Guidebook.
- DevSecOps Reference Design for CNCF Kubernetes.
- Learn more about Google's DORA and their DevSecOps research.

Appendix A Acronym Table

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

DDOS	Distributed Denial of Service
DevSecOps	Development, Security, and Operations
DISA	Defense Information Systems Agency
DNS	Domain Name Service
DoD	Department of Defense
DoDI	DoD Instruction
DORA	DevOps Research and Assessments
DTRA	Defense Threat Reduction Agency
EO	Executive Order
FaaS	Function as a Service
FMA	Forensic Media Analysis
FOSS	Free and Open-Source Software
GB	Gigabyte
GEP	Global Enterprise Partners
GOTS	Government Off The Shelf
HIPPA	Health Insurance Portability and Accountability Act
HTTP	Hypertext Transfer Protocol
IA	Information Assurance
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAST	Interactive Application Security Test
ICAM	Identity, Credential, and Access Management
IDE	Integrated Development Environment
IdP	Identity Provider
IDS	Intrusion Detection System
IE	Information Enterprise
IHR	Incident Handling Response
INFOCON	Information Operations Condition
IO	Input/Output
IPS	Intrusion Prevention System
IR	Incident Reporting
ISCM	Information Security Continuous Monitoring
IT	Information Technology
JVM	Java Virtual Machine
KPI	Key Performance Indicator
MB	Megabyte
MilDep	Military Department
MNP	Malware Notification Protection
mTLS	mutual Transport Layer Security authentication

MTTR	Mean Time to Resolution
MVP	Minimum Viable Product
NGG	Next Generation Governance
NIST	National Institute of Standards and Technology
NoSQL	Non SQL
NSM	Network Security Monitoring
OCI	Open Container Initiative
OS	Operating System
PA	Provisional Authorization
PaaS	Platform as a Service
PCI	Payment Card Industry
PEO	Program executive Officer
POA&M	Plan of Action and Milestones
QA	Quality Assurance
RBAC	Role-Based Access Control
RMF	Risk Management Framework
ROI	Return on Investment
SaaS	Software as a Service
SAF	Secretary of the Air Force
SAST	Static Application Security Test
SCAP	Security Content Automation Protocol
SCCA	Secure Cloud Computing Architecture
SCSS	Sidecar Container Security Stack
SLA	Service Level Agreement
SIEM	Security Information and Event Manager
SOAR	Security Orchestration Automation and Response
SRG	Security Requirements Guide
SQL	Structured Query Language
SSH	Secure Shell
STIG	Security Technical Implementation Guide
T&E	Testing and Evaluation
TRB	Technical Review Board
VM	Virtual Machine
ZTNA	Zero Trust Network Access

Appendix B Glossary of Key Terms

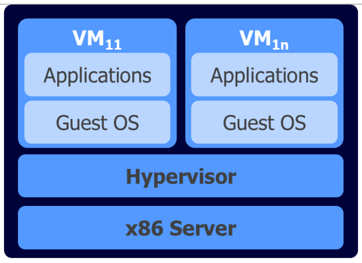
Following are the key terms used in describing the reference design in this document.

Term	Definition
Artifact **Software Artifact**	An artifact is a consumable piece of software produced during the software development process. Except for interpreted languages, the artifact is or contains compiled software. Important examples of artifacts include container images, virtual machine images, binary executables, jar files, test scripts, test results, security scan results, configuration scripts, Infrastructure as a Code, documentation, etc. Artifacts are usually accompanied by metadata, such as an id, version, name, license, dependencies, build date and time, etc. Note that items such as source code, test scripts, configuration scripts, build scripts, and Infrastructure as Code are checked into the source code repository, not the artifact repository, and are not considered artifacts.
Artifact Repository	An artifact repository is a system for storage, retrieval, and management of artifacts and their associated metadata. Note that programs may have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the content types.
Bare Metal **Bare Metal Server**	A bare metal or bare metal server refers to a traditional physical computer server that is dedicated to a single tenant and which does not run a hypervisor. This term is used to distinguish physical compute resources from modern forms of virtualization and cloud hosting.
Binary **Binary File**	Binary refers to a data file or computer executable file that is stored in binary format (as opposed to text), which is computer-readable, but not human-readable. Examples include images, audio/video files, exe files, and jar/war/ear files.
Build **Software Build**	The process of creating a set of executable code that is produced by compiling source code and linking binary code.
Build tools **Software Build Tools**	Used to retrieve software source code, build software, and generate artifacts.
CI/CD Orchestrator	CI/CD orchestrator is a tool that enables fully or semi-automated short duration software development cycles through integration of build, test, secure, store artifacts tools. CI/CD orchestrator is the central automation engine of the CI/CD pipeline
CI/CD Pipeline	CI/CD pipeline is the set of tools and the associated process workflows to achieve continuous integration and continuous delivery with build, test, security, and release delivery activities, which are steered by a CI/CD orchestrator and automated as much as practice allows.
CI/CD Pipeline Instance	CI/CD pipeline instance is a single process workflow and the tools to execute the workflow for a specific software language and application type for a software component. As much of the pipeline process is automated as is practicable.

Cloud Native Computing Foundation (CNCF)	CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable.
CNCF Certified Kubernetes	CNCF has created a Certified Kubernetes Conformance Program. Software conformance ensures that every vendor's version of Kubernetes supports the required APIs. Conformance guarantees interoperability between Kubernetes from different vendors. Most of the world's leading vendors and cloud computing providers have CNCF Certified Kubernetes offerings.
Cloud Native (Architecture)	"Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster." - https://www.cncf.io/
Code	Software instructions for a computer, written in a programming language. These instructions may be in the form of either human-readable source code, or machine code, which is source code that has been compiled into machine executable instructions.
Configuration Management	Capability to establish and maintain a specific configuration within operating systems and applications.
Container	A standard unit of software that packages up code and all its dependencies, down to, but not including the OS. It is a lightweight, standalone, executable package of software that includes everything needed to run an application except the OS: code, runtime, system tools, system libraries and settings.
Continuous Build	Continuous build is an automated process to compile and build software source code into artifacts. The common activities in the continuous build process include compiling code, running static code analysis such as code style checking, binary linking (in the case of languages such as C++), and executing unit tests. The outputs from continuous build process are build results, build reports (e.g., the unit test report, and a static code analysis report), and artifacts stored into Artifact Repository. The trigger to this process could be a developer code commit or a code merge of a branch into the main trunk.
Continuous Delivery	Continuous delivery is an extension of continuous integration to ensure that a team can release the software changes to production quickly and in a sustainable way. The additional activities involved in continuous integration include release control gate validation and storing the artifacts in the artifact repository, which may be different than the build artifact repository. The trigger to these additional activities is successful integration, which means all automation tests and security scans have been passed. The human input from the manual test and security activities should be included in the release control gate.

	The outputs of continuous delivery are a release go/no-go decision and released artifacts, if the decision is to release.
Continuous Deployment	Continuous deployment is an extension of continuous delivery. It is triggered by a successful delivery of released artifacts to the artifact repository. The additional activities for continuous deployment include, but are not limited to, deploying a new release to the production environment, running a smoke test to make sure essential functionality is working, and a security scan. The output of continuous deployment includes the deployment status. In the case of a successful deployment, it also provides a new software release running in production. On the other hand, a failed deployment causes a rollback to the previous release.
Continuous Integration	Continuous integration goes one step further than continuous build. It extends continuous build with more automated tests and security scans. Any test or security activities that require human intervention can be managed by separate process flows. The automated tests include, but are not limited to, integration tests, a system test, and regression tests. The security scans include, but are not limited to, dynamic code analysis, test coverage, dependency/BOM checking, and compliance checking. The outputs from continuous integration include the continuous build outputs, plus automation test results and security scan results. The trigger to the automated tests and security scan is a successful build.
Continuous monitoring	Continuous monitoring is an extension to continuous operation. It continuously monitors and inventories all system components, monitors the performance and security of all the components, and audits & logs the system events.
Continuous Operation	Continuous operation is an extension to continuous deployment. It is triggered by a successful deployment. The production environment operates continuously with the latest stable software release. The activities of continuous operation include, but are not limited to: system patching, compliance scanning, data backup, and resource optimization with load balancing and scaling (both horizontal and vertical).
CSP Managed Service	Proprietary tooling offered by a CSP
Cybersecurity, Software Cybersecurity	The preventative methods used to protect software from threats, weaknesses and vulnerabilities.
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).
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 software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of software development: plan, develop, build, test, release, deliver, deploy, operate, and monitor.
DevSecOps Ecosystem	A collection of tools and process workflows created and executed on the tools to support all the activities throughout the full DevSecOps lifecycle. The process workflows may be fully automated, semi-automated, or manual.
DevSecOps Pipeline	DevSecOps pipeline is a collection of DevSecOps tools, upon which the DevSecOps process workflows can be created and executed.
DevSecOps phase	The software development, security, and operation activities in the software lifecycle are divided into phases. Each phase completes a part of related activities using tools; DevSecOps defines eight distinct phases.
Environment	Sets a runtime boundary for the software component to be deployed and executed. Typical environments include development, test, integration, pre-production, and production.
Factory, Software Factory	A software assembly plant that contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. The software factory supports multi-tenancy.
Software Factory Artifact Repository	Stores artifacts pulled from DCAR as well as locally developed artifacts to be used in DevSecOps processes. The artifacts include, but are not limited to, VM images, container images, binary executables, archives, and documentation. It supports multi-tenancy. Note that program could have separate artifact repositories to store local artifacts and released artifacts. It is also possible to have a single artifact repository and use tags to distinguish the contents.
Hypervisor	A hypervisor is a kind of low-level software that creates and runs virtual machines (VMs). Each VM has its own Operating System (OS). Several VMs can run on one physical machine, depicted in *Figure 16*.

	 VM$_{11}$ Applications Guest OS VM$_{1n}$ Applications Guest OS Hypervisor x86 Server *Figure 16 Notional Hypervisor Construct*
Image Management, Software Image Management, Binary Image Management, Container Image Management, VM Image Management	The process of centralizing, organizing, distributing, and tracking of software artifacts.
Immutable infrastructure	An infrastructure paradigm in which servers are never modified after they're deployed. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they're validated, they're put into use and the old ones are decommissioned. The benefits of an immutable infrastructure include more consistency and reliability in your infrastructure and a simpler, more predictable deployment process. (from: https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure)
Infrastructure as Code	The management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning that the DevSecOps team uses for source code. Infrastructure as Code evolved to solve the problem of environment drift in the release pipeline.
Kubernetes	An open-source system for automating deployment, scaling, and management of containerized applications. It was originally designed by Google and is now maintained by the CNCF. Many vendors also provide their own branded Kubernetes. It works with a range of container runtimes. Many cloud services offer a Kubernetes-based platform as a service.
Lockdown	The closing or removal of weaknesses and vulnerabilities from software.
Microservices	Microservices are both an architecture and an approach to software development in which a monolith application is broken down into a suite of loosely coupled independent services that can be altered, updated, or taken down without affecting the rest of the application.
Mission Application Platform	The mission application platform is the underlying hosting environment resources and capabilities, plus any mission program enhanced capabilities that form the base upon which the mission software application operates.

Monitoring **Security Monitoring**	The regular observation, recording, and presentation of activities.
Node **Cluster node**	A node is a worker machine in CNCF Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components, including the node controller.
OCI	“An open governance structure for the express purpose of creating open industry standards around container formats and runtime” - https://www.opencontainers.org/
OCI Compliant Container	Container image conforms with the OCI Image Specification.
OCI Compliant Container Runtime	A container runtime is software that executes containers and manages container images on a node. OCI compliant container runtime must conform with the OCI Runtime Specification.
Orchestration	Automated configuration, coordination, and management.
Platform	A platform is a group of resources and capabilities that form a base upon which other capabilities or services are built and operated.
Pod	A group of containers that run on the same CNCF Kubernetes worker node and share libraries and OS resources.
Provisioning	Instantiation, configuration, and management of software or the environments that host or contain software.
Reporting	An account or statement describing an event.
Repository	A central place in which data is aggregated and maintained in an organized way.
Resource	CPU, Memory, Disk, Networking
Scanning, **Security Scanning**	The evaluation of software for cybersecurity weaknesses and vulnerabilities.
Sidecar Container Security Stack	A sidecar container security stack is a sidecar container(s) that enhances the security capabilities of the main containers in the same Pod.
Sidecar	A sidecar is a container used to extend or enhance the functionality of an application container without strong coupling between two. When using CNCF Kubernetes, a pod is composed of one or more containers. A sidecar is a utility container in the pod and its purpose is to support the main application container or containers inside the same pod. For more information, see Section **Error! Reference source not found.** and Kubernetes documentation.
Telemetry	Capability to take measurements and collect and distribute the data.
Test Coverage, **Code Coverage**	Test coverage is a measure used to describe what percentage of application code is exercised when a test suite runs. A higher percentage indicates more source code executed during testing, which suggests a lower chance of containing undetected bugs.
Virtual Machine (VM)	Emulates a physical computer in software. Several VMs can run on the same physical device.
Virtual Network	Networks constructed of software-defined devices.
Virtual Storage	Storage constructed of software-defined devices.