UNCLASSIFIED

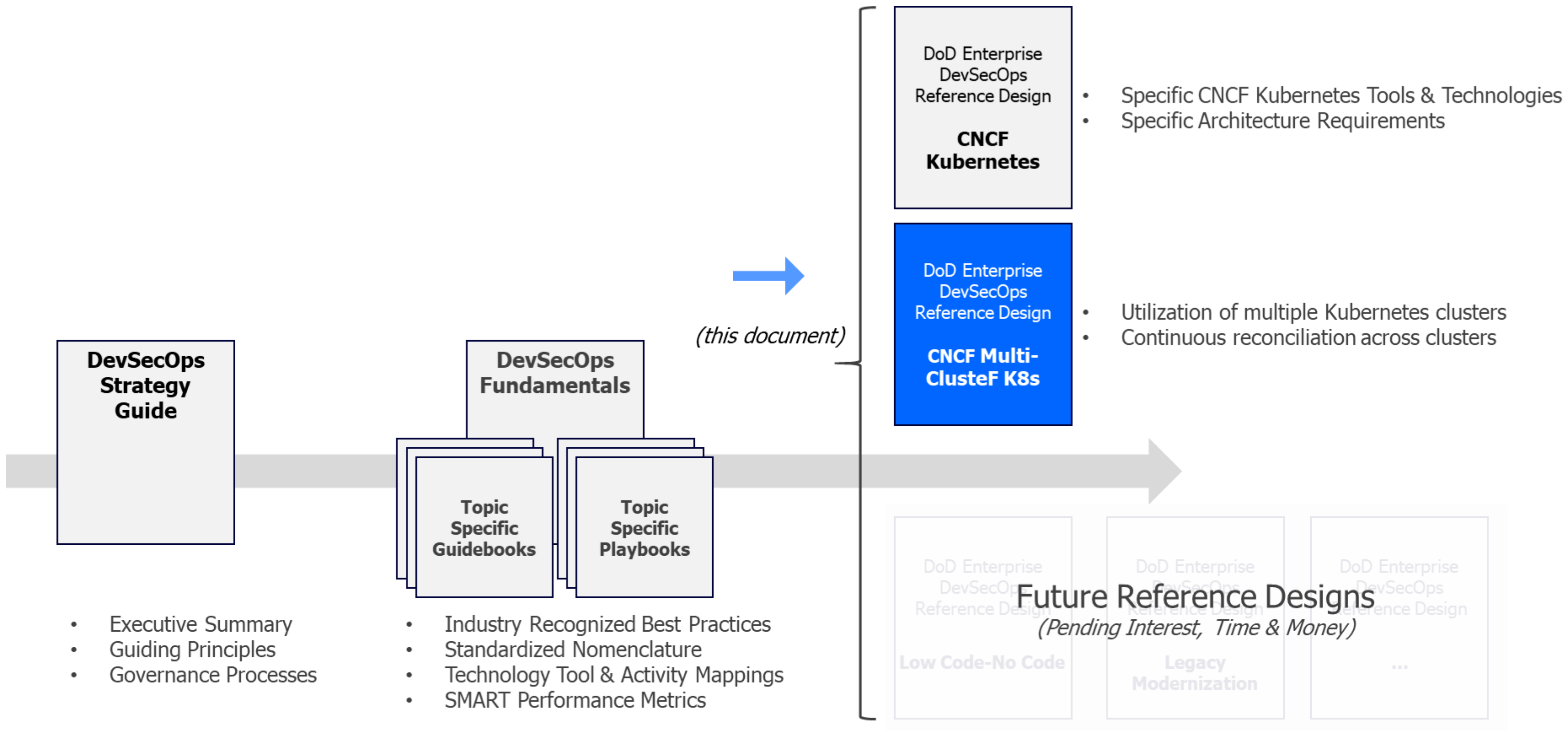
# DoD Enterprise DevSecOps Reference Design:

CNCF Multi-Cluster Kubernetes

July 2022

Version 1.0

## Document Set Reference


DoD Enterprise DevSecOps Reference Design
CNCF Kubernetes
Specific CNCF Kubernetes Tools & Technologies
Specific Architecture Requirements
(this document)
DoD Enterprise DevSecOps Reference Design
CNCF Multi-ClusteF K8s
Utilization of multiple Kubernetes clusters
Continuous reconciliation across clusters
DevSecOps Strategy Guide
DevSecOps Fundamentals
Topic Specific Guidebooks
Topic Specific Playbooks
Executive Summary
Guiding Principles
Governance Processes
Industry Recognized Best Practices
Standardized Nomenclature
Technology Tool & Activity Mappings
SMART Performance Metrics
DoD Enterprise DevSecOps Reference Design
Low Code-No Code
Legacy Modernization
...
Future Reference Designs
(Pending Interest, Time & Money)

## Document Approvals

Approved by:

PUCKETT.PAUL.B.1277536486 Digitally signed by PUCKETT.PAUL.B.1277536486 Date: 2022.08.15 11:52:50 -04'00'

Paul B. Puckett III
Director, Enterprise Cloud Management Agency
HQDA CIO

LAMB.GEORGE.WILLIE.IV.1506724321 Digitally signed by LAMB.GEORGE.WILLIE.IV.1506724321 Date: 2022.08.08 12:02:32 -04'00'

George Lamb, DoD CIO - IE/CSM

for Robert Vietmeyer
DoD CIO – Information Enterprise Directorate

## Trademark Information

## Contents

## Figures

## Tables

# 1 Introduction

## 1.1 Background

The Department of Defense (DoD) is at a defining moment as the services pivot to deliver cloud-based services, adopt cloud-native software development practices, and DevSecOps methodologies[1] that serve to modernize the DoD in preparation for the digital battlefield and competition with near-peer threats. The CNCF Multi-Cluster Kubernetes Reference Design provides the DoD with a model and guidance on building and deploying cloud-native software on Kubernetes to deliver measurable software-based outcomes to the mission with resilience and at the speed of relevance.

## 1.2 Purpose

The CNCF Multi-Cluster Kubernetes Reference Design is part of the DoD Enterprise DevSecOps Reference Design family and adheres to the structure of other DoD Reference Design documents. This reference design is intended to serve as architectural and design guidance to DoD organizations that intend to build and deploy cloud-native software on Kubernetes. This reference design describes the verifiable attributes and lists the preferred and required set of tools and activities across all phases of the DevSecOps lifecycle.

## 1.3 Scope

The CNCF Multi-Cluster Kubernetes Reference Design is a blue-print for designing a secure application platform for DoD organizations. This platform will allow these organizations to interconnect and share infrastructure services while maintaining the independence and flexibility to enact on their unique missions. While this design provides options to centralize management services, such management plane points can be forward deployed based on the needs of the truly unique and organic composition of DoD networks, which includes disconnected, hard to access, and highly secured networks. By allowing flexibility for Authorizing Officials (AO) and mission owners to own, share, or borrow components of this design, the design naturally enables progressive adoption for workloads that might be in a state of migration, both in their hosting locations and technical architectures.

All components of this design can be elastically provisioned and dynamically scaled in their installation while also taking advantage of building upon existing installations, either in service capabilities or workload hosting. The design accommodates for workload mobility, cloud migration, on-premises migration, and continued portability across various hosting locations (with special consideration including edge and tactical networks).

[1] https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsFundamentals.pdf

Each recommendation in this design relies heavily on the concept of cloud native computing, which "empowers organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds."[2] The platform utilizes containers, a software packaging abstraction that focuses on application components and their dependencies. To run distributed applications at scale using containers, Kubernetes is used as a foundational substrate across the environments. In addition to providing a consistent abstraction for computing resource allocation, Kubernetes has become the industry standard for orchestration and operation of modern distributed applications. Alternative form factors to containers can also be hosted if required.

## 1.4 DevSecOps Compatibility

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

- DoD Enterprise DevSecOps Strategy Guide, Version 2.1[3]
- DevSecOps Tools and Activities Guidebook, Version 2.1[4]
- Army's DevSecOps Playbook 1.1[5]

The CNCF Multi-Cluster Kubernetes Reference Design further supports the Army's DevSecOps Playbook 1.1, which codifies the continuous Risk Management Framework (cRMF) model in the Army. cRMF supports application Mission Owners in rapid delivery of Authority to Operate (ATO)-ed software applications. By utilizing the ECMA's accredited CReATE, Mission Owners can receive a full ATO by adhering to a standard set of security guardrails and maximum use of controls inheritance outlined in the CReATE's Continuous Integration / Continuous Delivery (CI/CD) pipeline and Platform determined by the CReATE Authorizing Official, and codified in the DevSecOps Playbook.

# 2 Assumptions

- Network boundaries are within an existing accreditation boundary with a private IP space.
- Communication through network boundaries, especially on DoD networks, might require additional cybersecurity, coordination and configuration.
- Existing accreditation boundaries should have an accredited Virtual Data Center Managed Services (VDMS), this is also referred to as delegated authentication and this VDMS providing industry standard authentication is assumed. Alternative technologies that leverage industry standard authentication can also be used if they meet Defense

[2] https://github.com/cncf/toc/blob/main/DEFINITION.md
[3] https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsStrategyGuide.pdf
[4] https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsTools-ActivitiesGuidebook.pdf
[5] https://www.milsuite.mil/book/docs/DOC-1050076 (CAC required)

Information Systems Agency's (DISA) Secure Cloud Computing Architecture (SCCA)[6] requirements.

- The selected Kubernetes implementation must pass the Kubernetes conformance test suite and support lifecycle management through the Kubernetes SIG Cluster Lifecycle Cluster API project in order to fully benefit from certain control mappings.
- Adoption of the ecosystem design can in some instances be achieved through different alternative choices of software tools. The reference design makes an effort to recommend the most mature and suitable tools for the core design principles as of the time of this writing, along with tradeoffs and decision making criteria when evaluating a particular technology selection over the other.
- This reference design will be updated as tools and design principles evolve in industry.
- The Software Factory Interconnects and Detailed Descriptions of Core Principles are not fully prescriptive about tool selection and are presented as the means to adopt the model. The later section on Creating an Ecosystem will illustrate how the Army set about building out an implementation that is compliant with this reference design, and offer the reader recommendations of some open source tooling that is not required for adoption of the design.
- Common shared services within an existing network boundary can be leveraged to reduce the overall burden of accreditation and maintenance. This is encouraged for Foundational Services and Enterprise Services (see the Army Cloud Plan[7], Appendix A for examples of some of these services).

## 3 Software Factory Interconnects

The DoD Enterprise DevSecOps Fundamentals[8] describes a DevSecOps platform as: "*a multi-tenant environment consisting of three distinct layers: Infrastructure, Platform/Software Factory, and Application(s). Each reference design is expected to identify its unique set of tools and activities that exist at the boundaries between the discrete layers, known as Reference Design Interconnects. Well-defined interconnects in a reference design enable tailoring of the software factory design, while ensuring that core capabilities of the software factory remain intact*."

The core principles of the platform for the CNCF Multi-Cluster Kubernetes Reference Design are designed to be further enhanced with future reference designs for higher levels of abstraction on top of Kubernetes, and therefore this design focuses on the foundational layers rather than those closer to the application level.

---

[6] https://storefront.disa.mil/kinetic/disa/service-catalog#/forms/secure-cloud-computing-architecture
[7] https://www.army.mil/standto/archive/2020/10/09/
[8] https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsFundamentals.pdf

A DevSecOps Ecosystem on Kubernetes must demonstrate that the following verifiable attributes have been programmatically implemented:

- **Immutable Infrastructure:** Create infrastructure that is scaled up, destroyed, and recreated rather than persisted.
- **Continuous Reconciliation:** Microservices and software components are installed to enable rapid patching, prevent configuration drift, and alignment with approved baselines. The configuration is policy-driven to enable guardrails and self-service layering.
- **Multi-Cluster Kubernetes:** The principle of multiple relatively small clusters compared to a single super cluster further reduces blast radiuses, allows better isolation, and enables better distribution of resources. Distributed resources allow better collaboration across organizations and disparate environments. Multiple clusters thrive on composable toolchains; when setup and configuration of the environment is done declaratively through intent driven APIs, additional management burden is not created. Clusters effectively communicate workloads running on top to coordinate more complex deployment requirements.
    - **Smaller production locations and minimalist edge environments:** The ability to distribute workloads means production environments can be disconnected, in a different location from development, and still have high confidence on consistent environments.
    - **Secure inter-service communications:** Communications between and within clusters are authenticated and encrypted.
- **Provenance in Secure Supply Chain:** Third party software is verified and documented; containers and software originating from upstream locations meet DoD requirements that include vendor-support where applicable, continually maintained from upstream, updated repeatedly, and strive to have binding agreements that software is maintained. Stewardship of this supply chain is assured to meet DoD ongoing maintenance needs.
    - **Scanned and verified images:** No in-place patching. All artifacts to be deployed with special consideration to container images are to be modularly built with layers that can be individually updated, scanned, and verified.
- **Production Ready Kubernetes, Even in Development:** Complete operating environments: clusters that possess the necessary additional components deployed in a thoughtful way to achieve compliance needs, even when deployed into development or lab environments, to create a consistent environment with the necessary oversight for authorizing officials and security teams. Minimize operational overheads and complexity while going above and beyond.

Figure 1: Multi-Cluster Kubernetes Reference Design Interconnects visually depicts these specific interconnects.

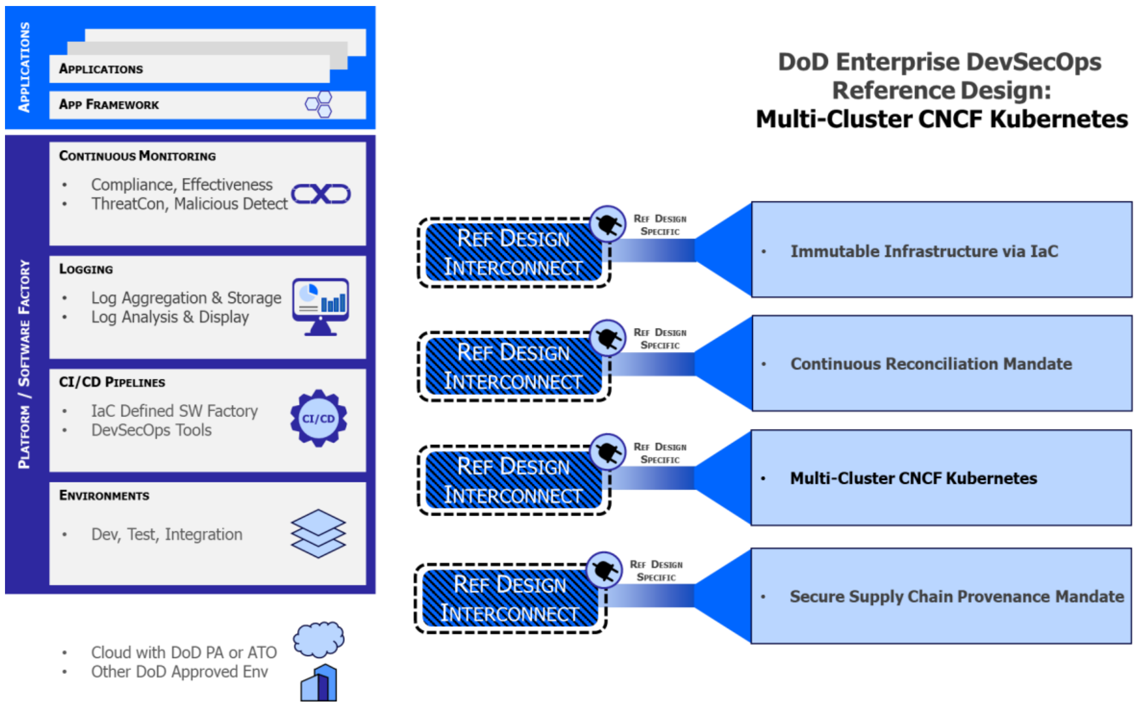

Applications
Applications
App Framework
Platform / Software Factory
Continuous Monitoring
Compliance, Effectiveness
ThreatCon, Malicious Detect
Logging
Log Aggregation & Storage
Log Analysis & Display
CI/CD Pipelines
IaC Defined SW Factory
DevSecOps Tools
CI/CD
Environments
Dev, Test, Integration
Cloud with DoD PA or ATO
Other DoD Approved Env
DoD Enterprise DevSecOps Reference Design: Multi-Cluster CNCF Kubernetes
Ref Design Interconnect
Ref Design Specific
Immutable Infrastructure via IaC
Continuous Reconciliation Mandate
Multi-Cluster CNCF Kubernetes
Secure Supply Chain Provenance Mandate

Figure 1: Multi-Cluster Kubernetes Reference Design Interconnects

# 4 Detailed Descriptions of Core Principles

## 4.1 Immutable Infrastructure

In a modern platform there are numerous layers of infrastructure and abstraction that teams can operate at - from all the way down to the bare metal to virtual constructs like Kubernetes services. Shrinking this sphere of concern is of utmost importance for operations teams to be able to effectively maintain and harden their environments. Immutable infrastructure creates the basis on which consistent underlying environments can be provisioned for the platform to operate on and deliver mission value.

Kubernetes runs on top of worker machines, called nodes, that run traditional operating systems to provide the underlying capabilities containerized applications need. Since containers are not operating systems, an environment still has to take care of proper hardening, compliance, and maintenance on these underlying nodes. Cluster API is a Kubernetes sub-project focused on providing declarative APIs and tooling to simplify provisioning, upgrading, and operating multiple Kubernetes clusters. Cluster API facilitates provisioning the underlying nodes in a declarative, immutable manner that greatly facilitates immutable approaches to infrastructure management.

Mutating operating systems in place creates risk that changes will impact running workloads, allow configuration drift, or permit advanced persistent threats. Leveraging Kubernetes ability to cordon and remove underlying nodes, along with proper accountability for disruption budgets across tenants, enables delegating of upgrading underlying systems more readily to the environment’s control. As a result, operating systems are more easily patched while providing necessary assurances that any drift in configuration is periodically reconciled.

New Virtual Machines are instantiated from a known good state template that has been cleaned, including the latest updated packages, General Purpose Operating System Security Requirements Guide (SRG)[9], and any relevant Security Technical Implementation Guides (STIG) baselines, and brought in to replace the old virtual machines. This operation occurs prior to removing the old virtual machines, creating a scale up operation prior to scaling down, so running workloads are given the necessary time to schedule over to a new machine. By managing this from a central source, even when multiple Kubernetes clusters are deployed, every Virtual Machine across the entire network can be properly updated on a prearranged scheduled basis to meet the requirements of a given Authorizing Official’s posture.

Kubernetes running on top of these nodes follows a similar pattern with the higher level abstractions, like Deployments, StatefulSets, Jobs, and DaemonSets. These constructs leverage immutable container images instantiated with similar security controls applied to prevent mutability of running workloads. By mounting configuration into these ephemeral environments as read-only there is significantly better control over what is visible from the requester and the resources are given access to the information on a per-session basis. Nodes that do not require access to secrets or other environment variables relevant to the workloads the node is running are not able to access that information.

As virtual machines are rotated in this fashion, the underlying authentication credentials that allow nodes to communicate with the Kubernetes control plane are similarly refreshed. Newly scheduled pods on these nodes authenticate through these refreshed credentials. The node's credentials provide access to the underlying storage for the configuration necessary for all workloads.

This process also greatly enhances the security posture of Configuration Management (CM) and Contingency Planning (CP) in the NIST SP 800-53[10] family. Combining this capability with continuous reconciliation of components gives the organization fully patched systems on regular intervals. A lot of the concepts mentioned in this section are also requirements for proper implementations of Zero-Trust Architectures based on NIST SP 800-207[11].

## 4.2 Continuous Reconciliation of Components

Modern distributed systems, especially those leveraging microservices, typically have a larger deployment topology than a single binary. For example, a Source Code Repository may include databases, caching mechanisms, proxy servers, etc. Installation and configuration of these

[9] https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_General_Purpose_Operating_System_V2R1_SRG.zip
[10] https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
[11] https://csrc.nist.gov/publications/detail/sp/800-207/final

components is a complicated and error prone process. A successful deployment (and maintenance) process needs to address the following concerns:

- There must be an obvious, single, trusted process to run the deployment/updates, complete with appropriate access controls. This can be called the Control Plane.
- There must be a means to validate the processes and artifacts used in the deployment.
- There must be a method to allow granular configuration on where the components are deployed.
- There must be a method to allow per-deployment customization of the installed software beyond vendor-provided templates.

Recent trends in industry guide organizations towards concepts such as Infrastructure as Code (IaC) or Configuration as Code. These trends, while useful, fail to consider the end user. For example, a playbook, written in some automation framework, that stands up a given environment is of little use to an end user who doesn’t know where or how to run that playbook It is also difficult to ascertain the level, and duration of, permissions required to run the playbook.

A better approach involves continuously enforcing the state of the system, a process known as Continuous Reconciliation. Figure 2 visualizes cluster creation and enforcement using continuous reconciliation. By placing the Kubernetes component configurations in a central repository (considered as a “source of truth”), each cluster can be configured to refer to this repository. Each cluster, independently, will work to ensure that its own configuration is the same as that defined in the central repository. When clusters are created, the standard baseline deployment and configuration are pulled from the central repository. After creation, the same central repository is continuously referenced and the central configurations are enforced on the cluster. While still maintaining compliance with a central set of instructions, this arrangement means troubleshooting and customization are easier to achieve. Such an approach also provides a more reliable mechanism to facilitate upgrades and ownership of those components' health while preventing unauthorized modification.

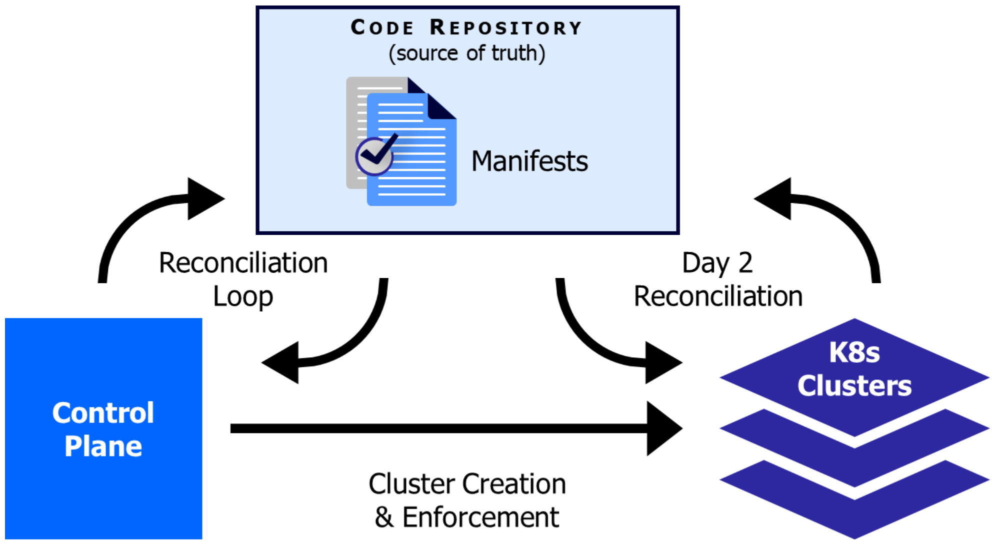


Figure 2: Cluster Creation and Enforcement via Continuous Reconciliation

Continuous Reconciliation provides better accountability for the deployment and maintenance steps than a repository that simply holds configuration (i.e., a standalone playbook). It forces an organization to centralize and automate the process for running their configuration on the environment. By automating this process, it can be repeated and leveraged to prevent configuration drift.

Configuration of components can be bundled with the Kubernetes manifests necessary to run those configurations, regardless of whether they are custom built, vendor provided, or open source. Additionally, these bundles can be validated, signed, and stored in a repository. The repository can be relocated to different environments while maintaining the signature for validation in a decentralized manner.

Finally, while vendor-provided hooks exist for customization, the author of a given Kubernetes component cannot predict the needs of every end user. There will be cases where customization is needed but the author-supplied manifest does not allow for this. In order to address this concern, the same workflow mentioned above can inject, or overlay, additional configurations on top of the manifests. This per-site customization strikes the balance of honoring central management while providing an agile method for responding to unknown circumstances in the field.

By design, Continuous Reconciliation of Components is decentralized in nature. Therefore, it can occur from multiple focal points in an environment. For individual teams or areas of responsibility within this environment, the centralizing of these components naturally becomes a regional control plane, or a team centered control plane.  If a single team has responsibility for the majority of the components in a given environment (for small footprint, or small teams) then that centralized control plane would form a regional control plane.

## 4.3 Multi-Cluster Kubernetes

DoD environments are numerous and complicated; some enclaves have limited or zero connectivity to egress out to the public internet. These restrictions are accentuated for enclaves on the Department of Defense Information Network (DODIN). Due to the sensitive nature of the data that resides inside of these environments, there is higher regulation for any requests to egress or ingress into the network, further complicated by the need to respect Ports, Protocols, and Services Management (PPSM) process[12]. Additional regulations create friction and barriers to entry for new organizations trying to stand up development environments, operational environments, and create additional hurdles for timely patching and maintenance on environments.

Multi-Cluster Kubernetes is a design paradigm to provide reduced friction on bringing the correct tools to the correct environments. Creating a global view across all enclaves allows instantiation of workloads and developer tooling into the areas that make the most sense in which to establish peering and connectivity around the complexity of DoD environments. When possible, leveraging industry standard caching and information transfer for artifacts and fully secured connections allows workloads to follow a path to production that might result in the ability for workloads to run in disconnected or limited connectivity environments. Developers are able to develop code, visualize testing, and push workloads through numerous services that might be hosted in cloud environments, data centers, or tactical networks. Using a common substrate across these environments enables elasticity and better efficiency while simultaneously enabling Authorizing Officials (AOs) to maintain full control over their enclaves.

Kubernetes is a distributed system, meaning that elasticity and on-demand capabilities are possible in a localized fashion (in a single cloud environment or a single datacenter). Kubernetes provides an Application Programming Interfaces (API) that enables consumption of these on-demand capabilities. In addition, the declarative and intent driven nature of the Kubernetes API means that work can be described at rest more efficiently, so that when a cluster becomes available it can be equipped with the information needed to perform a task. More on this is described in the previous section on Continuous Reconciliation of Components. While environments that have nonstandard forms of communication cannot be continuously reconciled against, a fallback pattern is to relocate this information effectively (see next section on Provenance in Secure Supply Chain) so that the environments can still stand up and participate in a larger ecosystem of capabilities.

[12] https://public.cyber.mil/connect/ppsm/

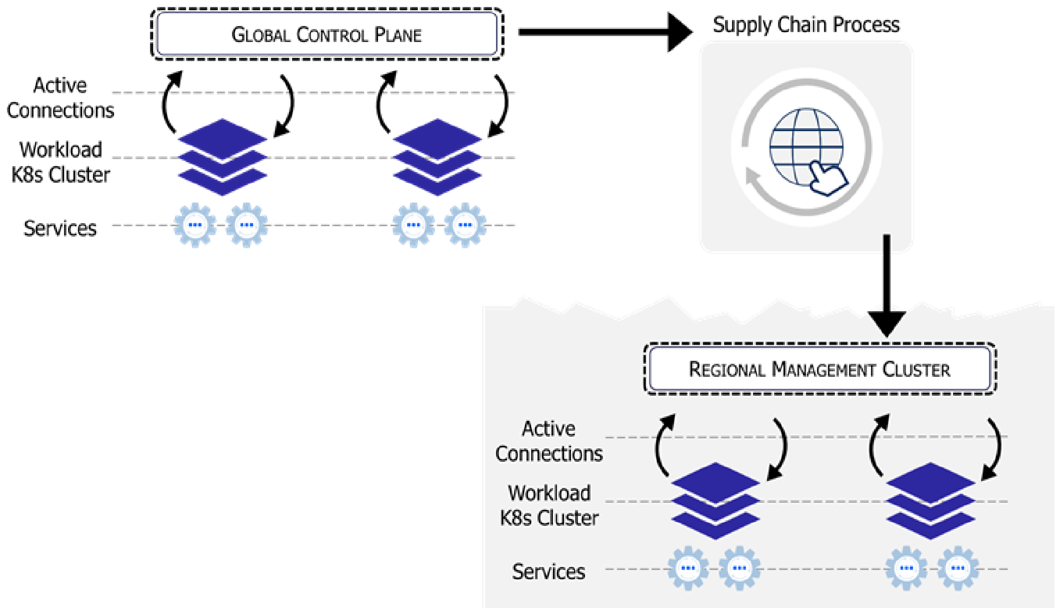


Figure 3. Multi-Cluster Kubernetes relies on a global control plane

Kubernetes APIs are exposed to different users based on Role-Based Access Control (RBAC) through the delegated authentication, enabling a self-service and API driven ecosystem. This enables a larger classification of users, rather than just privileged and non-privileged. For example, the Army DevSecOps Playbook asserts enough controls and guardrails can be implemented to enable non-privileged developers appropriate access to their source code and the path to production to reach all security and compliance requirements. Engineers are able to push new changes to subsequent environments because there is enough visibility, testing, and predictability for security and authorizing officials to understand the risk posture each push to the next environment would create.

This design structure posits that the individual secure components of a software factory can be scaled and instantiated independently if those services are able to maintain secure connectivity. For example, containers require a hosting cluster for various stages in a software development cycle, but the creation of those containers could occur in a central location. This would allow certain mission owners that operate in a Denied, Disrupted, Intermittent, and Limited (DDIL) environments for production to build and develop their containers in a cloud environment while deploying production workloads onto a cluster hosted in that DDIL environment. As a direct result, mission owners reduce duplication of software factory development areas and gain the benefit of flexibility. This also contributes to a much smaller footprint for production environments, which is sometimes crucial when that environment is DDIL.

<u>Secure inter-service communication</u>:

When sending messages between distributed software systems, it is common to rely on the network to determine the identity of the system that sent a message, to ensure that messages can only be received by systems that are authorized to view them, and to ensure that the message has not been seen or modified in transit by a malicious party. However, for complex distributed applications - which may span multiple networks that share many services deployed by different teams - it may be undesirable to rely on the network to protect communication. Mutual Transport Layer Security (mTLS) is a well-established technology that relies on asymmetric cryptography to prove the identity of the sender and recipient of a message, as well as assert that the message has not been viewed or modified in transit by a third party. It is thus useful for establishing communication in potentially untrusted networks.

## 4.4 Provenance in Secure Supply Chain

One of the critical aspects of modern application security is understanding all of the components that come together to create a distributed application. Recent security breaches have been caused by everything from misconfigured data service interfaces to maliciously altered code injected into an application build. DoD applications are especially sensitive to security vulnerabilities, and require stringent guardrails against risk that do not stifle developer productivity and innovation. NIST has a large effort on informing guidance on the entirety of supply chain security[13] and this section is meant as supplemental guidance in the frame of DoD DevSecOps methodologies. Provenance in Secure Supply Chain is meant to address specifically the origins of components when taken in an immutable environment and making sure that what is being written earlier in the process is preserved, scanned, and verified through to the later steps in the software development process.

Most existing software environments rely on a combination of developer discipline and “review by committee” to create a safe supply chain. Unfortunately, even with the best of intentions, this approach runs counter to the needs of agile software development approaches. In many large organizations, security acceptance reviews can be—by far—the biggest constraint in the software release process.

The CNCF Multi-Cluster Kubernetes Reference Design provides additional secure supply chain guidelines beyond existing guidance:

- **Source curation:** Components such as application source code, libraries, and shared services, must be vetted *before* the component is integrated into the application. Curated repositories of software build components and approved services (such as database or messaging services) can be combined with application code and image scanning to help automatically shift key security activity to the beginning of the application development process. Furthermore, an advanced software supply chain can make sure security patches are applied quickly and automatically across a large and complex application portfolio.

- **Artifact protection:** Elements moving through the supply chain must be immutable and verifiable, through techniques such as source code and container image signing. The supply chain system must protect against malicious parties modifying code or images,

[13] https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity

without the application team's knowledge. The supply chain should also protect against alteration of software process attestations (see next bullet), configuration files, and anything else that might make the application more vulnerable to a breach or attack.

- **Attestation:** One of the really powerful things a modern secure software supply chain can do is provide verifiable documentation of application composition, process steps taken to build and deploy the application, and test and vulnerability scanning output. These artifacts are then always instantly available for audit and review at any point during or after the application lifecycle. In fact, some aspects of auditing and verification can be automated using supply chain attestation artifacts.

To address these needs, additional guidelines to the software supply chain will consist of:

- Curated inputs from commercial vendors, open source communities, and even the DoD itself (and its agency partners).
- Process automation for both development and operations processes, including building, testing, scanning, and deploying software components.
- Observability and verification tools (including security tools such as the aforementioned vulnerability scanners, governance automation, etc.)

Curated inputs can be handled a number of ways, but the selective use of cloud native buildpacks, service brokers, and private source code management repositories (such as git) enable the DoD to mitigate risk without having to manage every single input individually. When managing these inputs consideration should be taken for compliance needs like NIST SP 800-53 and NIST SP 800-190[14]. There are other DISA references that apply NIST 800-53 controls to containers, such as section 2 and 3 in the DevSecOps Enterprise Container Image Creation and Deployment Guide, but this reference design does not expect adherence to the other examples in that guide and supports following any process that aligns with the intended outputs of OCI containers[15]. Similar guidance can be found in the FedRAMP Vulnerability Scanning Requirements for Containers[16].

Process automation is typically applied to the source code build-test-deploy cycle, but the larger supply chain can include everything from buildpack management to vulnerability patching in production. Tools that orchestrate steps can be combined with a good supply chain platform to compose an optimal flow for any given application.

Finally, verifying that the various artifacts of the software lifecycle are as they should be requires collecting data on both the software supply chain tools and the application itself. Analysis and

[14] https://csrc.nist.gov/publications/detail/sp/800-190/final

[15] https://dl.dod.cyber.mil/wp-content/uploads/devsecops/pdf/DevSecOps_Enterprise_Container_Image_Creation_and_Deployment_Guide_2.6-Public-Release.pdf

[16] https://www.fedramp.gov/assets/resources/documents/Vulnerability_Scanning_Requirements_for_Containers.pdf

visualization of this data should be as close to real-time as possible, enabling developers and operators alike to respond to anomalies quickly. This allows faster recovery times for issues discovered as early in the software lifecycle as possible.

These tools also allow the software development process to remain largely self-service, which is critical to developer productivity and experimentation. The DoD strives to minimize developer toil while also removing as much risk as possible from the software development lifecycle.

## 4.5 Production Ready Kubernetes, Even in Development

Kubernetes offers a primitive for building platforms within an organization. Most organizations within the DoD prefer to operate at a higher turn-key solution level and need compliance, security, and stability more readily available to growing organizations. By injecting additional capabilities into each Kubernetes cluster as it is created, and continually reconciling those capabilities to prevent configuration drift, clusters can be spun up on-demand to meet the needs of a growing ecosystem. Applying these capabilities into Kubernetes in thoughtful places, like as additional workloads (such as Daemon Sets), Service API extensions, or Admission Controllers, Kubernetes can meet all compliance needs from a security technical implementation guide (STIG) or security requirements guide (SRG) need.

These additional capabilities consist of:

- **Storage Integrations**, making sure requisitioned storage is properly encrypted for data at rest for both platform components, the underlying VMs, and critical supporting databases (like etcd). This is also expandable to secrets at rest being properly encrypted and in a readily rotatable position.
- **Container Networking**, making sure the container overlay networking complies with necessary encryption at transit requirements and conforms to the underlying network topology. This networking also integrates with policies at the cluster level to isolate workloads and prevent unnecessary routing internally (allow listing, internal cluster traffic, and egress).
- **Service Routing**, used for exposing workloads securely for ingress while making sure proper certificates and TLS are employed.
- **Observability**, visibility into logging and metrics of running workloads, but also forwarding necessary audit logging to the enclave's audit and logging solution.
- **Identity, Credential, and Access Management,** providing secured, credentialed access to users, services, applications, and interaction with the underlying infrastructure. Limiting this access to the least privileged and making sure sessions are validated.
- **Admission and Validation,** scheduling new workloads should be validated against best practices, and mutating those workloads when possible to meet those requirements. This provides additional gates prior to entry into a cluster, some non-exhaustive examples: enforcing only signed images, validating images against vulnerabilities, applying DoD issued certificate authorities, or inserting proxy settings. These processes,

implemented as controllers running on the clusters, can provide additional security, governance, and configuration management.

Implementing these capabilities makes adhering to the compliance guidance laid out in the Kubernetes STIG[17] distinctly more manageable. Additionally, these capabilities have a high overlap with the guidance in the Kubernetes Hardening Guidance[18] provided by the National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA), which all result in a more secure, authorizable, production ready Kubernetes platform.

## 5 Additional Tools and Activities

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

[17] https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_Kubernetes_V1R2_STIG.zip
[18] https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR_KUBERNETESHARDENINGGUIDANCE.PDF

Table 1: Tools Spanning all Phases

| Tool | Features | Benefits | Inputs | Outputs | Baseline |
|---|---|---|---|---|---|
| CI/CD Orchestrator | Creation and execution of automated tasks, jobs, and workflows. Does not have to be a single orchestrator but can be.<br><br>Preferred to be managed by the Global Control Plane, but not a hard requirement. | The underpinning to the CNCF Multi Cluster Kubernetes phase operations. | A set of processes, workflows, and/or stages that are run based on some trigger - a code push, timeline, etc. | A visible set of out outputs to include, but not limited to:<br>- Status<br>- Resulting outcome<br>- Logs | REQUIRED |
| Identity, Credential, and Access Management | Provide RBAC and other identity/authorization services to the cluster and its operations | Allows for segmentation and scoping of capabilities. | A notion of roles and/or permissions. Requests from users. | Decisions on allowing or permitting an action | REQUIRED |
| Common service/bundle repository | The concept here is that there is a shared interface for organizations to request common services and/or “bundles” - examples can be applications, network connections, databases, etc. | Creates mechanisms for the various clusters and organizations to interface on common abstractions, minimizing the workload of any given team and reducing redundant work. | A request for a running service/connection/appliance. | Access to a provisioned, hardened, authorized instance of the requested entity. | PREFERRED |

Table 2: Security Activities Summary and Cross-Reference

| Activities | Phase | Activities Table Reference | Tool Dependency | Tool Table Reference |
|---|---|---|---|---|
| Enforce node immutability | Build | Table 4 | Node mutation prevention tool | Table 3 |
| Artifact signing/validation | Build | Table 4 | Release packaging tool | Table 3 |
| Container image security scan | Test | Table 6 | Test tool suite | Table 9 (DevSecOps Tools and Activities Guidebook) |
| Bundle compliance scan | Test | Table 6 | Test tool suite | Table 9 (DevSecOps Tools and Activities Guidebook) |
| Admission control | Test | Table 6 | Admission controller | Table 5 |
| Node rotation | Deploy | Table 10 | Rotation automation tools | Table 9 |
| Cluster network management | Deploy | Table 10 | Networking service | Table 9 |

Table 3: Build Phase Tools

| Tools | Features | Benefits | Inputs | Outputs | Baseline |
|---|---|---|---|---|---|
| Container builder | Build a container image based on minimal and hardened base images and appropriate language runtimes. | Container image build automation | Source code, language runtimes, minimal base images | OCI compliant container image | REQUIRED |
| Bundler | Build a relevant deployment bundle for Kubernetes as a container is not a sufficient mechanism to deploy standalone and most deployments require more than a single container to run. | Pairs specific runtime requirements in Kubernetes alongside the associated containers - also provides a mechanism to indicate how containers should be run. | Kubernetes configuration and dependent OCI container image | OCI compliant container artifact: a bundle. | REQUIRED |
| Node mutation prevention tool | Packages the underlying operating systems Kubernetes runs on as Read Only. | Prevents mutations to the underlying operating systems reducing escalations of privilege | Underlying operating system for the deployed nodes. | Read Only operating system/runtime on the deployed nodes. | PREFERRED |
| Release packaging tool | Extends the packaging tool referenced in Table 11 of the DevSecOps Tools and Activities guidebook to strongly encourage signing other deployed artifacts beyond VM's. | Signing of Kubernetes deployable artifacts. | OCI compliant bundle of deployable Kubernetes objects. | Signed bundle. | PREFERRED |

Table 4: Build Phase Activities

| Activities | Description | Inputs | Outputs | Tool Dependencies |
|---|---|---|---|---|
| Containerize | Packages all required OS components, developed code, runtime libraries, etc. into a hardened container | Source code, language runtimes, minimal base images | OCI compliant container image | Container builder |
| Bundle the deployment | Build the deployable artifact for Kubernetes. | Kubernetes configuration and dependent OCI container image | OCI compliant container artifact in a bundle. | Bundler |
| Enforce node immutability | Harden the base nodes so as to prevent any inadvertent or malicious mutations to the shared resource. | Underlying operating system for the deployed node | Read Only operating system/runtime on the deployed node | Node mutation prevention tool |
| Artifact signing/validation | Sign built release bundles for non-repudiation and attestation. | Raw deployable Kubernetes bundle | Signed deployable Kubernetes bundle | Release packaging tool |

Table 5: Test Phase Tools

| Tools | Features | Benefits | Inputs | Outputs | Baseline |
|---|---|---|---|---|---|
| Admission controller | Automatically facilitates policy checking for Kuberenetes bundles. | Prevents unsuitable, improperly configured, or outright malicious bundles/components from entering the Kubernetes environment. | Policies and rules. Bundles to be deployed | Decision on whether the bundle is permissible to be deployed | REQUIRED |

Table 6: Test Phase Activities

| Activities | Description | Inputs | Outputs | Tool Dependencies |
|---|---|---|---|---|
| Container image security scan | Scans all layers of container images down to the OS for known misconfigurations and CVEs | Container image | Security issues for remediation | Test tool suite |
| Bundle compliance scan | Often a custom built or configured set of tests that validate for adherence to organizationally defined controls and known STIGs/SRGs. | Bundle and/or it's components | Compliance issues for remediation | Test tool suite |
| Admission control | Checks against policies and rules to allow or deny bundle deployment. | Policies and rules. Bundles to be deployed. | Decision on whether the bundle is deployable. | Admission controller |

Table 7: Release and Deliver Phase Tools

| Tools | Features | Benefits | Inputs | Outputs | Baseline |
|---|---|---|---|---|---|
| Kubernetes bundle reconciliation tool | Enables the automated flow of bundle/component deployment and updating. An example implementation can be a GitOps flow. | Connects the source of truth of declarative Kubernetes bundles/components automatically to the various running environments | Raw Kubernetes bundles (be they deployed via helm, yaml, etc.) | Deployed bundles on running Kubernetes clusters. | REQUIRED |

Table 8: Release and Deliver Phase Activities

| Activities | Description | Inputs | Outputs | Tool Dependencies |
|---|---|---|---|---|
| Kubernetes bundle reconciliation | An automated loop for reconciling Kubernetes declarative sources of truth. | Stored declarative Kubernetes bundles | Running bundles on Kubernetes clusters in the appropriate environment | Kubernetes bundle reconciliation tool |

Table 9: Deploy Phase Tools

| Tools | Features | Benefits | Inputs | Outputs | Baseline |
|---|---|---|---|---|---|
| CNCF Certified Kubernetes | Features described in the “Container Deployment” model of the DevSecOps Tools and Activities Guide | Benefits described in the “Container Deployment” model of the DevSecOps Tools and Activities Guide | Kubernetes deployment configuration | Running cluster(s) with conformant application deployments. | REQUIRED |
| Global control plane | Ability to communicate with child deployed environments and provided shared resources like services and images | Enables a form of centralized management and planning while encouraging decentralized execution - all from a similar base substrate. | Commands or requests from child environments | Provisioned hardened resources and services for consumption by child environments and applications/systems. | REQUIRED |
| Rotation automation tools | Automatically cycles the underlying nodes beneath Kubernetes | Allows for continuous reconciliation of Node state, while reducing the risks of potential advanced persistent threats. | Node baseline configuration | Baselined and patched underlying nodes | REQUIRED |
| Service routing tool | Ability to administer mutual TLS (mTLS) and TLS across environments | Creates a more secure means for systems, services, and applications to communicate across various network environments. | Requests for certificates, routing policy, and authentication. | Pathways for services to communicate using the provided mTLS/TLS pathway. | REQUIRED |
| Networking service | Facilitate communication of Kubernetes pods with the internal cluster and outside world. Ideally this is Kubernetes Container Networking Interface (CNI) compliant[19]. | Make a secure and configurable mechanism to control network communications of pods. | Requests for allow listing or new connection creation | Configuration of the connection and determination on permissibility of the new connection request | REQUIRED |

[19] https://github.com/containernetworking/cni

Table 10: Deploy Phase Activities

| Activities | Description | Inputs | Outputs | Tool Dependencies |
|---|---|---|---|---|
| Automated service and resource requests | Requesting Kubernetes components, reconciliation, and/or usable services from a centralized location | Request to global control plane for some form of resource | The requested resource to be acted upon | Global control plane |
| Node rotation | Repave the underlying nodes in Kubernetes on a regular interval to reprovision all relevant secrets, perform updates, and reconcile drift. | Time trigger (bi-weekly at a minimum) | Baselined and patched underlying nodes | Rotation automation tool |
| Service to Service communication | Varying network services communicating securely with one another. | Requests for mTLS communication. | Successful communication across diverse network boundaries. | Service routing tool |
| Cluster network management | Ensure that pod and cluster communication pathways are secured, known, and controlled. | Creation of new bundles/components and/or requests for new interconnections within or outside the cluster. | A decision on the permissibility of the new connection request. | Networking service |