# Containerization Readiness Guide

May 12, 2021

Office of Information Integrity and Access

General Services Administration
Office of Government-wide Policy

# Table of Contents

# Purpose

As the Federal Government continues to adopt increasingly cutting-edge technologies and embrace large changes to existing Information Technology (IT) infrastructure, containers have become a growing topic of discussion throughout agencies. Some agencies already have budding containerization practices, other agencies are in the process of preparing for and building container capabilities and skills, and still others are early in the maturation process.

This guide provides a basic overview of container technologies to educate agencies that have limited to no containerization maturity. It will help agencies make informed and intelligent decisions on adopting container technologies. Intended as a "Container Readiness Guide," the reader will first find an overview of container technology and references to external resources for additional learning to set a baseline of terms, definitions, and types of technologies.

The guide also contains resources, such as a decision tree, to:

- Assist agencies in determining if they are ready for containers;
- Determine if containers are a true solution to existing challenges; and
- Evaluate if containers are a feasible and cost-effective solution.

Interwoven throughout are case studies, taken from other federal agencies that have already adopted container technologies. The case studies also provide best practices and lessons learned for agencies considering the adoption of container technologies.

Any references to vendors throughout the guide do not constitute an endorsement of their services. While the guide includes descriptions of some services to help inform your agency's container journey, the container landscape is vast, quickly evolving, and includes many services not listed here. Your agency is encouraged to consider all available options. **This document is in no way intended to guide any procurement decisions for your agency, and is intended to be strictly informational.**

# Overview of Container Technology

Containers are packages of software services that exist separately and independently from an existing host infrastructure. Containers consist of all the services required to operate on the Runtime Environment (RTE). Container environments house the application, all required dependencies, software libraries, and configuration files. Because container images hold everything needed for an application, developers do not need to code applications for new environments and deployment is greatly streamlined. Generally, applications have multiple containers functioning like isolated, secure building blocks for the application's software (**Figure 1**).

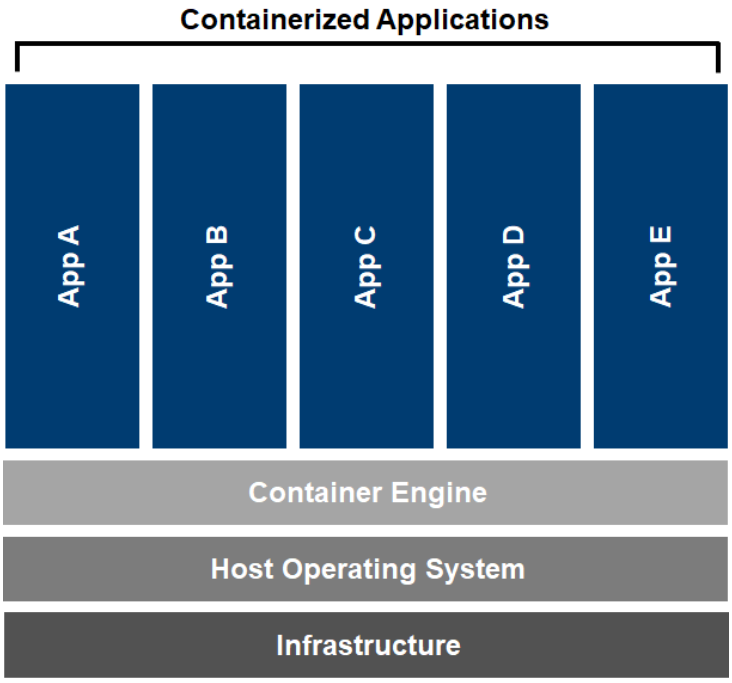


**Figure 1.** *This schematic illustrates a typical container environment, consisting of containerized applications, the container engine, host operating system, and the underlying infrastructure.*

Ultimately, containers are appealing to developers because of the technology rapidity, uniformity, and security that they provide in creating and deploying applications. By packaging up software code into a container, developers can run the code consistently across different infrastructures and environments.

## Container Images

A container image is the initial building block that contains files needed to run an application. Images are read-only files that are a "snapshot" of a single point in time. Containers *require* an image to exist. Since containers carry their own runtime and dependencies with them, container images can launch at any instance.

The Open Container Initiative (OCI) was formed to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. Image Format Specification defines an OCI Image as consisting of a manifest, an image index (optional), a set of filesystem layers, and a configuration.

## Types of Container Technology

- **Mirantis Kubernetes System -** Enterprise-ready container platform for building, configuring, and distributing Docker containers.

- **Kubernetes (k8s) -** Open-source container orchestration system. The system consists of a group of one or more containers that share storage and a local network, known as Pods.
- **Podman -** Open-source container engine. Performs the same role as Docker Engine. Docker and Podman CLI commands follow the same pattern.
- **Amazon Web Service (AWS) Elastic Container Service (ECS) -** Proprietary, fully-managed container orchestration service used to run and scale containerized applications.[1] Supports Docker containers.
- **Microsoft Azure Kubernetes Service (AKS) -** Open-source, fully managed container orchestration service based on Kubernetes. Supports Docker containers.

# Container Management

Container management refers to the complex system and environment that enables organizations to efficiently and effectively run and operate containers within their existing IT environment. Containers can run without the implementation of container management services; however, lack of these services will severely limit scalability. Different solutions may fit different business cases and needs. The primary elements of container management include container orchestration, monitoring, security, and governance.[2]

Use the CNCF Cloud Native Interactive Landscape to explore available industry solutions and tools for various container needs. Many of the emerging technologies surrounding container technologies, such as artificial intelligence/machine learning (AI/ML), Hyperledger, edge, and Internet of Things (IoT), are built based on open-source Linux Foundation frameworks and tools.

## Container Registries

A container registry is a repository or a collection of repositories that store container images. These images are usually different versions of an application or a service. We recommend agencies use container registries as they progress in their adoption journey.

Container registries may be public or private, though public registries may not meet the security requirements for your agency. Docker Hub is a public container repository with many publicly available and pre-configured container images that can quickly run applications in Docker containers. Private and hosted registries provide additional security controls, such as the ability to use multiple authentication methods, vulnerability scanning, role-based access, and traceability through audit logs. These controls offer your agency a level of assurance when storing and using containers. Iron Bank is an example of a private registry run by the USAF that provides the DoD with secure container images at an enterprise level.

[1] Any references to vendors do not constitute an endorsement of their services.
[2] *Market Guide for Container Management*, Gartner

There are several managed container registries available to federal agencies through commercial providers.

## Container Orchestration

Individually, containers are easy to deploy and maintain. However, the management burden rises as operations scale and more containers, services, and applications are added. The need to automate the deployment, networking, and availability of containers becomes critical at scale.[3] Container orchestration is a critical component of overall container management. In addition to orchestration, a successful container management system also contains load balancing, networking, schedulers, monitoring, and testing.

Operators are software extensions that help build and manage applications in Kubernetes. They can automate the packaging, deployment, and management of containerized applications. By automating an application's lifecycle, they at once reduce the management burden, and make processes scalable and standardized. For a comprehensive list of Operators, visit OperatorHub.io.

When multiple Operators are used, as is commonly the case, an Operator Lifecycle Manager (OLM) can manage these Operator clusters. An OLM is also known as an "Operator of Operators." Appendix 5 illustrates how an Operator of Operators, the Open Data Hub Operator, may be used to install other Operators.

### Business Case for Container Orchestration Solutions

Containers are implemented as their own service. As these business functions grow, so does the number of containerized services. Each service should have its own integration and deployment pipeline. Containerized services are subject to frequent deployments. Consequently, a coordination layer is needed to track which hosts run on each container. This layer should track the state of containers and the resources available in a cluster.

### Container Orchestration Tools

Container orchestration tools help organizations combine individual tools to create a comprehensive solution. For example, Docker and Kubernetes can work together to create and manage a modern cloud architecture. Orchestration tools help manage the systems, such as a specific number of repositories, containers, ports, and data volumes.

Docker is the tool that allows organizations to create containers and run services for applications in the Docker environment. Kubernetes is the tool that allows organizations to manage and scale these efforts. While this guide focuses on Docker because it is most commonly used, Kubernetes supports several container runtimes in addition to Docker,

[3] *How Docker and Kubernetes Work Together*, Pavan Belagatti
https://containerjournal.com/topics/container-ecosystems/how-docker-and-kubernetes-work-together/

including: containerd, CRI-O, and any implementation of the Kubernetes Container Runtime Interface (CRI).

Agencies need to build a total solution when preparing to operate with containers at scale. Industry options for container orchestration tools include but are not limited to:

- **Kubernetes (k8s) -** Open-source container orchestration system. The system consists of a group of one or more containers that share storage and a local network, known as Pods.
- **Docker Swarm -** Open-source container orchestration platform created by Docker that manages Docker hosts as a single, virtual host.

### Container Monitoring

Containers and containerized applications provide great assistance to users and developers until issues arise. Agencies need to ensure containerized environments are continuously monitored to:

- Maintain optimal or close to peak performance
- Proactively identify issues
- Perform maintenance and changes safely
- Lower risk

Multiple aspects of containers can be monitored. How much monitoring depends on multiple variables such as budget, goals, and acceptable risk. Agencies should consider each of these variables before exploring the numerous monitoring tools and vendors. The relationship between distribution and tool-side vendors facilitates a cost-effective and high-performance solution. See the list of security and compliance tools.

## Container Business Value

Containers offer federal agencies a unique opportunity to modernize their current legacy applications and develop new applications to take advantage of cloud services. They allow agencies to develop applications quickly, scale rapidly, and efficiently use their valuable resources. The key benefits of containers are as follows:

**Immutable Infrastructure -** Immutable infrastructure refers to software or systems that are never modified and remain in the same state. Containers are a good example of immutable infrastructure. A container image contains the code to run an application and provides a “static” element for IT operations teams to work with. The immutable aspect of the container provides a higher level of confidence for both testing and production.

**Quick Application Deployment -** Using containers frees developers from the tedious task of managing multiple configuration environments, supporting libraries, and configurations from testing to production environments. Containers can be created once and used

multiple times without additional effort. Through containers, developers can focus on application deployments rather than maintaining supporting configurations. The template-oriented nature of containers allows expeditious deployments with reliability and consistency across multiple environments. Rather than designing a new architecture for each new container, developers often reuse existing, reliable Kubernetes patterns to create the intended container architecture.

**Standardization -** A container unit has everything the software needs to run (e.g., libraries, system tools, code, and runtime) and allow for service modularity.

**Agility to Scale to Demand -** Federal agencies can use a container management system to cluster multiple containers together, schedule and automate deployments, and manage containers to meet mission needs and priorities. For example, by using multiple Pods to run multiple instances of an application, your agency can scale an application horizontally to support increased demand and reduce the incoming (ingress) and/or outgoing (egress) traffic from a single, overloaded Pod.

**Optimized Compute Resources -** Unlike virtual machines (VMs), multiple containers can run on a single operating system (OS) due to their lightweight nature, and their ability to quickly execute and maintain a consistent runtime model.[4] Given these benefits, the right tooling to help manage container optimization and efficiency can translate into significant resource optimization to organizations. Please refer to the CNCF Cloud Native Interactive Landscape for different types of optimization tools.

**Improved Security -** Containers can provide improved security through the use of additional tools and services. Potential benefits of the use of container security tools and services include:[5]

- **Transparency -** Containers are typically easier to inspect allowing a greater understanding of its continents compared to traditional VMs.
- **Security Isolation -** Deconstruction of applications into microservices allows identification and resolution of vulnerabilities without affecting the entire application.
- **Reduced Attack Surfaces -** A focus on securing the application running inside the container reduces the attack surface. Alternatively, on a virtual service, agencies must secure the host server, the virtual server, and application itself.
- **Updates -** Update containers in a centralized repository and easily deploy into a production environment. Quickly disseminate patches from a central repository.

[4] https://www.cloudops.com/blog/docker-and-kubernetes-what-is-the-value-of-containerization/
[5] https://containerjournal.com/features/security-benefits-docker-containers/

- **Consistent Environment -** Containers' ability to provide a consistent environment helps ensure that the containerized application remains secure because the environment variables are kept uniform prior to and during production. This uniformity, called environment parity, is a notable value of Docker.

Agencies should still follow good security practices and hygiene, and adhere to federal security guidelines and requirements, as outlined in the Security section. They should not expect improved security from adopting containers alone.

### Containerization Limitations

Containerization strategy pays off for modern application systems and middle tier infrastructure services that are loosely coupled and where "build once and deploy many" is being employed to combat configuration drift. However, in the case of A) legacy infrastructure where complex middleware systems with tightly coupled interdependent sub systems and B) monolithic applications, the cost outweighs the benefits. In such cases, it is recommended to optimize infrastructure and adopt application architecture that lends itself to reap containerization benefits. Another consideration is the complexity of orchestration and change management containerization required for both the team's workflow and tooling to implement it successfully in an enterprise.

## Measuring Container Value

As with all IT efforts, successful value measurement requires equal balance between benefit and cost measurement.

**Key benefits** to measure container value can include:

- Developer productivity
- Agile Continuous Integration/Continuous Development (CI/CD) environment
- Infrastructure gain
- Reduced operational overhead

**Key costs** to measure container return on investment (ROI) can include:

- CaaS/PaaS subscription/licenses
- Infrastructure acquisition and upgrades
- Staff training and hiring
- Rollout/implementation costs

Creating a clear cut and complete value proposition for executives or project managers typically requires a creative approach from IT teams. An example is provided below to get started:

> *One custom methodology to assess value for containers could assess how much start-up time, overhead, and coding time it would take to move an application to a new VM*

*environment and compare it to how long and how much time the team would need to move a container to a new environment. The time saved would be a proxy measurement for value, as saved employee times means employees can be working on other, higher-value projects.*

# Complementary Technologies

Several types of complementary technologies can work together to improve containers' functionality. These technologies can improve optimization, management, deployment, or scaling of containers. View these technologies as key add-ons for containers.

## Service Mesh

A service mesh is a layer of infrastructure responsible for communication between multiple services and the context of a container architecture. It manages data-sharing and communication between various microservices residing in multiple containers, and allows for east-west traffic enablement (i.e., data flow between devices within a given data center).

In a more complex container environment, whether due to increasing numbers of microservices or unpredictable traffic patterns, interactions between the containers can be challenging. Here, the service mesh plays a critical role. Identifying performance bottlenecks, faults, and traffic flow would be challenging without a system that interconnects the architecture. In addition to connecting containers, agencies can monitor and secure container clusters. Add-on open-source solutions, such as Prometheus, allow for transparency and metrics. Native service mesh solutions can help secure traffic through policy restrictions.

Several open-source service mesh products are available on the market for Kubernetes, such as Istio and Linkerd, along with other commercially available products.

## Serverless Computing

Serverless computing is an application development approach that employs computing power on-demand/as needed. Serverless computing allows development teams to focus on writing code functions to deploy in the serverless computing environment. Instead of a VM, the computing power of serverless comes into existence as needed and then disappears immediately after use. In other words, the server will only run as computing power is triggered. This reduces costs by running servers for a much shorter time (notably in comparison to containers, which require a maintained host location).

Relative to serverless computing, containers are also more difficult to monitor at scale. As applications grow, more containers are added, and their scattered/distributed nature makes monitoring them significantly more difficult. Despite these setbacks, containers are

highly flexible and portable. Serverless computing will not work well with long-running applications due to the limited time frame.

## Edge Computing

Edge computing allows for the use of edge containers. These containers are located as close as possible to the source of data to reduce the latency in executing an application. Edge containers stand in contrast to cloud containers. Cloud containers exist within a centralized cloud that can be far away from the end user's location, thus producing latency issues. On the other hand, edge computing networks have more extensive networks that require more secure network policies. Additionally, using edge containers rather than cloud containers offers a value proposition, as edge containers offload processing power from the cloud. Please refer to LF Edge Interactive Landscape to explore available industry solutions and tools for various edge requirements.

## Infrastructure As Code

Infrastructure as code (IaC) is an approach to managing a technology stack with software rather than hardware. IaC can be used to provision cloud systems and to virtualize different kinds of software environments and can be automated, freeing up teams from manual infrastructure management. IaC can also minimize configuration drift through effective detection and response, ensuring consistency of configurations and reducing human error that may in turn lead to runtime errors or compromise security.

For example, as a ubiquitous framework for IaC, HashiCorp Terraform enables teams to describe and automate provisioning of cloud infrastructure resources across different cloud platforms. More commonly, IaC frameworks are only compatible with the platforms upon which they were developed, as in the case of AWS CloudFormation.

IaC and containers work together when developing applications deployed to the cloud. Once deployed to the cloud, the container image alone does not describe the full application. Sharing a container image with others does not mean that someone else can easily run the application in the cloud, because they would still need to recreate all the infrastructure around the image. Rather, the complete application is best described with a combination of the container image and an IaC template containing all this configuration.

The automation of infrastructure provisioning for microservices has freed developers from manually performing tasks for routine operations and processes. Both microservices architectures and IaC are key best practices for what are called "Modern Applications."

# Application Development and Containers

## DevOps and DevSecOps

DevOps, or the combination of software development and operations, relies heavily on implementing strong version control practices along the software development and delivery lifecycle. DevSecOps is the practice of implementing security best practices at all points in the DevOps cycle, making it a shared responsibility of owners at every stage.

Containers support DevOps and DevSecOps by allowing for the rapid scaling of security features along with dispersed development cycles. Development teams can duplicate secure container images for each new version, rather than rely on the capacity of the security team to configure and secure each new development environment.

## GitOps

GitOps synthesizes Git, a distributed version control system to manage source code, also known as Git, with DevOps. It relies on a Git repository for applications, in tandem with an automated process, or agent, to ensure accurate execution of source code. Moreover, because Kubernetes relies on a set of facts, or declarations, in Git, GitOps can serve as the operating model of a Kubernetes production environment.

The benefits of GitOps include faster development and deployment, shortened meantime to recovery (MTTR), and reduced attack surfaces. However, scaling GitOps in a larger agency with many environments and applications means an increase in the number of Git repositories, which developers and operations staff may struggle to manage. Agencies must determine if the benefits of this approach outweigh the costs.

## Microservices

One benefit of container utilization is the improved ability to implement microservices. Microservices involve isolating individual application functions to operate as independent services. Each function can be updated or scaled without impacting service delivery for end users, making applications easier to scale and faster to develop. By distributing these functions rather than keeping them in a centralized database, the entire technology stack becomes more agile and service delivery continuity improves. Building an architecture for microservices that scales and takes advantage of the service delivery benefits typically requires the use of containers, which help keep operational expenses low in the cloud environment.

Microservices architectures share characteristics — automated deployment, intelligence in the endpoints, and decentralized control of languages and data — that can expand business capabilities.

Software is composed of small, independent microservices that communicate with one another through an intermediate set of functions and procedures, collectively known as an Application Programming Interface (API). Containers and microservices complement each other but can exist without each other. Containers function most closely to virtual operating environments.

## Common Container Use Cases

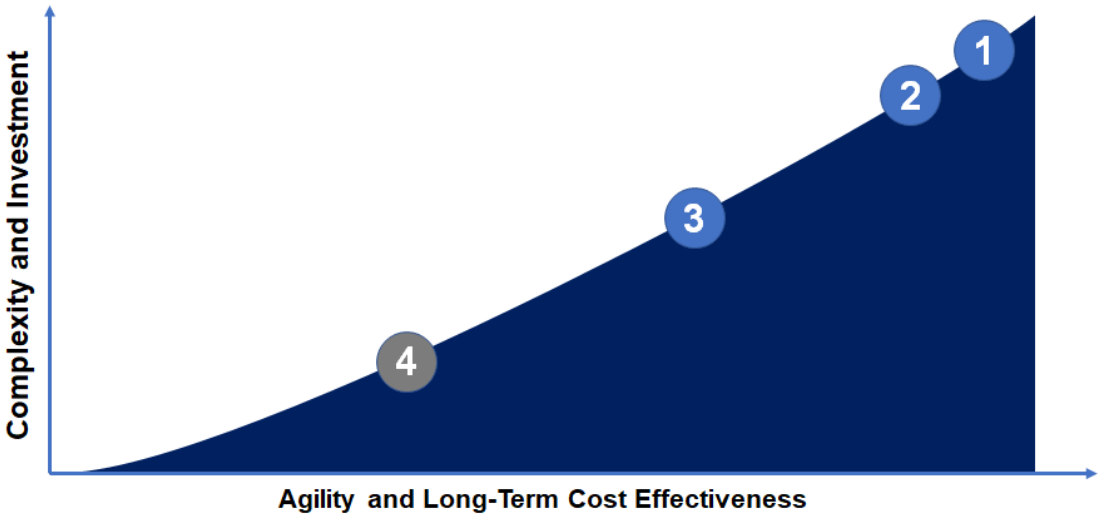


**Figure 2.** *Various use cases for containers have different initial investment and complexity concerns but can develop better agility and long-term cost effectiveness over time.*

### Priority Use Cases

1. **Develop New Container-Native Applications**
   Starting in a container-native setting is the highest priority and most self-explanatory of the use cases. When building new applications, agencies should consider building a container-native application, as this is the most resource-efficient time to do so.
   - ***Optimize Support for Microservices Architectures*** - *Microservices architectures are modular and are distributed across environments. Containers*

*enhance the flexibility that microservices architectures need to handle scaling and deployment, as compared to traditional instances.*[6]

2. **Rebuild Existing Applications**
   Organizations, particularly government agencies, use legacy systems and applications that have withstood the test of time. Unfortunately, this means that these same legacy systems will not easily migrate into a modern cloud architecture. When other solutions fail, organizations may need to rebuild applications from the ground up with built-in container support.[7]
   - ***Improve and Simplify CI/CD with DevOps Integration*** *- Containers support DevOps efforts to accelerate development, test, and production cycles within the CI/CD cycle. The modular and portability aspects of containers allow highly configurable and synchronized build and development environments.*[8]

3. **Refactor Legacy Applications**
   Organizations can choose to partially or completely refactor existing applications. A complete refactor typically requires more time and resources, but it can offer better performance and optimization for existing applications. A partial refactor can allow organizations to choose a single or multiple pieces of an application's existing functionality to refactor into containerized microservices, partially improving the performance of an application at a lower cost and resource point.[9]
   - ***Simplify Development and Deployment of Highly Repeatable Tasks*** *- Containers are highly repeatable because their infrastructure and configuration is uniform. Given how small containers are, they provide lightning-fast deployment in a repeatable way.*[10]

## Non-Priority Use Cases

4. **Legacy Application Migration**
   Containers can be used to make legacy applications more agile to work with and more cost effective as your agency migrates to the cloud. Since legacy applications are not cloud native, they require custom configuration in order to run in containers. The level of technical complexity, cost saving realization, and IT stack agility varies across configurations.
   - **"Lift and Shift" Existing Applications -** Organizations can use containers to meet modern development environments requirements, which often rely on containers to package and deploy applications. Using containers to "lift and shift" applications allows organizations to skip time-consuming code

[6] https://www.netapp.com/devops-solutions/what-are-containers/
[7] Ibid.
[8] Ibid.
[9] Ibid.
[10] Ibid.

rewriting processes and easily move applications into a modern cloud architecture. Although "lift and shift" is simple to implement, organizations typically are not able to harness all the native aspects of the cloud architecture/platform.[11]

# Container Challenges

## Security

The use of containers provides inherent security advantages and challenges that your agency should consider in its readiness assessment. The following content should be used to help guide your agency's assessment, but it is recommended to consult the NIST Application Container Security Guide (SP 800-190)[12] for more information on best practices for securing your agency's container infrastructure and services. Additional security resources are provided in Appendix 6.

The following are a sample of container-specific challenges. The list is not all-inclusive but provides an understanding of some of the unique security considerations associated with containers.

**Image Security -** The images on which the container is built can have their own security vulnerabilities. Implement policies requiring periodic image scanning for these vulnerabilities or non-approved image sources.

**Application Layer Sharing -** Because application layers are usually shared across containers within the same application, a compromise to the application can breach multiple containers at once.

**Container Privileges -** When containers are given the same privileges as the application host, breaches into these containers can mean access to the host device. Avoid running containers with a privileged flag, or implement more intricate security filters to limit OS capabilities.[13]

**Host Isolation -** In addition to heightened privileges, containers can also gain access to the host if they are misconfigured. Avoid sharing the host network and process namespaces to guard against breaches due to configuration mishaps.[14]

Containers run as isolated, distinct processes, but they are not inherently more secure than VMs. A simple misconfiguration in any part of the container development lifecycle can compromise the entire system. Security considerations and solutions can be better

[11] https://www.netapp.com/devops-solutions/what-are-containers/
[12] https://csrc.nist.gov/publications/detail/sp/800-190/final
[13] https://docs.docker.com/engine/reference/run/
[14] https://www.cloudmanagementinsider.com/5-security-challenges-for-containers-and-their-remedies/

identified by listing out the layers (or stages) of the container development lifecycle and identifying them at each layer.
**Table 1** showcases high-level security mitigation considerations for each layer. NIST Special Publication 800-90 provides in-depth guidance on securing a container ecosystem, and agencies should reference this publication to safeguard their container ecosystem.

**Table 1.** *High-level security mitigation considerations*

| Layer | Definition | Security Considerations |
|---|---|---|
| Network | Interconnected systems (both virtual and physical) that are part of container creation (e.g., code and image repository, data storage, cloud, etc.) | Segmentation, Zero Trust Network Architecture, Encrypted communications, Secure Service Mesh, logging |
| Host | Machine running the OS and runtime requisite for container creation and facilitating access to networked resources | Known-good baselines, security policies, vulnerability scanning, multi-factor authentication, system call restriction, container/environment isolation, privilege restrictions, logging |
| Runtime | The underlying infrastructure that allows the orchestrator to interact with the OS; consumes the config.json and root filesystem | Known-good baselines, security policies, privilege restrictions, logging |
| Orchestrator | Dynamically schedules container workloads within a cluster of computers; provides a standardized application definition file | Cluster policies, API access and plug-in restrictions, logging |
| Data | The code repository and storage and maintenance of optimized, secure images available for consumption; also, metadata and application definitions | Container hygiene, version control, access control, scans for sensitive items, logging |
| Cloud | Environment that may house some or all of the container development stack | Cloud security configurations scanned, templates |

With **Table 1** in mind, agencies can assess their current security maturity when delivering container-based capabilities. For small- to mid-sized agencies, reliance on a container distribution platform can be one way to manage security needs in a multi-cloud environment, because it provides a platform for securely moving between various cloud vendors' service environments.

One critical security control your agency should adopt is a service mesh. As noted earlier, a service mesh provides a dedicated layer for communication between containerized services. It becomes a comprehensive infrastructure to improve both security and network management of the container.

A service mesh can also provide policy controls and be deployed with proxy services to improve security management. One such proxy is the widely used Envoy proxy for cloud native applications, which is used by most service mesh products. Envoy provides a wide range of security services in an open-source format.

While smaller agencies could create a separate authentication network before they adopt a service mesh, they should create security solutions (such as authentications) as they mature into a service mesh.

## Workforce

Containers require employees to become familiar with a new development approach and tools for implementing containerization effectively. Training is a cost that agencies need to consider when adopting containers, as employees need to know how to use the containerization platform that your agency is using, as well as how to redesign applications for containers. To address workforce challenges, one approach is to pursue a managed service model for container adoption efforts along with continued operations. There is a clear lack of skilled specialists in this area, and the quickest path to a mature container management model is through partnering with companies that specialize in this area.

If your agency decides to develop internal skill sets for container management, there are several online private companies that offer training at a cost and free publications from open-source solutions such as Docker, Kubernetes, and their associated open-source supporting systems. It is recommended that agencies take a measured approach, starting small with very low complexity deployments and growing as your workforce skill sets grow.

Key roles for successful deployment include:

- Software Engineering
- Platform Engineering and Operations
- Reliability Engineering
- Build and Release Engineering

- Security Engineering

## Container Adoption Assessment

Container adoption happens over time and evolves with your agency's technical maturity. The following journey map (**Figure 3**) shows some overarching stages of container adoption maturity and the challenges they often entail, which are expanded upon in greater detail in Appendix 1.

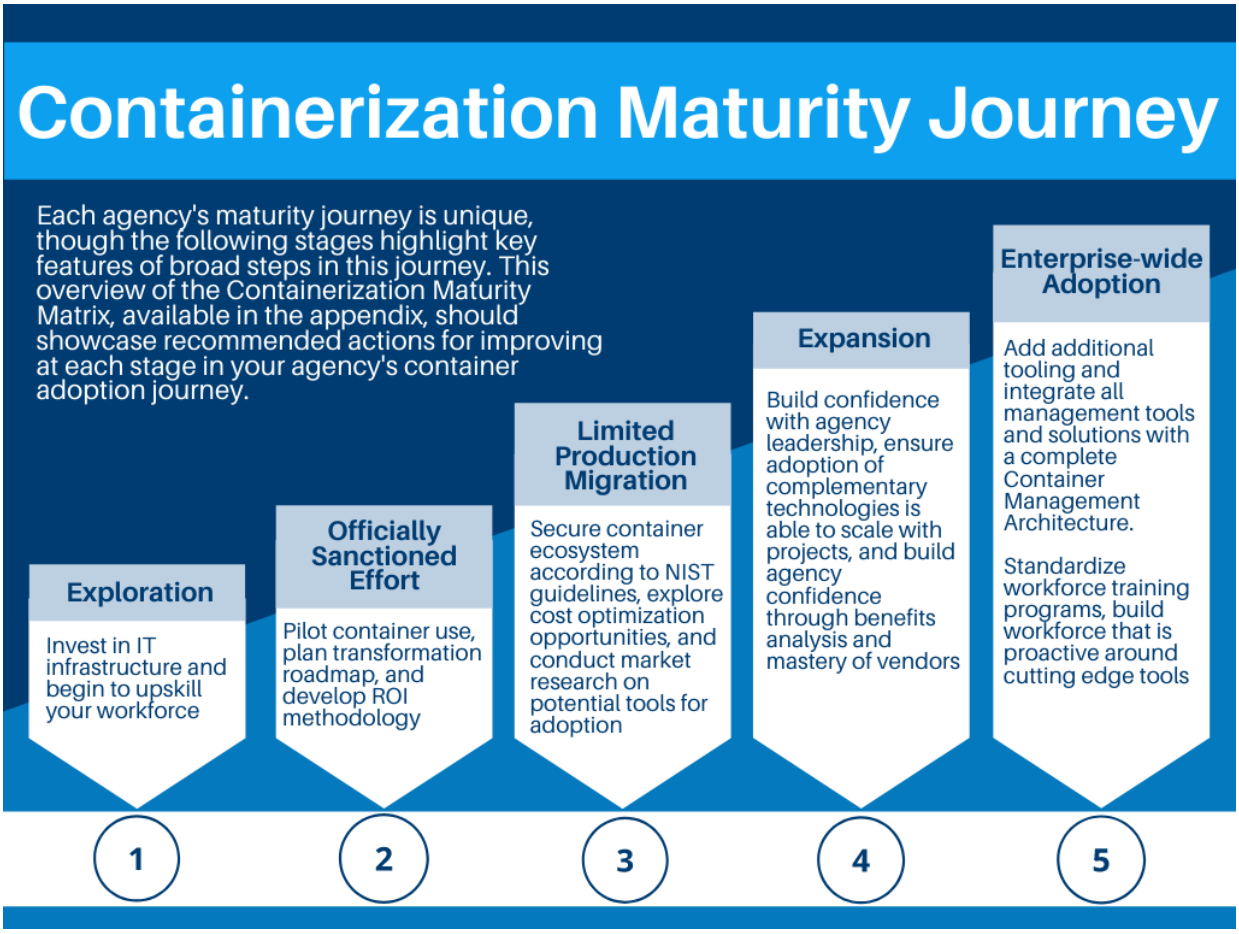


**Figure 3.** *Each agency's containerization maturity journey is unique, and this infographic provides an overview of the Container Maturity Matrix in Appendix 1.*

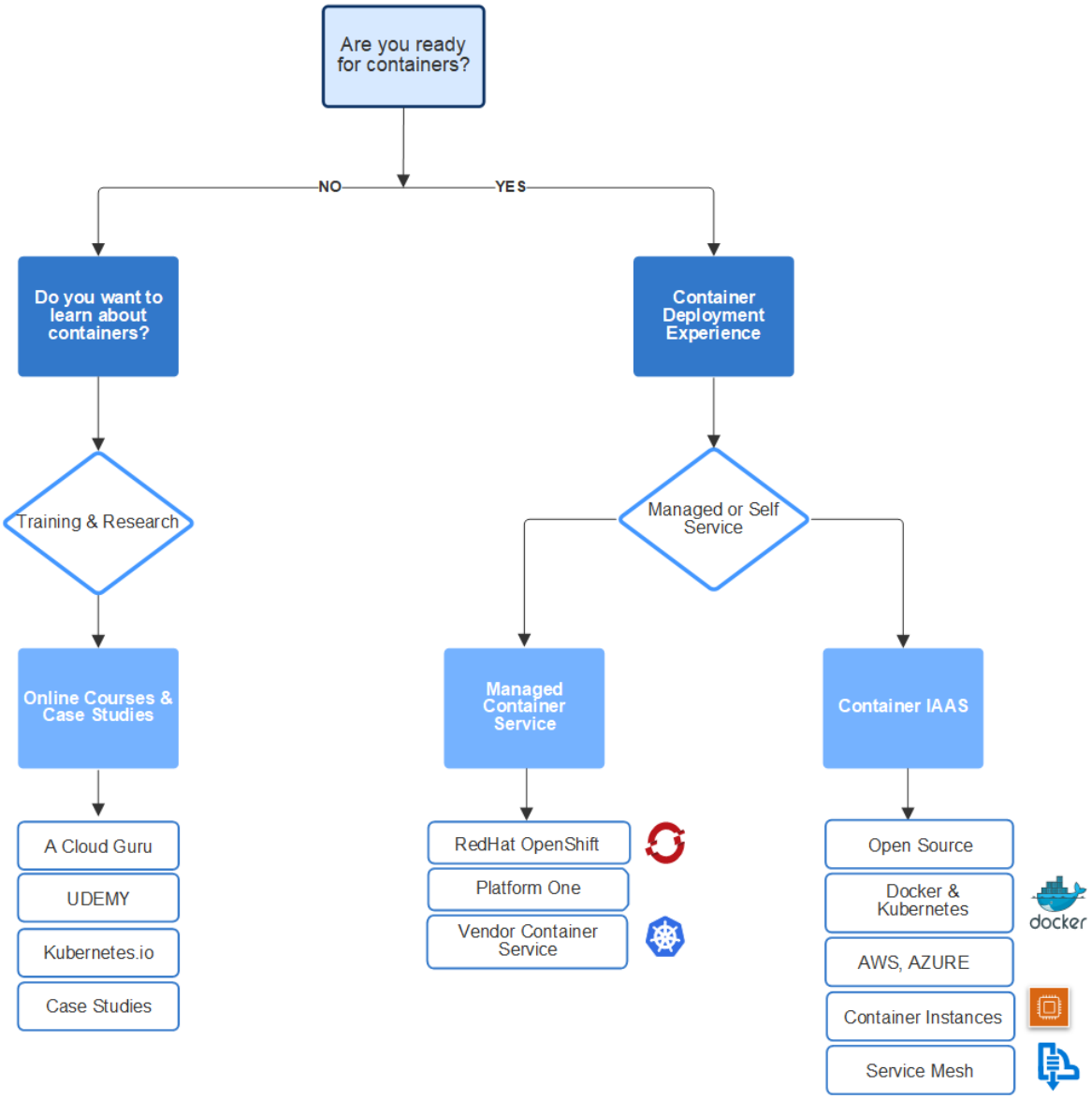


**Figure 4.** *This container decision tree provides agencies with a clear series of decision points that enable simple container readiness and viability assessments.*[15]

As your agency considers containerizing its applications, check how ready it is to transition to containers, as well as whether or not containers are the best solution. Please refer to the decision flow chart (**Figure 4)** to assess your agency's readiness and viability.

[15] Any references to vendors do not constitute an endorsement of their services.

# Container Service Delivery Models

**Table 2** shows recommended container delivery models based on your agency's purpose in adopting container technologies and its maturity stage. Note that, for a given maturity stage, different services may be recommended depending on your agency's purpose. We recommend you conduct thorough market research by evaluating FedRAMP approved providers, as many of them offer container services for each stage of maturity listed below.

**Table 2.** *Recommended container delivery models*

| Purpose | Maturity Stage(s) |
|---|---|
| **Container as a service:** Run a container without provisioning any virtual machines or adopting a higher-level orchestration service. | Officially Sanctioned Effort, Limited Production Migration, |
| **Container PaaS:** Allows customers to store Docker-formatted images. Used to create all types of container deployments. | Limited Production Migration, Expansion |
| **Orchestration and Kubernetes as a service:** Deploy orchestrated, containerized applications with Kubernetes. | Limited Production Migration, Expansion, Enterprise-wide Adoption |
| **Fully managed service:** Enable developers to deploy microservices applications without managing virtual machines, storage, or networking. | Expansion, Enterprise-wide Adoption |
| **Full autonomy:** Infrastructure as a service. | Expansion, Enterprise-wide Adoption |

Kubernetes has expanded significantly in the past few years, with many vendors emerging downstream in the software ecosystem. Upstream Kubernetes,[16] based on the latest version of the Kubernetes source code, ensures that the latest functionality is available and prevents your agency from being dependent on any particular vendor. For agencies with small container environments, these benefits may outweigh the costs associated with independently deploying and managing a container cluster. However, agencies with large container environments should consider Kubernetes vendors downstream[17] from the source code, because they offer features that make container environments easier to manage, from load balancing and auto-scaling to monitoring. At the enterprise level, some agencies may want to utilize a vendor's expertise to protect critical and sensitive IT infrastructure.

[16] Upstream Kubernetes is also known as native Kubernetes.

[17] Downstream Kubernetes is also known as distributed Kubernetes, software that is built on top of native Kubernetes and generally offers additional features to ease installation, monitoring, and security.

The below checklist can help your agency evaluate a downstream Kubernetes service:

1. After installation, provides Day 2 OS patching and enabled supported security management for the long term
2. Provides an Integrated Development Environment (IDE) support for container supported developer languages that speed up container adoption such as Quarkus and Spring
3. Is part of a large ecosystem that includes certified independent software vendors (ISVs), Operators, Telco Virtual Network Functions (VNFs), and AWS Marketplace, among others
4. Is FIPS, NIST, FedRAMP, and ISO-27001 certified[18]
5. Is fully compliant with HIPAA regulations
6. Meets CIS Benchmark security standards
7. Supports and is specific to the underlying OS
8. Supports migration/modernization of applications to cloud native
9. Supports emerging technologies such as edge, IoT, and AI/ML
10. Ensures that the service works well with adjacent, already supported products and services before release of version updates
11. Answers the below questions in the affirmative:
    a. Can I move or synchronize the IaC from on-premise to multi-cloud easily?
    b. Are the containers and infrastructure certified and thereby shareable to other government organizations?

# Conclusion

The decision to adopt containers as part of your agency's cloud environment comes with unique benefits and challenges for governance, DevOps, security, automation and the agility to rapidly innovate. The use of containers and microservices can help your agency modernize its software stack by allowing more rapid scalability, but these benefits should be weighed against other considerations to decide whether containers are the right choice to support your agency's mission. Using this document to better understand the pros and cons of container adoption can help your agency evaluate its technical maturity and readiness.

This guide contains resources to help your agency evaluate its readiness for and the potential value of container adoption. The decision flow chart (**Figure 4)** is a tool to determine the readiness and viability of your agency's transition to containers. The Container Transformation Maturity Matrix in Appendix 1 details the likely challenges, proposed solutions, and outcomes associated with each stage of the containerization maturity journey. To further assist your agency:

- Appendix 2 answers frequently asked questions
- Appendix 3 illustrates a generic use case

[18] The list of certifications provided here is not comprehensive. Others may also be relevant to your agency.

- Appendix 4 reviews agency case studies and lessons learned
- Appendix 5 provides an example of an Operator of Operators
- Appendix 6 provides supplemental information on security and service meshes

All of these resources are here to help your agency chart the best path forward.

For more information, please contact the Data Center and Cloud Optimization Initiative (DCCIO) Project Management Office (PMO) at dccoi@gsa.gov.

# Appendix 1: Container Transformation Maturity Matrix

The following tables detail the likely challenges, proposed solutions, and outcomes associated with each stage of the containerization maturity journey.

- **Table 3:** Exploration
- **Table 4:** Officially Sanctioned Effort
- **Table 5:** Limited Production Migration
- **Table 6:** Expansion
- **Table 7:** Enterprise-wide Adoption

**Table 3.** *Container Transformation Maturity Matrix: Exploration*

| Typical Agency Challenge | Recommended Action(s) to Address Challenges | Outcome of Action |
|---|---|---|
| Limited to no DevOps practices in place; agency still has some waterfall principles in application development processes | Invest in workforce upskilling and target specific DevOps skills and training to integrate with container efforts | Application development processes and teams that are prepared for seamless, efficient, and effective initial container adoption |
| Little to no workforce knowledge of or experience with containers | Develop business case for development modernization practices | Skilled workforce to support the initial adoption and development of containerized applications and systems |
| Low IT modernization budget or leadership prioritization | Showcase federal case studies to demonstrate potential ROI | Potential for increased leadership support and necessary funding |
| Current reliance on physical hardware with some mix of VMs | Invest in IT infrastructure upgrades | Improved infrastructure to better align with container adoption infrastructure needs |

**Table 4.** *Container Transformation Maturity Matrix: Officially Sanctioned Effort*

| Typical Agency Challenge | Recommended Action(s) to Address Challenges | Outcome of Action |
|---|---|---|
| Either a very limited number of experienced professionals for container efforts or a group of lightly trained IT professionals | Diffuse skills from experienced professionals via informal/formal training and integrate with governmentwide communities and training for low-cost upskilling | Moderately skilled workforce that is prepared to integrate key technologies and cloud efforts with container adoption |
| Lack of organized planning to scale and implement containers across the organization | Develop an initial transformation roadmap and series of container-specific objectives and goals over a 3- to 5-year format | Achievable and detailed plan with identified resource needs (people, technology, infrastructure, and funding) with clear outcomes and goals |
| Pilot efforts for containers struggle to gain visibility for leadership or fail to show quick wins and early ROI | Create a clear-cut ROI methodology that resonates with leadership. Build-in success stories from other federal agencies, if possible | Adoption efforts quickly move out of pilot status and into limited or long-term efforts |

**Table 5.** *Container Transformation Maturity Matrix: Limited Production Migration*

| Typical Agency Challenge | Recommended Action(s) to Address Challenges | Outcome of Action |
|---|---|---|
| IT teams become overwhelmed with the increasing complexity of container environment | Conduct market research on and consider adopting complementary technologies such as a service mesh and container orchestration tools to help manage a more complex container environment | IT teams move recurring or time-consuming tasks to automated technologies. Container ecosystem gains resource efficiency from new management systems, freeing up teams to focus on other value-add projects |
| Consistently running into security policy roadblocks, throwing off project and migration timelines | Incorporate and adopt principles from NIST Publication 800-53 Rev 5[19] | A proactive container adoption plan and project that aligns with existing federal security policies, limiting current and future roadblocks |
| Heavy reliance on managed services and difficulty reducing costs now that internal knowledge and expertise is increasing | Explore opportunities to reduce costs by moving to CaaS/PaaS/IaaS offerings | Balanced plan of mixed resources that maximize cost and government resources that gives agencies flexibility to scale at a reasonable cost |

[19] https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final

**Table 6.** *Container Transformation Maturity Matrix: Expansion*

| Typical Agency Challenge | Recommended Action(s) to Address Challenges | Outcome of Action |
|---|---|---|
| Lack of load balancing and auto scaling as resources and complexity grow | Expand the maturity of existing container orchestration tools and add integration from security, orchestration, scheduling, policy, and governance for a complete container management ecosystem | Growing container infrastructure that scales with container expansion projects and begins integrating tools to create a comprehensive container management architecture |
| Limited automation throughout container ecosystem | Explore opportunities for additional container automation tooling | Teams see a decrease/same amount of manual IT work, even as container complexity grows |
| Difficulty adapting to different cloud service providers and infrastructure (AWS, Azure, Google, etc.)[20] | Ensure that there is mastery through one vendor/provider before continuing onto others | Increased confidence in adopting containers for a specific vendor/provider |
| Lack of movement to containers for legacy applications | Conduct change management efforts to demonstrate ROI and benefits (use existing case studies) to Product Managers, System Owners, and executives | Product Managers, System Owners, and executives that are open to adopting containers on legacy applications and systems |

[20] Any references to vendors do not constitute an endorsement of their services.

**Table 7.** *Container Transformation Maturity Matrix: Enterprise-wide Adoption*

| Typical Agency Challenge | Recommended Action(s) to Address Challenges | Outcome of Action |
|---|---|---|
| Difficulty maintaining workforce skills with constantly changing market and industry products/tools | Consider investing in repeatable industry training programs for workforce (AWS, Microsoft, Google)[21] | A highly trained and skilled IT team that can proactively identify and plan for cutting-edge tooling to constantly improve the state of containers within the organization |
| Limited automation of optimization of systems (self-healing, self-configurable, infrastructure, etc.) | Add additional tooling and integrate all management tools and solutions with a complete Container Management Architecture | Container ecosystem and management system that automates the health and configuration of containers to optimize the performance, cost, and resource allocation of a portfolio of applications |

[21] Any references to vendors do not constitute an endorsement of their services.

# Appendix 2: Frequently Asked Questions

### How should we pick our first project?

Any first step involves piloting your container technology prior to any production efforts. Pilots should be formally sanctioned with a structured approach including pilot objectives and formal testing criteria. Once your workforce's technical maturity and confidence increases, start with small projects to build and test environments or build simple microservice-based applications within container environments.

### How do we accelerate to multiple projects?

Build upon successes and knowledge acquired during your pilot and initial adoption stages. Develop a standardized containerization pipeline, with consistent considerations for governance and best practices. Concerns such as management systems and scheduling should be accounted for in this pipeline. This will help ensure that multiple projects follow the same trajectory and are optimized for successful implementation. Adopt a container-first approach to deploy production once your agency moves past the initial maturity stages.

### We are hearing a lot about containers and automation. How do these concepts fit together?

Automation principles such as IaC and DevOps exponentially decrease the time to complete tasks, maintain consistency, and drastically increase productivity. The goal is to reduce overhead and inefficiency of performing manual tasks for routine operations and processes. The same concept applies to container adoption. Container orchestration solutions automate many of the routine manual tasks that administrators would normally undertake. These orchestration solutions automate the deployment, scaling, and networking of containers. Without automation, container solutions would be unable to meet the fast-paced process requirements of a DevOps pipeline with daily builds and deployments. Truly efficient organizations leverage automation wherever possible to improve ROI and use the full potential of the technology.

### Where can we find training resources for container implementation?

There are many training resources available for container implementation and adoption. Agencies can leverage basic free awareness training for containers and orchestration solutions from websites such as edX.org or consult with their vendors for recommended training partners. There are a range of paid training solutions available, ranging from low-cost, self-paced training (from vendors such as Udemy or the Linux Foundation) to instructor-led programs (from vendors such as Global Knowledge). GSA does not recommend any specific vendor and recommends agencies research training resources online and find training solutions that best fit their needs.

### Where can we find more educational resources on containers and other cloud and infrastructure topics?

Agencies can learn more about containers by visiting websites such as Container Journal to stay updated on the latest developments in this quickly evolving space. Agencies can also visit open-source project websites, such as Docker's and Kubernetes' home pages. Furthermore, major cloud vendors that provide container services will communicate up-to-date information related to services on their websites.

# Appendix 3: Generic Use Case

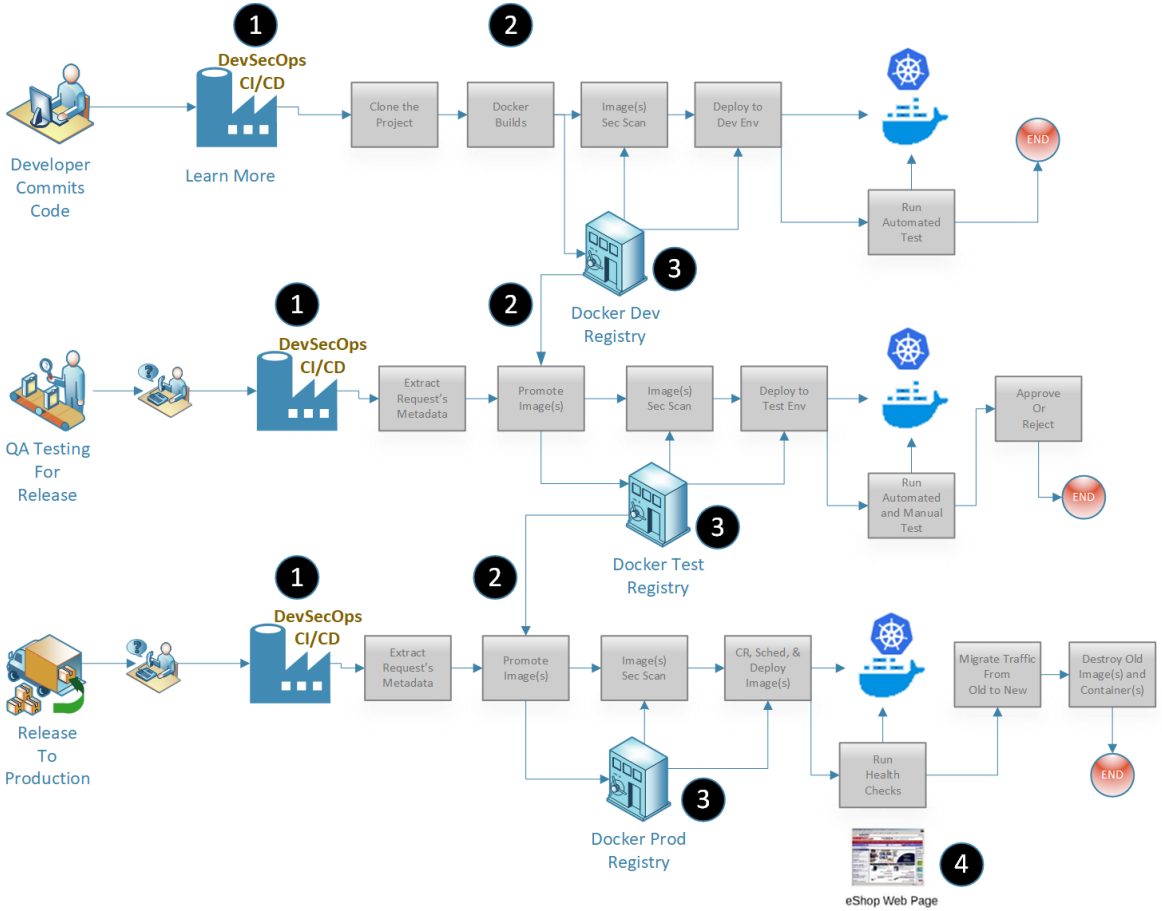


**Figure 5.** *This flow diagram provides a notional pipeline from container development to testing to production, using Docker and Kubernetes.* ***Table 8*** *on the following page provides links to the numbered steps.*

**Table 8.** *Numbered steps of the genetic use case and corresponding sections*

| Step | Corresponding Section |
|---|---|
| 1 | DevSecOps |
| 2 | Container Images[22] |
| 3 | Container Registries[23] |
| 4 | Notional Concept of Container Architecture Diagram |

# Appendix 4: Case Studies

As part of the development of the Container Readiness Guide, the team engaged in discussions with members of the federal community. The four case studies below provide non-attributional context for container efforts and specific best practices and lessons learned during planning, development, and implementation phases.

## Case Study 1: Cabinet-Level Federal Agency Container Journey

**Overview of Case Study and Project Timeline**

A large, cabinet-level federal agency with multiple components is in the process of developing a pilot and proof of concept container effort. In June 2019, a small team within the agency began working on the concept of building and utilizing containers to advance the agency's IT environment. In June 2019, an official proof of concept effort came together, and execution of the project began in September 2019.

**Development Specifics**

The team used a .Net C# Application for the initial proof of concept. A team of two, one developer and one administrator, conducted initial container efforts in a lab environment. This effort has since expanded to include efforts outside of a lab environment. The team engaged industry partners to gather advice and resources for the container journey. Today, the team is envisioning a container image repository and adding additional container efforts into agency-wide IT planning and strategy.

### Best Practices and Lessons Learned from Case Study 1:

**Define the Scope and Requirements Early and Dedicate Time Upfront -** The team stressed the lesson learned to spend more time upfront dedicated to the definition and development of the scope, and to the requirements of the proof of concept and pilot efforts. Several

[22] Figure 5 uses a Docker container image.
[23] Figure 5 uses Docker registries for development, testing, and production.

instances arose that could have been clarified in initial scoping efforts, costing the team an estimated two to three weeks in excess time.

**Hold Leadership Conversations Early and Often -** The team underestimated the conversations that needed to occur with leadership, as containers were a much different conversation than VMs. The team recommended that discussions with leadership about Docker and/or Kubernetes should start with a conversation about new and different platforms. The team also recommended initial component leadership discussions scoped around an overview on containers, container use cases, and lessons learned from other agencies to establish the rationale for why containers are necessary.

**Standardize and Streamline Cross-Environment Solutions -** The team wanted to create a solution that worked across their existing environments: AWS, Azure, and Google.[24] At times, they felt there were too many choices, and a streamlined or recommended solution may have been easier to follow and implement. The team recommended choosing just one environment to work at first, and then expanding efforts to include multi-environments in the future.

**Understand NIST 800-190 (Application Container Security Guide) First -** Understanding this guide and documentation was critical to the effort. Security countermeasure efforts were particularly taxing, resulting in a lot of back-and-forth with the security team. It would have been helpful to have the entire team understand the content of this document first. The team strongly recommended other agencies do the same.

## Case Study 2: Cabinet-level Federal Agency Application Container-based Platform

### Overview of Case Study and Project Timeline

A large, cabinet-level agency with multiple components began its container journey in 2016. The agency encountered an issue in which the existing hosting provider removed some of its websites. To mitigate this issue, the team turned to a container-based platform to host these sites. The team went enterprise-wide in 2018 after developing a successful proof of concept. This effort was conducted even before Kubernetes was available to the marketplace, and, while some of the tools are insufficient for today's purposes, the principles remain the same.

### Development Specifics

The team utilized their existing knowledge and strengths and chose Drupal as the platform for the container effort. A team of six engineers provided support for the first two years. As the platform matured to an enterprise-wide solution in 2018, more support was added.

[24] Any references to vendors do not constitute an endorsement of their services.

Today, the Drupal platform hosts many of the agency's websites, and has realized significant cost savings for the agency by reducing hosting and maintenance costs.

### Best Practices and Lessons Learned from Case Study 2:

**Compose the Team Correctly -** For this team, a large amount of success on containerization efforts came from DevOps and development work. Fundamentally, it is about investing in the team and/or contractors to ensure the team has the right people in place. On the management side, strong upward communication of needs and requirements was most effective. The team operated the platform with two very strong technical personnel.

**Narrow Scope of Effort -** For example, if your team knows Drupal and Apache, use that as your baseline for containerization. Do not upgrade or pivot to different services that your team is unfamiliar with, which makes the entire container effort easier rather than adding more "new" topics to the team's list.

**Procure the Right Contract Team -** The team lead recommended building procurement products that have constant evaluation of experience and skills to ensure that contractor teams have the right mix of skills (across Docker, Drupal, and other cloud products). The team does not recommend adding container efforts onto existing general IT and data center contracts.

**Consider Starting with Systems that Represent Lower Risk -** The team lead recommended that any team starting out on the container journey start with systems and applications that have no Personally Identifiable Information (PII), as they generally will have lower risk. The team considered themselves very pro-experimentation and configuration of systems without PII. The team lead recommended giving agencies the space to experiment in a lower risk environment by:

- Rearchitecting systems
- Decoupling systems to create microservices
- Using systems that your team is already familiar with

## Case Study 3: Cabinet-Level Federal Agency Mobile Application Platform

### Overview of Case Study and Project Timeline

A cabinet-level federal agency developed a cloud-hosted system that provides the infrastructure and hosting platform for Mobile Shared Services (i.e., common services used for mobile applications) and web components of applications used on mobile devices. These mobile applications connect directly to enterprise services.

**Development Specifics**

The agency developed fully containerized services running 53 mobile applications on 244 instances in AWS GovCloud using MongoDB, Oracle, and WebSphere.[25] The containerization effort expedited the migration and deployment process, but it added some complexity to the effort. A pilot was initiated prior to the start of deployment.

The team developed the platform using Docker with AWS infrastructure (EC2 instances) and AWS security products in place. The containers were deployed with Kubernetes clusters resources that were allocated for each container group, and a load balancer provided configuration and documentation.

### Best Practices and Lessons Learned from Case Study 3:

**Establish a High-Level Foundation to Lift and Shift "Fully Containerized" Applications and Services -** For this team, a large amount of success on containerization efforts came from critical observations and the categorization of fully containerized applications and not containerized applications of the 53+ production mobile apps. These efforts gave the team a clear understanding of the degree of readiness, ranging from high to moderate to low.

**Build a Basic Containerized Baseline First -** The team recommends the following process to build a containerized baseline early:

- Build application servers
- Build an environment using Docker Compose, or your applicable tool
- Establish reverse proxy
- Deploy in AWS

**Establish Migration Waves -** The team correctly identified a relatively high risk in migrating 53+ applications/services in one coordinated effort, due to unavailability of resources, incomplete migration artifacts, and other unresolved risk mitigations. The team used a team of developers to migrate each app/service in a wave, reducing risk. Whether waves or sprints, the team recommends pre-establishing a cadence to have a more modular and low risk migration.

**Establish Migration and Deployment Runbooks -** The team developed multiple runbooks to establish timelines, waves, and tasks, such as Firewall Rules, DNS, SQL connections, Image Registry, Orchestration, Monitoring, and IAM Access. These runbooks helped standardize procedures and operations.

[25] Any references to vendors do not constitute an endorsement of their services.

## Case Study 4: Cabinet-Level Federal Agency Enterprise-wide Container Journey

**Overview of Case Study and Project Timeline**

A cabinet-level federal agency began its container journey with the goal of moving DevOps to an enterprise-wide solution, OpenShift as a service to provide Container as a Service offerings and support the enterprise-wide creation and adoption of containers.[26] The agency is rolling this effort into production in 2021.

**Development Specifics**

As mentioned above, the agency started with OpenShift. The team used a Git repository to build a bridge between OpenShift and containers, allowing OpenShift to pull containers to be used from the Git repository. The team used Kubernetes as the container management platform. The team hopes to explore using Kubernetes to replace some Virtual Machines for high density work in the future.

### Best Practices and Lessons Learned from Case Study 4:

**Large Discrepancy Between Existing Skill Set and Optimization of Kubernetes Services -** Using and optimizing Kubernetes services requires a high level of skill and expertise. Most of the federal workforce currently does not have a high enough level of expertise to optimize Kubernetes benefits. The team recommends scaling efforts with Kubernetes to match scaling skills and expertise, as to better realize Kubernetes benefits over time.

**Enterprise-wide Adoption Challenges -** Along the way, the team did encounter mission areas/offices that had already procured SaaS solutions, particularly with IaC tools, and had to account for these as they scaled to the enterprise.

# Appendix 5: Example of the Operator Pattern

It's simple to install tools and services, such as the Operator pattern, once Kubernetes is installed. As shown on the following page, an Operator pattern simplifies a suite of resources to support agile, DevOps, or data science teams from the command line (**Figure 6**). Corresponding code can be found here. As an alternative, a GUI can be used for installation.

[26] Any references to vendors do not constitute an endorsement of their services.

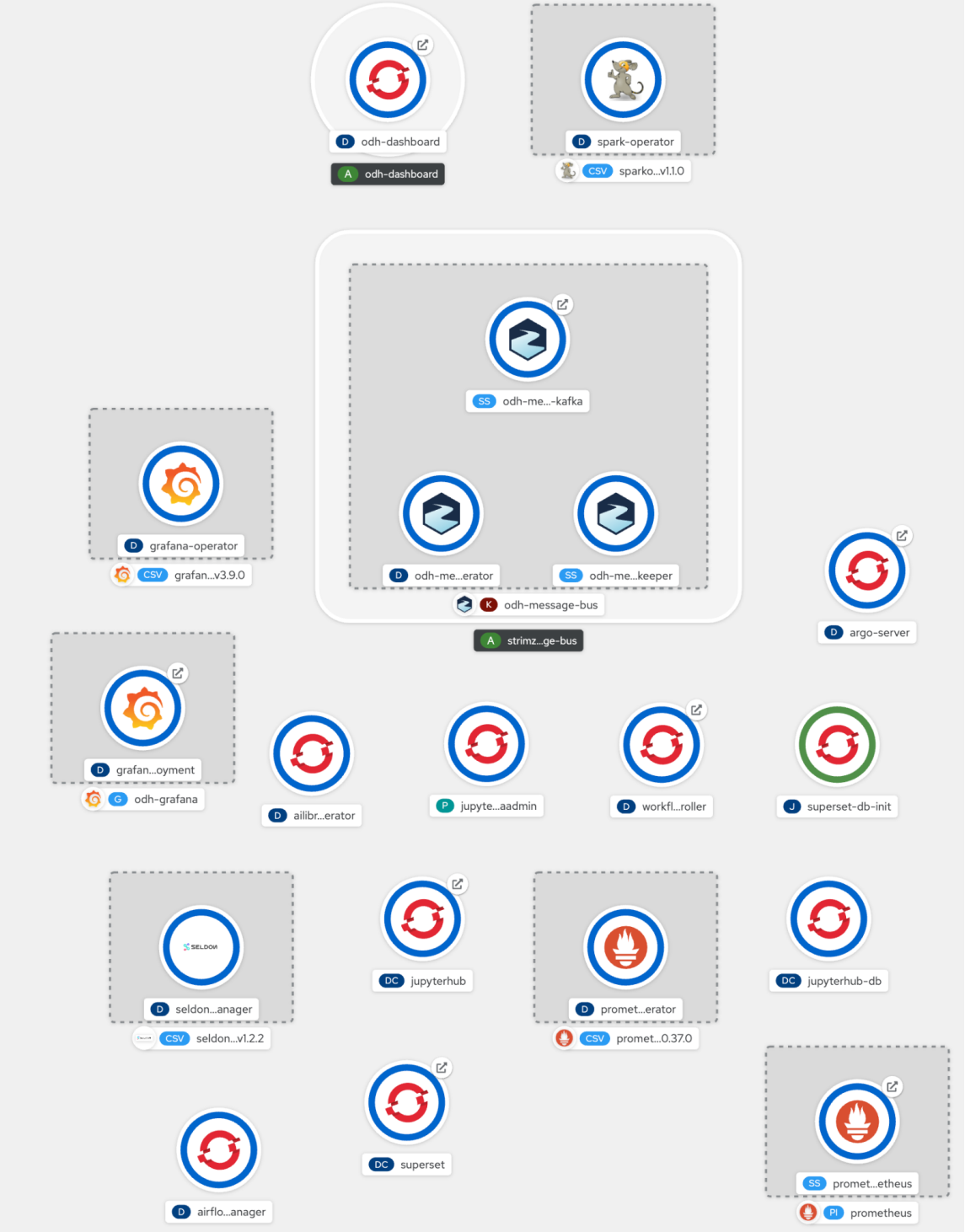


***Figure 6.*** *In the Red Hat OpenShift Container Platform, the Open Data Hub Operator installs other Operators, which have various Pods/containers that are automatically installed for use in AI/ML/data operations.*[27]

[27] Any references to vendors do not constitute an endorsement of their services.

# Appendix 6: Additional Resources

## Security Resources

Containers, being relatively new, are of interest to attackers. Addressing security challenges at defined levels of container creation and potential threats from the perspective of a well-defined framework is valuable. The table below is a draft mapping of threat-to-container based on the MITRE ATT&CK framework. The MITRE ATT&CK framework evolved out of Microsoft's Threat Matrix for Kubernetes, and is an evolving resource for container threat analysis. Cells in Green are proposed technologies and sub-technologies.

| Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion | Credential Access | Discovery | Impact |
|---|---|---|---|---|---|---|---|
| Exploit Public-Facing Application | Container Service | Implant Internal Image (NAME CHANGE) | Escape to Host | Build Image on Host | Brute Force | Container Resource Discovery | Endpoint Denial of Service |
| External Remote Services | Deploy Container | Scheduled Task/Job | Scheduled Task/Job | Deploy Container | Brute Force: Password Guessing | | Network Denial of Service |
| Valid Accounts | Scheduled Task/Job | Scheduled Task/Job: Container Orchestration Job | Scheduled Task/Job: Container Orchestration Job | Masquerading | Brute Force: Password Spraying | | Resource Hijacking |
| Valid Accounts: Local Accounts | Scheduled Task/Job: Container Orchestration Job | Valid Accounts | Valid Accounts | Masquerading: Match Legitimate Name or Location | Brute Force: Credential Stuffing | | |
| | User Execution | Valid Accounts: Local Accounts | Valid Accounts: Local Accounts | Valid Accounts | Unsecured Credentials | | |
| | User Execution: Malicious Image | | | Valid Accounts: Local Accounts | Unsecured Credentials: Credentials in File | | |
| | | | | | Unsecured Credentials: Container API | | |

## Notional Concept of Container Architecture Diagram

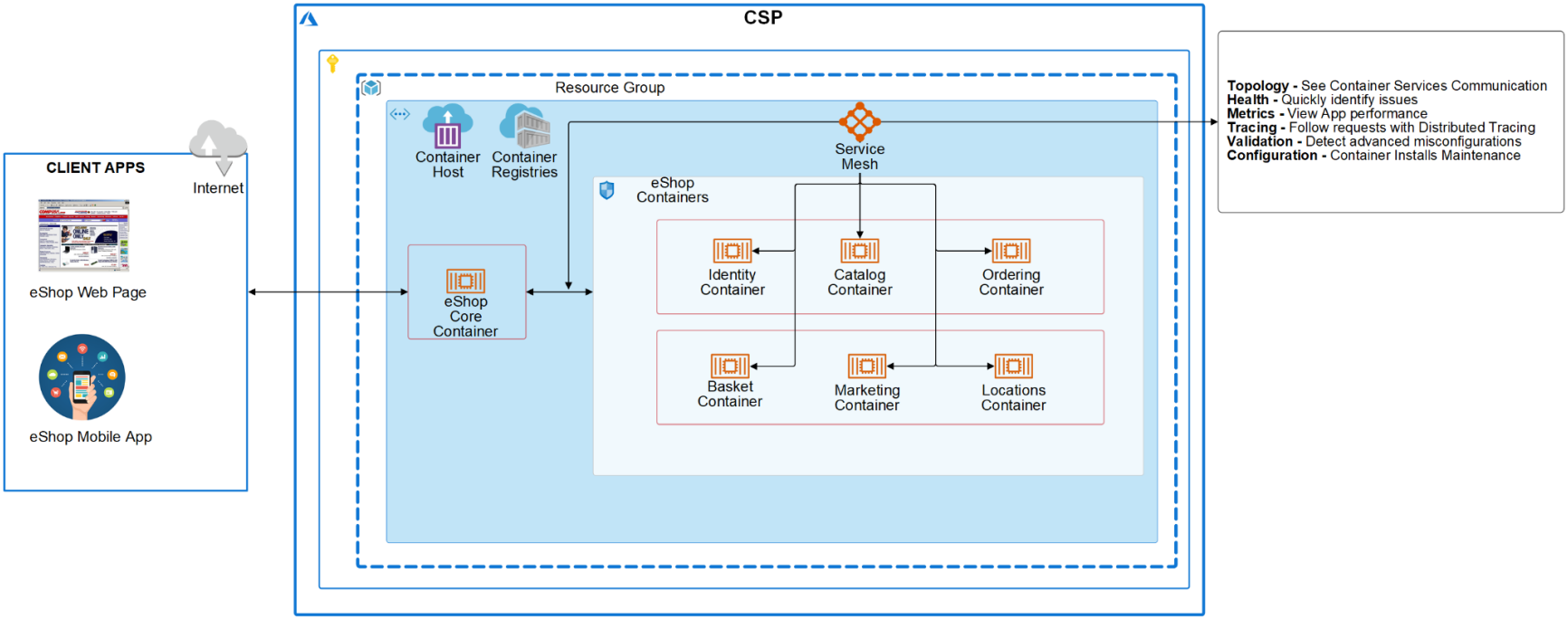


**Figure 7.** *This container architecture diagram highlights the advantages and benefits of a service mesh in managing container instances and optimizing the health and performance of client applications and containers.*

## Additional Resources List

- DevSecOps: Slaying the Myths of Container Security
- Docker Security Page
- Securing a Kubernetes Cluster
- NIST SP 800-190, Application Container Security Guide
- NIST SP 800-125, Guide to Security for Full Virtualization Technologies
- NIST SP 800-125A, Security Recommendations for Hypervisor Deployment (Second Draft)
- NIST SP 800-125B, Secure Virtual Network Configuration for Virtual Machine (VM) Protection
- DoD Enterprise DevSecOps Reference Design
- DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes
- DoD/DISA Container Image Creation and Deployment Guide
- Defense Security/Cybersecurity Authorization Working Group (DSAWG) DevSecOps Kubernetes Matrix Comparison
- DoD Enterprise DevSecOps Initiative (DSOP) Documents