VIRGINIA IT AGENCY

Commonwealth of Virginia

Enterprise Architecture Standard (EA-225)

Enterprise Solutions Architecture

Containers

vita.virginia.gov

Sept 22, 2025

Revision History

Version History		
Revision	**Date**	**Description**
0.9	4/9/2025	Initial draft document created
1.0	9/1/2025	Finalized language for initial publication

Review Process
This requirements document was posted on the Virginia Information Technologies Agency's (VITA) Online Review and Comment application (ORCA). All agencies, stakeholders, and the public were encouraged to provide their comments through ORCA. All comments were evaluated, and individual commenters were notified of action(s) taken.

Requirements and Agency Exceptions
The requirements included within this document are mandatory. Agencies deviating from these requirements must request an exception for each desired deviation, and receive an approved *enterprise Architecture Exception* via Archer, prior to developing, procuring, or deploying such technology, or not complying with a requirement specified in this document. The instructions for completing and submitting an exception request are contained within the *Commonwealth Enterprise Architecture Policy*.

Glossary
As appropriate, terms and definitions used in this document are in the COV ITRM IT Glossary. The COV ITRM IT Glossary is available on the VITA website. Additional terms and definitions unique to this standard can be found in the Definitions section of this document.

References/Links
Other documents referenced or linked in this document are additional resources that agencies or workers may consult to find best practices and guidelines or obtain increased understanding. Unless expressly stated in this document, references or links do not incorporate or require compliance with such other documents.

Contents

Introduction

Vision and Strategy	
Vision	
This standard seeks to enable an efficient and secure way for COV agencies to utilize cloud-based container services. It provides structured guidance on how container technology is to be built, validated, and deployed within COV public cloud environments. Containers can provide management and runtime benefits to an organization, but without proper oversight, can elevate agency risk. Providing a set of requirements and a structured approach to container lifecycle management allows agencies to utilize this technology in a compliant and secure way. Additionally, agencies are also aided by leveraging a VITA managed enterprise container repository that provides build, validation, and deployment tooling to largely automate these mechanisms, thereby reducing the efforts an agency would otherwise take to deploy to the cloud. Reducing or eliminating work to comply with this standard allows agencies to focus on the functions that serve their mission. Running containers in the public cloud permits many governance functions to be performed in an automated and scalable way. Structured descriptions of software defined infrastructure and security controls in the form of JSON can be received from cloud providers and checked by software to validate compliance. The COV efforts to migrate computing services to the cloud can be accelerated by using modern techniques that can also provide a new model for a scalable governance approach. Standardized deployment approaches create efficiencies in governance: the number of technology patterns is reduced which diminishes the complexity that agencies experience when developing a system for deployment. Consistent container formats ensure smooth orchestration, auto-scaling, and failover mechanisms in large-scale deployments, which is what the COV is. Developers can focus on application logic instead of dealing with environment-specific dependencies. Standardized security practices that are embedded into container templates help mitigate risks like vulnerabilities and misconfigurations. Deployment via an automated continuous integration/continuous delivery (CI/CD) pipeline causes manual steps to be eliminated. Finally, establishing a common operational structure can better foster tool compatibility across agencies and suppliers. This can result in reusability of function, storage and monitoring approaches. Without consistency, technology will be consistently re-invented and duplicated, resulting in harder security efforts, sprawl and therefore cost and risk.	
Strategy	
Objective 1	Establish consistent and secure deployment practices and methodologies for containers.
Objective 2	Establish role responsibility and accountability in governance activities.
Objective 3	Provide baseline requirements for the enterprise registry.

Objective 4	Provide platform agnosticism and portability.
Objective 5	Enable scalable, secure, resilient, and modular architecture.

Background

Containers have become a popular mechanism to bundle, deploy and manage software applications, especially in the cloud. Containers bundle the entire dependency tree required for running an application as a single unitized image which decouples the dependency between a container and the host machine's file system and library structure. Importantly, system level dependencies are embedded in the container image, which emancipates the application from the runtime operating system, thereby simplifying containerized application deployment. Conversely, in a traditional deployment environment, the application requires the operating system to provide environmental dependencies. This can result in non-explicit dependencies between potentially unrelated applications. This results in an administrative overhead to ensure that all applications in a VM environment rely on compatible system libraries.

Compute and storage resources have fallen in cost and can be made available essentially on-demand, which opens the door for new execution models, which containers are an example. As the COV moves applications to the public cloud, containerization can be used to facilitate this migration.

Purpose

This standard helps to enable safe and responsible utilization of container technologies across COV agencies by establishing common, automated practices that reduce risk, improve governance, and which are supportive of agency technology objectives.

Some of these requirements are only logically applicable when building or running a custom application. Others still apply when using third-party provided software which is to be bundled into a container. If the container is provided by a third party as a full image, the container must still be validated as meeting security rules, including scanning for vulnerabilities, and connectivity rules. If the packaged system cannot meet these rules, a virtual machine-based deployment should be used instead.

Scope

This standard is currently scoped to cover container production deployments. On-premises based container deployments must be compatible with and ready to move to an approved public cloud solution within 5 years. Those systems deployed with containers prior to the publishing of this standard should develop a plan to migrate to an approved solution within 5 years unless otherwise approved by VITA EA.

This standard is applicable to all Commonwealth agencies (hereinafter collectively referred to as "agencies") that are responsible for the management, development, purchase and use of information technology resources in the Commonwealth of Virginia.

This standard does not apply to research projects, research initiatives, or instructional programs at public institutions of higher education.

In addition to the requirements below all COV IT technology solutions comply with the standards found on VITA's Policies, Standards & Guidelines.

Authority

Code of Virginia, §2.2-2007	Powers of the CIO
Code of Virginia, §2.2-2007.1	Additional duties of the CIO relating to information technology planning and budgeting
Code of Virginia, §2.2-2009(A)	Additional duties of the CIO relating to security of government information
Code of Virginia, §2.2-2012(A)	Additional powers and duties related to the procurement of information technology

Principal Requirement Objectives

The standard uses the following tenets to establish the requirements:

- **Risk Reduction** – seek to reduce the presence of risk in a system
- **Traceability** – identify the cause or source of a specific system component or property
- **Consistency** – by establishing a repeatable property of the system, the system becomes easier to maintain, understand, monitor and change.
- **Controls** – support the measurement of compliance within a system to applicable policy
- **Simplification** – seeking to reduce the complexity of a system leads to easier manageability amongst components as dependencies are lessened and easier to identify.
- **Scalability** - seek to ensure that a system can naturally grow in the amount of work it can perform
- **Continuity** - ensuring that a system is resilient to single points of failure
- **Portability** – seek to ensure that the system isn't tightly coupled to specific external dependencies and that the correct abstractions exist.

Each requirement is tagged with the tenets that it supports. These tags follow the requirement text itself. More information is found in the appendix.

VITA shall provide an enterprise-wide container registry available to all COV agencies if they wish to use it. The COV enterprise registry shall be partitioned by agency and controls established within the CBTI. Agencies can also set up their own private registries if they meet the requirements set forth in this standard.

Perspective 1 (Construction and Build)

The construction and build perspective centers around the process of creating the container by working with either a pre-built image from a container registry or developing a custom container. Templated container images help meet baseline requirements to assist agencies to stay compliant with COV security and architectural standards.

CB-001	Containers shall only be bundled with required dependencies for the application logic they contain. Risk reduction Simplification
CB-002	Only system and application dependencies that are sourced from approved, baselined COV system images shall be bundled in a container. Risk reduction
CB-003	Session and long-term state shall not be stored in the container image. Scalability Portability
CB-004	Session and long-term state shall be placed in network accessible shared storage accessible to authorized accessors. Scalability Continuity Portability
CB-005	Container environments shall be configured using the principle of least privilege. Risk reduction

CB-006 Container resources shall not be directly accessible from an external network without a VITA CSRM approved network filtering and detection device in place. Risk reduction

CB-007 All outbound communications shall only be initiated by processes on the container itself. Risk reduction Portability Scalability Continuity

CB-008 Container instances shall only accept network connections on the container's designated service request port. Risk reduction Scalability

CB-009 Container images shall use continuous integration/continuous delivery (CI/CD) to facilitate building, bundling, testing, validation, packaging, and deployment. Risk reduction Portability Scalability Traceability

CB-010 Containers that are to be used as part of a production system shall be functionally tested in a non-production environment prior to deployment as part of a continuous integration/continuous delivery methodology. Risk reduction Consistency

CB-011 Container images shall not have secrets such as tokens, credentials, private keys or other confidential information bundled within the image or image template. Risk reduction Consistency Simplification Portability Continuity

CB-012 Containers shall access an approved secret store during runtime for any needed secrets, configuration or other authentication related data. Risk reduction Consistency Simplification Portability Continuity

CB-013 Development activities requiring container deployments shall occur in a non-production classified environment. They shall be deployed to a non-production container registry. Traceability Controls Risk reduction

CB-014 Deployed container images shall be immutable. Risk reduction Traceability Governance Controls Consistency

CB-015 Any changes to a deployed container's image shall require a new container image to be built, validated and deployed. Risk reduction Traceability Controls Consistency

CB-016 Only VITA EA approved source repositories shall be able to deploy to the COV registry. Risk reduction Controls Traceability

Perspective 2 (Validations)

Validations are defined as the pre-deployment verification that the contents of the container meet security requirements. This should include security scanning, validation to baselined OS libraries, checking for forbidden items, such as shells, or non-required dependencies.

V-001 All pre-deployment testing and validation processes shall occur in an automated manner with no manual processing. Risk reduction Traceability Simplification

V-002	Containers shall be scanned by VITA CSRM authorized security detection and compliance tools resulting in a successful pass prior to deployment. Risk reduction
V-003	Container images shall be scanned for vulnerabilities using a VITA CSRM authorized vulnerability scanner prior to deployment to a production registry. Risk reduction
V-004	Containers must have vulnerabilities remediated according to the COV IT Risk Management Standard prior to deployment. Risk reduction
V-005	The CI/CD pipeline shall log, in structured JSON form, the validations that have been performed on the released image in accordance with VITA EA requirements. Controls Traceability
V-006	The designated deployment administrator for the container image shall publish the documentation to the VITA EA designated location. Controls Consistency

Perspective 3 (Deployment)

This section concerns releasing the container image into production once all the necessary validations have occurred.

D-001	Containers shall only be deployed to VITA EA approved container management and deployment services. Consistency Risk reduction
D-002	Containers shall only be deployed to U.S. regions and availability zones. Continuity Risk reduction
D-003	Systems that are to be packaged and released via a container shall use automation to record a bill of materials of all contents. Simplification Traceability
D-004	Each container image shall be assigned a unique and immutable tag as a means of identification. Traceability Consistency
D-005	The registry shall store the container image's unique tag. Consistency Traceability Scalability
D-006	A container registry shall only accept images that have at least one tag unique to that registry. Risk reduction Scalability Traceability
D-007	The final container image shall be signed by the deployment team prior to deployment to demonstrate that it has successfully gone through the validations process and its security has been verified. Risk reduction Traceability Controls
D-008	The final container image checksum shall be electronically signed by the CI/CD system and stored as associated metadata in the registry for the lifespan of the container image. Traceability

D-009 Containers shall be registered in and deployed using a production classified container registry. Traceability

D-010 Containers that are to be deployed to a production system shall be deployed with a fully automated pipeline, in which no manual steps are performed during the build, validation and deployment phases. Traceability Simplification

D-011 Errors encountered during build, validation or deployment must be remediated prior to the container deployment. Risk reduction

D-012 Containers cannot be manually manipulated during the build, validate or deployment phases. Traceability Consistency Continuity Risk reduction

D-013 Only container images that have been sourced from the enterprise registry shall be deployed into a COV production environment. Traceability Risk reduction

D-014 Containers shall only be deployed to an on-prem QTS registry after providing a written rationale and approved by COV EA based on: Risk reduction

- confidentiality
- regulation
- agency or citizen risk

D-015 The enterprise registry shall only allow container images to be visible to the accountable agency. Risk reduction

D-016 Agencies shall be accountable to keep their deployed container images current and secure. Risk reduction

Perspective 4 (Monitoring)

Monitoring activities focus on assessing whether the container and its environment is performing per the design requirements, remains free of known vulnerabilities, and the collection and processing of information needed to support these activities.

M-001 Monitoring shall be instrumented at the container group level, and container instance level in accordance with current CSRM security policies. Risk reduction Continuity

M-002 After deployment, if a security vulnerability for a container's assets is published or detected, the container shall be rebuilt and redeployed in accordance with current CSRM security policy. Risk reduction Continuity

M-003 Container and Container registry solutions must be integrated with VITA CSRM identified security solutions for monitoring and compliance. Risk reduction Controls

M-004 CSRM shall have the ability to remove container images that are found to contain vulnerabilities. Risk reduction

References

For further information on container management practices see NIST Special Publication 800-190.

Definitions

As appropriate, terms and definitions used in this document are included in the COV ITRM IT Glossary.

Availability Zone	An availability zone (AZ) is a subdivision within a cloud region that is designed to maximize fault tolerance and availability. AZs are made up of one or more data centers that are separated by significant distances, often miles apart. This separation reduces the likelihood that more than one AZ will be affected by a disaster, such as a power outage or natural disaster.
Container Orchestration	Container orchestration is the automated process of managing and coordinating the deployment, scaling, and networking of containerized applications, simplifying the complex tasks associated with managing large numbers of containers. Container orchestration systems automate the deployment and scaling activities of the container runtime engines.
Container runtime	A platform that allows you to instantiate container images as an executable process, providing the bridging between the container's internal resources and the host kernel. Often container runtimes provide a control plane to provide the necessary rules and controls to handle scale out events, resource access and limits. The container runtime often works together with a container orchestration system.
Continuous Integration/Continuous Delivery (CI/CD)	Continuous integration is focused on automatically building and testing code, as compared to continuous delivery, which automates the entire software release process up to production. Generally, incorporating both resolves to a set of software development practices that automate the process of building, testing, and deploying code changes frequently, ensuring that new features and bug fixes are quickly integrated into a shared code repository and readily available for release to production environments.

Dynamic Analysis	Refers to the practice of monitoring, testing, and evaluating a running instance of an application for problems, vulnerabilities and defects that are hard to detect using static analysis. Dynamic analysis can reveal performance problems, how the system interacts with the environment it runs in, and security problems. By performing a runtime analysis, accurate performance metrics can be gathered, which can highlight resource contention issues, memory access bottlenecks, and other properties that appear only when the code base is installed in the environment it operates within.
Functional Testing	Functional testing is a process that validates if the software meets the specified requirements. The objective is to ensure that the software fulfills the intended purpose and that the functionality it implements works as specified.
Hardening	A process of securing a system by exposing only resources that are required for operational use, ensuring that permissions are only those required to fulfill the system objective. Additional activities can include scanning and remediating the system for known vulnerabilities.
Immutability	Immutability means an unchangeable, constant form. In the case of containers, this means that, once a container image has been constructed, and deployed into a registry for use, it cannot be altered, added to, or otherwise changed. Every resource inside that container should be a read only resource.
Inbound Connections	These are defined as network connections made **to** the container and are initiated externally to the container.
Long-term State	The storage of information for extended periods, typically longer than a user's specific login session, for future reference, compliance, or historical analysis. Containers typically are part of a distributed application and would use network accessible resources. Examples of long-term state storage mechanisms are shared file systems, network accessible object stores and databases.
Microservices Architecture	This is a design approach that breaks down an application into smaller and separate parts, typically running distinctly (such as in separate run-times) from each other. Often, these smaller components are then integrated into sequences that comprise business level transactions.

Outbound Connections	These are defined as network connections made **from** the container and are initiated by the container to a system external to the container image.
Principle of Least Privilege	A security concept that limits access by an actor or system to the minimum resources and permissions needed to perform their tasks.
Secrets Management System	A secrets management system helps to securely encrypt, store, and retrieve credentials for your databases and other services. Instead of hardcoding credentials within application code or environments, a secrets management system will retrieve your credentials whenever needed. A secrets management system helps protect access to IT resources and data by decoupling rotation and management of secrets.
Session State	The persistence of data associated with a specific user's interaction with a web application across multiple HTTP requests, allowing for a more dynamic and personalized experience.
Static Analysis	Refers to a practice where code is analyzed prior to execution. Tools that facilitate static analysis can look for defects in code structure, code that is non-compliant to guidelines, harmful code patterns, security vulnerabilities and other information that can be gleaned from the code base or documentation itself. Static analysis is used during the coding and can provide a control point in a build pipeline.

Tenets and their Rationale

Tenet	Definition	Discussion
Risk Reduction	Seek to reduce the presence of risk in a system	Risk can be classified and measured in multiple ways. A risk can be classified by the impact it might have to a domain, such as security, financial, operational and technical. Risk can be quantified by its potential impact to a system or environment, such as minor, major, severe or critical. As systems increase in complexity and in number of interacting or inter-dependent components, a problem can cascade and impact other components, increasing the impact of the original problem. Taking active measures to isolate risk and recovery processes increases system resilience and reduces the impact of a problem across the whole system, which helps to reduce the total system risk.
Traceability	Identify the cause or source of a specific system component or property	Being able to identify how and what produced, modified or removed an element of the system can help with problem solving, identifying root cause and process compliance. In systems that employ multiple components, traceability becomes important for helping to identify the source of a system event or failure, which in turn helps reduce the impact of problems and helps to guide the COV to the right solution.
Consistency	By establishing a repeatable property of the system, the system becomes easier to maintain, understand, monitor and change	Systems that are consistent are cheaper, easier to troubleshoot, easier to maintain, and easier to scale. Conversely, a system that has multiple elements that perform the same function but are implemented differently requires more time to manage, creates additional cost and expense to maintain, and creates a drag on COV resources. This is because the same functionality must be learned multiple times in different ways, tested in different ways, and licensed in different ways to achieve the same functional outcome. Risk is higher because the number of

		distinct elements with different ways of failing increases.
Controls	Support the measurement of compliance within a system to applicable policy	Controls in this context are mechanisms that are placed into processes to help measure and prove compliance with policies. They typically are classified as risk management mechanisms as they help reduce organizational exposure to risk by implementing a means to observe and measure risk. Controls can be useful because, cumulatively, they help to quantify and measure an organization's or system's total risk exposure and provide information into a feedback cycle for addressing of those risks.
Simplification	Seeking to reduce the complexity of a system leads to easier manageability amongst components as dependencies are lessened and easier to identify	Reducing the number of elements that are required to facilitate a system is a key component of a good design practice. This is because there are typically less points of failure, and less redundant interdependencies. This tenet complements systemic organization: by designing an organized system, simplification becomes easier since patterns can be identified and leveraged to reduce the number of total elements. Each component typically requires a scaffold of supporting infrastructure: testing, documentation, validations, support mechanisms, and integration into the whole system. By aiming for simple solutions, we can help to reduce total cost to the COV and reduce the risk that the COV must bear for the total system.
Scalability	Seek to ensure that a system can naturally grow in the amount of work it can perform	Systems that have a linear or better relationship between the operational resources they have access to and the work that can be performed are considered scalable. Systems with these properties must be intentionally designed. Otherwise, resource and other contentions can develop. Scalability is typically done in a horizontal manner, with multiple elements performing the same task. This is in contrast to a vertical design, where elements don't replicate to handle work, but instead perform the work in a faster manner.
Continuity	Ensuring that a system is resilient to single points of failure	Systems that are organized, simple and scalable typically exhibit resilience in their operation, which enables continuity of function. This is because the

		system has multiple instantiations of the same component. The loss of an active component in a system reduces the capacity of the system to handle the workload in proportion to the total number of components actively handling the load. This is a general rule of systemic risk: if a system is comprised of 20 components that provide a function and one component fails, the failure impact is a capacity reduction of 1/20th (or 5%) of the total. If a system is comprised of 3 components and one of the components fails, the system capacity is reduced by 1/3rd, or 33%.
Portability	Seek to ensure that the system isn't tightly coupled to specific external dependencies.	When an element of a system is configured to directly point to an external dependency, it can be said to be tightly coupled to that external dependency. If the external dependency is then changed such that the dependency is no longer accessible by the referencing element, failure can occur. A better practice is to abstract dependencies between components, so that the linkage can be controlled externally. This is analogous to a requestor using a middleman, who facilitates the communication to the person that can supply the information that the requestor can use. This serves to *decouple* the requestor from the provider. A simple example of this is DNS. Instead of referencing external dependencies within a container by direct IP address, DNS enables the container to reference a service by name, which can be a constant in the configuration or build files. The DNS system provides the correct IP address depending on the locale, availability zone or other dynamic aspects of a deployment. In this case, the dependency is abstracted by externalizing the reference to the environment of the system, instead of being hard coded to a specific resource which may not be available if the container runs in a different availability zone or region.