

Service-Based Reference Architecture for NAS Automation

Version 1.2



Reference Architecture Team

August 2022

Acknowledgments

This document was created by the collaborative efforts of the Automation Evolution Strategy team, under the leadership of Rob Hunt (Director, Integrated Services and Analysis). Senior guidance and direction were provided by Steve Bradford (Chief Scientist, Architecture and NextGen Development), Natesh Manikoth (Chief Data Officer), Josh Gustin (Deputy Director, Air Traffic Systems), and Sean McIntyre (Director, Solution Delivery Services).

Automation Evolution Strategy team members included:

Name	Organization
Abilla, Walter D	AJM-22
Adams, Carleen	ANG-B22
Ajaegbulemh, Lakaisha	ANG-B22
Baird, James	ANG-B21
Beams, Jonathan	ADE-200
Bigio, Carmen	AJW-1761
Bohannon, Karen	AJM-3121
Buntin, Marc	AJM-13
Burdhimo, Gian	AJM-21
Burdick, Chris	AJM-222
Burgess, Shirley	AJM-0
Byrd, Warren	AJM-2522
Chaloux, David	MITRE
Cooley, Dale	AJM-24
Coplen, Kim	AJM-2
D'Avella, Anthony	AJM-24
Davison, Talia	ATM-221
Dinatale, Nick	ADE-230
Exum, Monique	MITRE
French, Jeff	Booz Allen
Fry, "Shane" Christopher	AJM-2
Gibson, Joseph	ANG-B1
Gill, Kimberly	ANG-B1
Ginsburg, Scott	ANG-B21
Guerrero, Summer	AJM-1310
Heagy, Win	MITRE
Hekl, Robert	MITRE
Henry, Javon	AJW-131
Howson, Mark	AJM-2113
Hritz, Mike	ANG-B2
Hunt, Rob	AJM-1
Jalleta, Ezra	MITRE
Kennedy, Christopher	AJM-21
Klein, Robert	ANG-B21

Name	Organization
Kubont, Derek	AJM-2531
Lewis, Michael	AJW-L4
Liggan, Michael	MITRE
Long, Phil	MITRE
Mackeen, Susan	AJM-21
Manoski, Stan	MITRE
Mayo, John	MITRE
McGeoch, David	AJM-24
Muhammad, Hawar	AJM-222
Muhammad, Jwan	AJM-21
Oscar Olmos	MITRE
Phifer, Joe	AJM-223
Pressler, Chris	AJM-31
Rush, Ted	AJW-B4
Sazon, Tony	AJM-21
Segers, Robert	ANG-B3
Shields, Rance	ADE-410
Smith, Barry	ANG-B21
Snyder, Steve	AJM-25
Soriano, James	AJM-13
Stratoti, Stephen	ANG-B12
Takata, Diana	ANG-B1
Thomson, Duncan	MITRE
Topiwala, Tejal	MITRE
Torrance, Kathy	AJM-25
Torrance, Mike	AJM-25
Tracy, Christopher	AJM-21
Wanke, Dr. Craig	MITRE
Winbush, James	ANG-B22
Young, Richelle	ADE-120
Young, Kevin	AJM-2100

Finally, the authors would like to thank Matt Warnock for his help in technical editing and preparing this document for delivery.

Executive Summary

Recognizing that automation systems within the National Airspace System (NAS) will eventually need to be replaced, the Federal Aviation Administration (FAA), with the support of The MITRE Corporation's Center for Advanced Aviation System Development (MITRE CAASD), created a vision for the evolution of these automation systems. An important part of the automation evolution vision is the transition to a layered, service-based architecture that can support modern software development and operations methodologies and take advantage of cloud computing technologies. This document describes that proposed future architecture and serves as a reference to align future evolution initiatives with Automation Evolution Strategy (AES). The document expands on the Automation Evolution work performed in fiscal year 2020 (FY20) by providing additional detail on the proposed future architecture, including a discussion of architecture layers, the service-based approach, monitoring and management, security, and support for multiple levels of criticality.

Strategic Outcomes

This Reference Architecture is one element of a larger effort intended to achieve the following desired strategic outcomes. These outcomes were articulated in FAA's Automation Evolution Strategy [1] and subsequently refined by senior FAA management. The outcomes are:

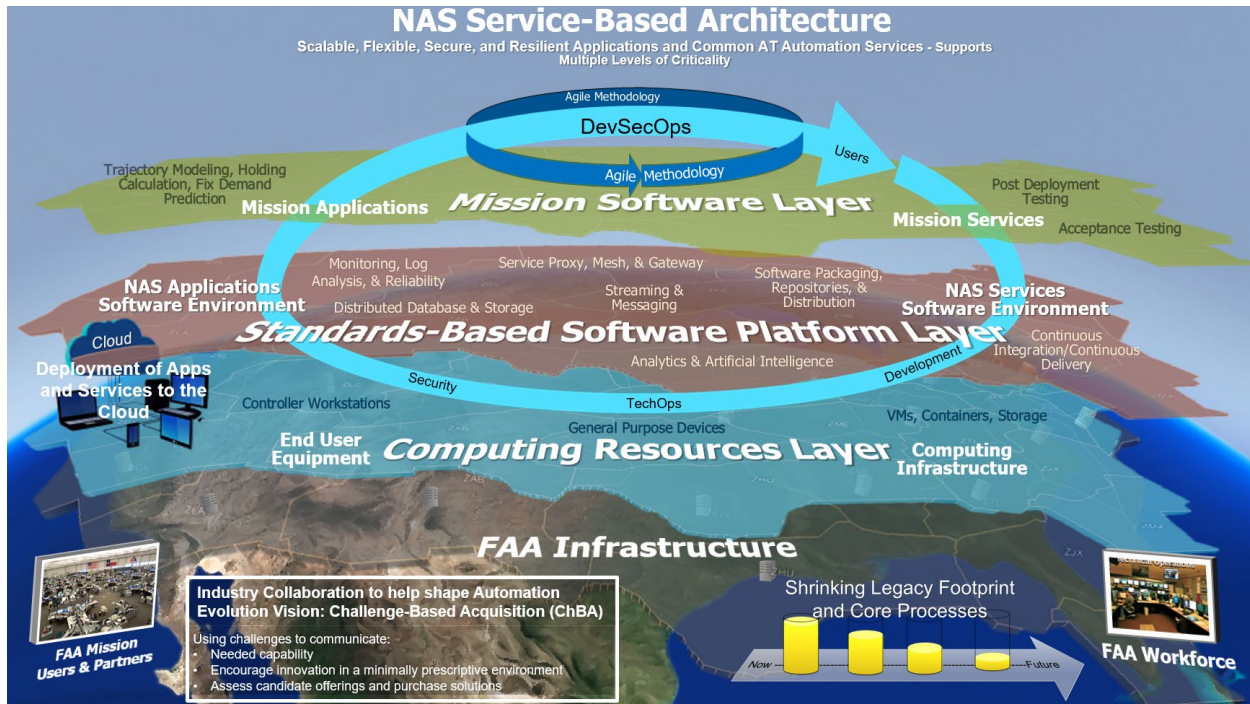
- Seek efficiencies for developing, operating, and sustaining NAS automation systems/services
- Reduce time to develop, integrate, and deploy new capabilities
- Leverage commercial industry best practices
- Establish broad industry base to support the FAA across a range of development and deployment capabilities
- Establish a scalable, flexible, secure, and resilient architecture

Purpose and Scope

This document describes the target technical state for the AES. It will serve as a reference that can be used to align future evolution initiatives with the overarching technical and operational objectives of the AES.

This document applies to automation that supports NAS operation. That includes core air traffic control (ATC) systems (i.e., En Route Automation Modernization [ERAM], Standard Terminal Automation Replacement System [STARS], etc.), flow management systems (i.e., Traffic Flow Management System [TFMS], Time-Based Flow Management [TBFM], etc.) and supporting functions (i.e., weather and aeronautical information management).

Architecture Vision



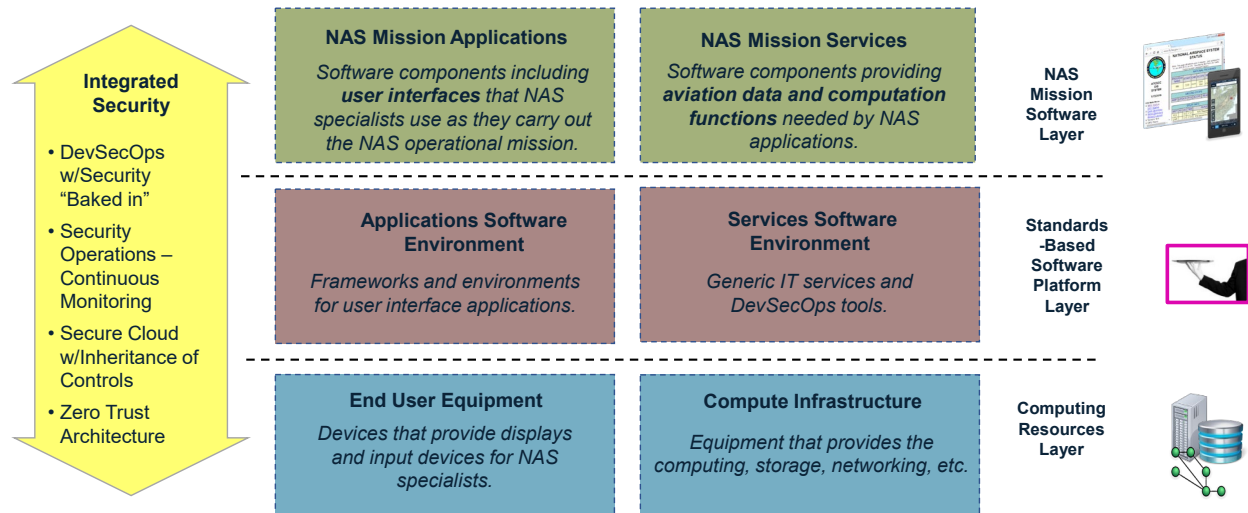
The above figure represents a high-level overview of the proposed future architecture framework that promotes the strategic outcomes listed above.

Guiding Principles

The Reference Architecture embodies a set of guiding principles that were stated in the FY20 Automation Evolution Strategy Briefing and white paper, with some additions from the Department of Defense (DoD) Enterprise Development, Security, and Operations (DevSecOps) Reference Design [2]. The principles are:

- **Enterprise Scope** – The automation strategy should support the implementation of services that meet the needs of multiple programs and associated Air Traffic Management (ATM) systems.
- **Incremental** – The automation strategy should promote the incremental development of services that can be implemented more rapidly and produce operational benefits in the near and long term.
- **Enable Opportunities and Expand Industry Base** – The architecture should promote expansion of the industry base to provide computing resources, platform software, and mission services.
- **Evolvable** – The architecture must expect that technologies and standards will change over time.
- **Streamlined Information Technology (IT)** – The architecture should automate as much of the development, testing, and deployment activities as possible.
- **Business Sense** – Apply technology and methodologies where it makes business sense to do so, and when requirements can be supported.

Architecture Key Characteristics



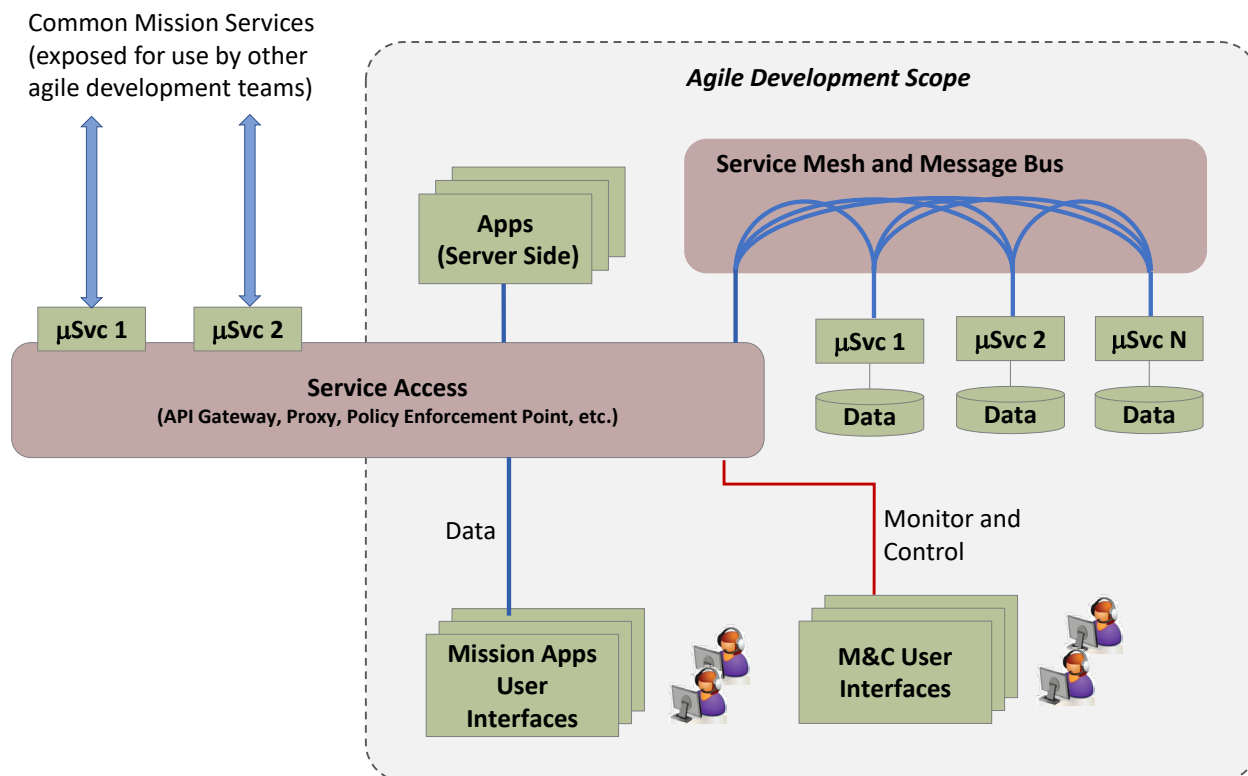
The Reference Architecture has the following key characteristics, based on the guiding principles, to achieve the strategic objectives:

- A layered architecture – the architecture is comprised of three layers including the NAS Mission Software Layer, the Standards-Based Software Platform Layer, and the Computing Resources Layer, as illustrated in the figure above
- A service-based architecture – small software components with loose coupling and minimized dependencies
- Operational monitoring and management tools to provide real-time visibility into status, performance, and availability of services, applications, and supporting layers, together with the ability to manage these elements
- Security
 - Built into DevSecOps tools and processes
 - Each layer inherits controls from layers below
 - Zero Trust principles
 - Cyber-defensive operations provide monitoring and response
- Multiple levels of criticality – Provides performance, reliability, and availability commensurate with safety-critical, efficiency-critical, essential, and routine service threads

Mission Layer

The NAS Mission Applications and Mission Services Layer (or Mission Layer, for brevity) contains applications and services that provide functionality to support the mission of the NAS. Software components in this layer are likely to be specifically developed for the NAS and are built to run in a standard environment provided by generic software components in the underlying Platform Layer.

The Mission layer contains mission services, which are software components that provide mission-specific data (i.e., flight data, surveillance data, and aeronautical data) and computation functions (i.e., tracking, weather prediction, and conflict probe) needed in the NAS. The Mission Layer also contains mission applications, which are user-facing software assemblies that provide the user interfaces needed by FAA specialists (e.g., controllers, traffic managers) to do their jobs.



In the Reference Architecture, each mission service implements some NAS business logic within a bounded context (i.e., each service does one specific well-defined thing). Mission services run independently and are accessed over the network via a well-defined interface.

Mission applications provide the functionality needed to support the operational mission of the NAS. In the Reference Architecture, these applications are created by combining a front-end component (e.g. a graphical user interface on the end user's workstation) with back-end mission services that provide access to data and computations (e.g. flight data processing services, aeronautical information services). Mission applications can be implemented as web applications, in which the application is downloaded from a web server and the user interface runs within a browser environment.

Mission services and applications are built following microservice design principles. These principles are important because they allow the Platform Layer to handle issues such as performance and availability using generic software. The Mission Layer also relies on Platform Layer components such as service meshes and message buses, Application Programming Interface (API) gateways, proxies, and policy enforcement points, to manage interactions among mission layer components.














The Reference Architecture allows variations and adjustments to basic microservice architecture when needed. One of these adjustments is the concept that some mission services are intended to be used within a limited scope, for example within the scope of effort of a single Agile

development team, whereas other mission services (referred to as common mission services) are intended to be used throughout the NAS.

Platform Layer

The Standards-Based Software Platform Layer (or Platform Layer, for brevity) consists of general-purpose IT software that provides the environment within which the Mission Software Layer components can run. The Platform Layer is expected to be assembled by licensing, configuring, and operating a suite of needed enterprise IT tools, including commercial-off-the-shelf (COTS), free and open source (FOSS) components, and Cloud Platform-as-a-Service (PaaS) offerings.

The Platform Layer includes software hosting/execution, monitoring and log analysis, development frameworks/libraries, and API/data management, and so on.

Runtime Platform Elements	Development Platform Elements
 Software Hosting/Execution	 API and Data Management
 Streaming and Messaging	 Planning and Requirements Management
 Service Proxy, Mesh, and Gateway	 Software Packaging, Repositories, and Distribution
 Virtual Networking, Policy, Authentication and Authorization	 Development Frameworks and Libraries
 Monitoring and Log Analysis	 CI/CD Toolchain
 Distributed Database and Storage	
 Workflow Choreography and Orchestration	
 Analytics and Artificial Intelligence	

The elements of the Platform Layer were informed by a survey of several programs that have been applying Agile and DevSecOps methodologies, and/or Service-Based or Microservices architectures. The survey results are summarized in Appendix B.

Computing Resources Layer

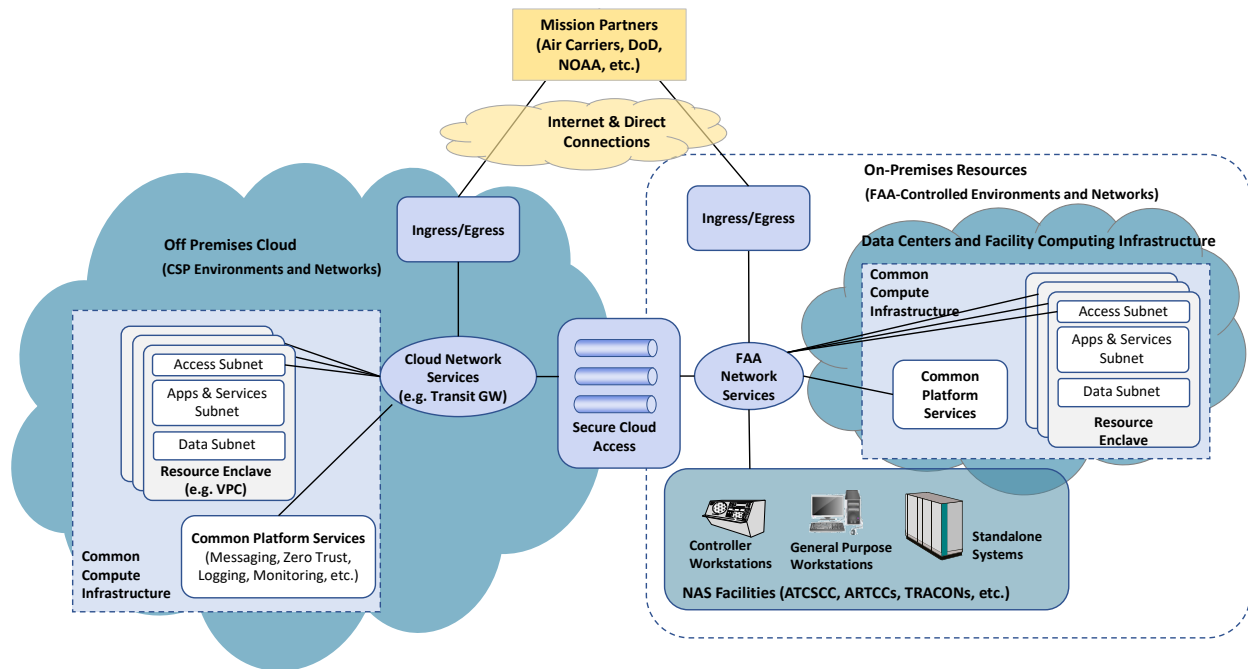
The Computing Resources Layer, depicted in the figure below, includes end-user equipment, compute infrastructure, and local area networks (LANs) and wide area networks (WANs) that tie them together. Examples of end-user equipment would be workstations, displays, tablets, mice, trackballs, and keyboards. Examples of compute infrastructure would be servers that run services, operating systems, storage, virtual machines (VMs), and containers.

The Computing Resources Layer has the following key characteristics:

- Secure: Meeting security standards allowing NAS Authorization to Operate (ATO)
- Available: Providing redundant processing in separate locations (e.g., Cloud Availability Zones)
- Reliable: Supporting continuity of operations (e.g., multiple cloud regions and multiple cloud providers)
- Responsive: Low latency and high data rate connectivity (e.g., direct connections to cloud environments from multiple NAS locations)

- Scalable: Able to expand or contract to continue to meet performance requirements as demand varies
- Supportable: Providing a support model that folds into FAA TechOps processes
- Trusted: Showing users and Operators the benefits of cloud and demonstrating cloud as a viable platform for the future

The elements that make up the Computing Resources Layer are a combination of on-premises as well as off-premises resources.



FAA Infrastructure Layer

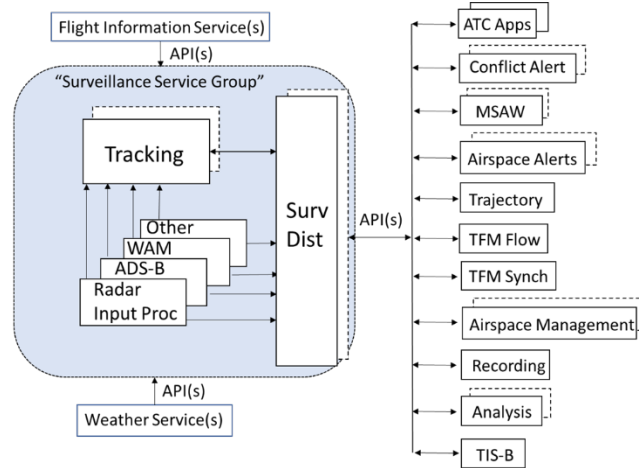
The FAA Infrastructure layer contains specialized equipment, for example instrument landing systems, and physical infrastructure, for example buildings and runways, that are included in the NAS, but do not fit clearly into the layers described above. Because of the specialized nature of the elements in the FAA Infrastructure Layer, this layer is not addressed in the Reference Architecture.

Surveillance Services Use Case

The body of the document concludes with a use case focusing on prospective Surveillance Mission Services. This use case informs perspectives regarding how mission services should be defined and how those mission services relate to other elements of the Reference Architecture. Surveillance was selected as a use case topic because it involves many properties of concern to determining the effective scope and applicability of the Reference Architecture (e.g., safety criticality, demanding performance, and aspects unique to ATC operations).

Service Definition Influences

- **Separation of functions/concerns**
- **Minimal coupling**
- **Criticality**
- **Legacy design experience**
- **Operational scope**
- **Scalability**
- **Anticipated change pace**
- **Software magnitude**
- **Resilience & disaster recovery**
- **Observability**



Principles applied to determine services involve a combination of general principles that apply to any service/microservice approach and domain-specific aspects that reflect the context of the operations to which the service applies. The influences applied to the definition of Surveillance Mission Services and the associated result are depicted in the figure above. Considering existing ATC automation designs as a guide, tentative surveillance services of radar input processing, Automatic Dependent Surveillance-Broadcast (ADS-B) input processing, tracking, and surveillance data distribution are identified.

Version Control

Version Number	Change by	Date	Description
0.1	MITRE	9/25/2020	Initial document structure created.
0.2	MITRE	11/19/2020	Draft with Sections 1 and 2 completed for review
0.3	MITRE	12/11/2020	Draft with comments on Sec 1 & 2 incorporated
0.4	MITRE	1/29/2021	Draft with Sections 4 and 6 completed for review
0.5	MITRE	2/19/2021	Draft incorporating “Orange Team” comments on Sections 4 and 6
0.6	MITRE	2/26/2021	Draft with section 5 for review
0.7	MITRE	3/31/2021	Draft of all sections complete (except Executive Summary)
1.0	MITRE	4/26/2021	Version 1.0 including Executive Summary, comments on Section 3, and other section comments delivered with section 3. Incorporated comments on V0.7, added Executive Summary, removed “Issues” section.
1.1	MITRE	8/11/2021	Final section 6 and other deferred comments have been addressed.
1.2	MITRE	8/12/2022	Removed material related to Concept of Use and minor editorial changes.

Table of Contents

1	Introduction	1-1
1.1	Background	1-1
1.2	Strategic Outcomes	1-1
1.3	Purpose.....	1-2
1.4	Scope.....	1-3
1.5	Approach.....	1-4
1.6	Related Infrastructure Programs/Projects	1-5
2	Architecture Overview	2-1
2.1	Architecture Vision.....	2-1
2.2	Key Concepts and Definitions	2-2
2.3	Guiding Principles	2-4
2.4	Assumptions.....	2-5
2.4.1	Agile and DevSecOps	2-5
2.4.2	Transition to Internet Protocol Networking	2-5
2.4.3	Use of Cloud for NAS Operations	2-6
2.4.4	Organizational Change.....	2-6
2.4.5	Acquisition	2-7
2.5	Architecture Key Characteristics	2-7
2.5.1	Layered.....	2-9
2.5.2	Service-based	2-11
2.5.3	Operational Monitoring and Management.....	2-12
2.5.4	Secure.....	2-13
2.5.4.1	Inheritance of Controls.....	2-13
2.5.4.2	Security Built-in to DevSecOps	2-14
2.5.4.3	Zero Trust	2-15
2.5.4.4	Cybersecurity Defensive Operations.....	2-17
2.5.5	Multiple Levels of Criticality.....	2-17
3	NAS Mission Applications and Mission Services Layer	3-1
3.1	Overview.....	3-1
3.2	Mission Applications	3-1
3.3	NAS Mission Services	3-2
3.4	Basic Pattern: Microservice Architecture	3-3
3.4.1	Microservices Concept.....	3-3

3.4.2	Assembling Microservices to Create Applications	3-4
3.5	Service-Based Architecture	3-5
3.5.1	Internal Mission Services Versus Common Mission Services	3-5
3.5.2	Geospatial Dependencies	3-6
3.5.3	Support for Transactions and Data-Centric Subsystems.....	3-7
3.5.4	NAS-Wide View of Service-Based Architecture.....	3-9
3.5.5	Monitoring and Control	3-10
3.5.6	Security Monitoring and Defensive Operations.....	3-10
4	Platform Layer.....	4-1
4.1	Overview	4-1
4.2	Platform Layer Elements	4-2
4.2.1	Runtime.....	4-2
4.2.1.1	Software Hosting/Execution.....	4-2
4.2.1.2	Workflow Choreography and Orchestration	4-3
4.2.1.3	Monitoring, Log Analysis, and Reliability.....	4-3
4.2.1.4	Service Proxy, Mesh, and API Gateway	4-4
4.2.1.5	Virtual Networking, Policy, Authentication, and Authorization.....	4-5
4.2.1.6	Distributed Database and Storage	4-5
4.2.1.7	Streaming and Messaging (Message Bus).....	4-6
4.2.1.8	Analytics and Artificial Intelligence	4-7
4.2.2	Development	4-7
4.2.2.1	Development Frameworks and Libraries	4-8
4.2.2.2	Planning and Requirements Management.....	4-8
4.2.2.3	Software Packaging, Repositories, and Distribution.....	4-9
4.2.2.4	Application Programming Interface and Data Management.....	4-9
4.2.2.5	Continuous Integration / Continuous Delivery Toolchain	4-10
4.2.3	Summary of Platform Elements in the Reference Architecture.....	4-11
4.3	Survey of Existing Platforms	4-12
5	Computing Resources Layer	5-1
5.1	Overview	5-1
5.2	Common Compute Infrastructure	5-3
5.2.1	Off-Premises Cloud.....	5-4
5.2.2	Data Centers and Facility-Based Computing Infrastructure	5-5
5.3	Standalone Systems	5-6
5.4	End-User Equipment.....	5-6

5.4.1	Safety-Critical User Interfaces	5-6
5.4.2	Web Browser-Based User Interfaces	5-7
5.5	Networking	5-7
6	Surveillance Services Use Case.....	6-1
6.1	Background	6-1
6.1.1	Surveillance Use Case Purpose.....	6-1
6.1.2	Current Surveillance Data Processing.....	6-1
6.2	Surveillance Services Overview	6-2
6.2.1	Surveillance Use Case Scope and Assumptions	6-2
6.2.2	Surveillance Services Context	6-3
6.3	Surveillance Services Detail	6-4
6.3.1	Surveillance Services Mission Layer	6-4
6.3.1.1	Surveillance Service Platform Layer.....	6-13
6.3.1.2	Surveillance Services Infrastructure Layer	6-15
6.3.2	Surveillance Services Challenges	6-17
Appendix A	Related Infrastructure Programs/Projects.....	A-1
A.1	FAA Telecommunications Infrastructure (FTI) and FAA Enterprise Network Service (FENS)	A-1
A.2	FAA Cloud Services (FCS)	A-1
A.3	Integrated Enterprise Services Platform (IESP)	A-1
A.4	National Cloud Integration Services (NCIS)	A-1
A.5	System Wide Information Management (SWIM).....	A-2
A.6	SWIM Cloud Distribution Service (SCDS).....	A-2
A.7	Enhanced SWIM Cloud Service	A-2
A.8	Enterprise Information Management (EIM) Platform	A-3
Appendix B	Platform Survey	B-1
B.1	Application Based Capability Development (ABCD).....	B-1
B.2	Configuration, Logistics, and Maintenance Resource Solutions	B-4
B.3	Enterprise Information Management (EIM) Data Platform (DP).....	B-5
B.4	Elroy.....	B-8
B.5	Platform One.....	B-10
Appendix C	References.....	C-1
Appendix D	Acronyms.....	D-1

List of Figures

Figure 1-1. Intended Use of Reference Architecture	1-3
Figure 2-1. Architecture Vision	2-1
Figure 2-2. Layered Architecture – Overview	2-9
Figure 2-3. High Level Concepts for Operational Monitoring and Management	2-12
Figure 2-4. Security Overview.....	2-13
Figure 2-5. Inheritance of Controls.....	2-14
Figure 2-6. Security in DevSecOps	2-15
Figure 2-7. Zero Trust Concepts in the Reference Architecture.....	2-16
Figure 2-8. Layered Architecture Supporting Multiple Levels of Criticality	2-18
Figure 3-1. Mission Layer.....	3-1
Figure 3-2. Microservice Concept	3-3
Figure 3-3. Application Assembled from User Interface and Mission Services	3-5
Figure 3-4. Microservices Exposed as Common Mission Services.....	3-6
Figure 3-5. Use of Shared Data.....	3-8
Figure 3-6. NAS-Wide View of Services and Applications	3-9
Figure 4-1. Standards-based Software Platform Layer.....	4-1
Figure 5-1. Computing Resources Layer	5-1
Figure 5-2. Computing Resources Elements	5-2
Figure 6-1. Layered Architecture.....	6-4
Figure 6-2. NAS Surveillance Services	6-10
Figure 6-3. Surveillance Services Infrastructure	6-16
 Figure B-1. ABCD Operational Architecture	 B-2
Figure B-2. ABCD Development Architecture	B-3
Figure B-3. EIM Platform Runtime Architecture	B-6
Figure B-4. EIM Platform Development Architecture	B-6
Figure B-5. Project Elroy System Overview	B-9
Figure B-6. Project Elroy Platform Overview	B-9
Figure B-7. Overview of DoD Enterprise DevSecOps Layers	B-11

List of Tables

Table 2-1. Key Characteristics of Reference Architecture	2-8
Table 4-1. Batch Versus Stream Processing	4-7
Table 4-2. Platform Elements in the Reference Architecture	4-11
Table 6-1. Surveillance Services Rationale	6-7
Table B-1. Platform Layer Elements in ABCD	B-3
Table B-2. Platform Layer in CLMRS	B-4
Table B-3. Platform Elements in EIM Platform	B-7
Table B-4. Elroy Platform.....	B-10
Table B-5. Platform Layer Elements in DoD Platform One.....	B-12

1 Introduction

The National Airspace System (NAS) comprises a set of systems that provide the Federal Aviation Administration (FAA) workforce with the tools and capabilities needed to perform the agency's operational mission. The NAS systems include the major automation systems that support air traffic control and flow management operations. Those automation systems are complex and were built over many years at considerable expense. Recognizing that those systems will eventually need to be replaced, the FAA, with the support of The MITRE Corporation's Center for Advanced Aviation System Development (MITRE CAASD), created a vision for the evolution of these automation systems, with the dual goals of reducing cost of ownership and increasing the speed for delivering new operational functions. An important part of the automation evolution vision is the transition to a layered, service-based architecture that can support Agile and Development, Security, and Operations (DevSecOps)¹ methodologies and take advantage of cloud computing technologies. This document describes that proposed architecture.

1.1 Background

Early in fiscal year 2020 (FY20), it became apparent that a consensus was emerging in the FAA and other parts of government to employ modern software practices to acquire and manage systems more efficiently. As an example, the United States Air Force (USAF) Kessel Run effort demonstrated how a government agency can implement capabilities more quickly and efficiently using modern Agile processes. The FAA and MITRE CAASD are working together to develop a plan to modernize their NAS automation architecture and transition strategy to enable those new Agile processes.

As part of this effort, MITRE CAASD collaborated with the FAA in FY20 to develop a consensus evolution strategy and path forward [3], which included the following:

- Characteristics and principles of an automation evolution strategy
- Strategic outcomes
- Work plan
- Socialization strategy
- Next steps

This document expands on the Automation Evolution work performed in FY20 by providing additional detail on the proposed future architecture, including a discussion of architecture layers, service-based approach, and security. Other concurrent tasks include developing other aspects of the strategy including transition planning, acquisition, and organizational changes.

1.2 Strategic Outcomes

This Reference Architecture document is one element of a larger effort intended to achieve the following desired Strategic Outcomes. These outcomes were articulated in FAA's Automation

¹ DevSecOps refers to a methodology that integrates development, security, and operations concerns.

Evolution Strategy [1] and subsequently refined by senior FAA management. The desired outcomes are:

- Seek efficiencies for developing, operating, and sustaining NAS automation systems/services
- Reduce time to develop, integrate, and deploy new capabilities
 - Move to incremental investments to focus on immediate needs that will enable faster deployment of user priorities
- Leverage commercial industry best practices
 - Agile/DevSecOps processes and tools
- Establish broad industry base to support the FAA across a range of development and deployment capabilities
- Establish a scalable, flexible, secure, and resilient architecture
 - Continue meeting safety, security, performance, monitoring, and maintenance requirements

1.3 Purpose

This document describes the target technical state for the Automation Evolution Strategy. It will serve as a reference that can be used to align evolution initiatives with the overarching technical and operational objectives of the Automation Evolution Strategy. The Reference Architecture focuses on exposing and maximizing the benefits of containerized, reusable software services and technology through layered components. It will serve as an important tool to inform the NAS Enterprise Architecture (EA), which describes the NAS evolution through roadmaps and models.

The purpose of this Reference Architecture is to:

- Identify and describe the major components or areas of the future NAS automation
- Describe the relationships among these components and how they interact

To facilitate:

- Communication among stakeholders based on a common understanding of what the major areas are
- Allocation of responsibility for different areas (design, implementation, sustainment, etc.)
- Guidance that can be applied to more detailed architecture description or design activities, resulting in a coherent and cohesive approach
- Governance that constrains possible designs of NAS applications and infrastructure to achieve desired properties in the future NAS architecture

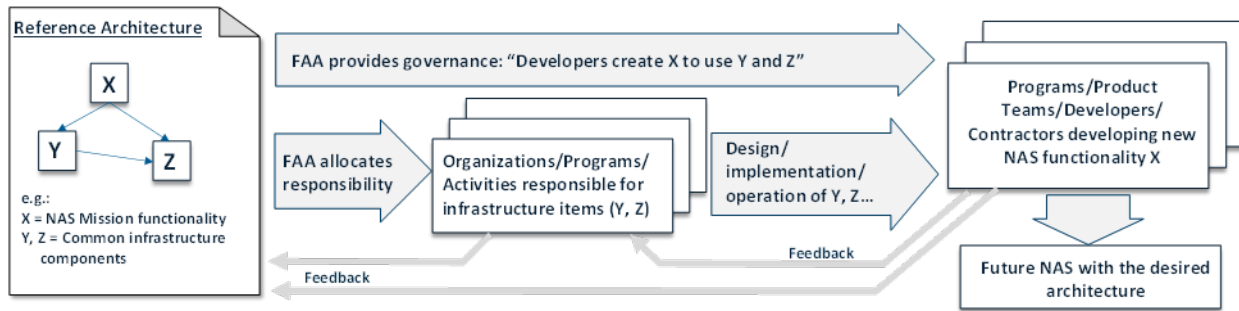


Figure 1-1. Intended Use of Reference Architecture

The intended use of this document is illustrated in Figure 1-1. The Reference Architecture will describe, in general terms, the future NAS components and how they are interrelated. Those are shown as X, Y, and Z in the figure. As a simple example, X might represent a software component that provides NAS mission functionality, such as a trajectory modeling service. Y and Z might refer to infrastructure components. For instance, Y might be generic software tools and Z might be cloud computing environments. The FAA would then need to allocate responsibility to program offices to develop the mission software (X) and to other organizational units to acquire, configure, and sustain the infrastructure components (Y, and Z). In that example, one infrastructure organization would be given responsibility for creating secure NAS cloud computing environments (Y), and another (or the same) infrastructure organization would select the common software tools (Z) and configure them in those cloud environments. The program office developing the trajectory modeling service (X) would then be given access to the cloud environments and would develop and deploy their software in these environments using the provided tools. The net result would be a future NAS that has the desired architecture characteristics.

It is important to note that Figure 1-1 also includes feedback loops. The Reference Architecture will evolve as our understanding improves, technology advances, and the operational environment changes. Teams, organizations, and activities need to work more collaboratively, and those responsible for infrastructure components should be expected to propose changes to the Reference Architecture as new technologies and industry practices become available. Similarly, program offices developing new mission software may be expected to request specific features or changes to the infrastructure to meet their needs, or even to request more fundamental changes to the overall architecture when needed.

1.4 Scope

This document applies to automation that supports NAS operation. That includes both Flow Management as well as air traffic control (ATC) functions and supporting functions (for example, weather and aeronautical information management).

While not the intended focus of this architecture, other FAA environments such as those used for training, off-line performance analysis, historical data analysis, research and development, and concept exploration, may also benefit by following a consistent architectural approach. The other environments may have different needs, which may suggest variation from the NAS architecture. For example, the NAS will be generally real-time and event-driven, whereas historical data analysis is more likely to require non-real-time capabilities for storing and analyzing very large data sets.

Supporting ATC presents special challenges due to the need to support safety-critical service threads that require real-time response times, high availability, and high levels of assurance that software will function correctly. For those reasons, ATC systems are difficult to change and may not be the first candidates to transition to the future architecture. Nevertheless, they are in scope, because achieving the strategic outcomes depends on reducing the costs of sustaining those systems and evolving them to support new operational concepts and adapting to a changing environment. Therefore, this Reference Architecture must be suitable for being applied when developing automation to support ATC, including safety-critical service threads.

As we will discuss, support for modern software development methodologies is key to obtaining the strategic outcomes. Therefore, this Reference Architecture applies to development environments (including test and operational suitability assessment) as well as operational environments.

The sustainment of legacy NAS ATC systems is one of the major cost drivers the FAA needs to consider as part of the Automation Evolution strategy. As a result, the FAA will conduct additional work activities to look at transition planning and assess acquisition implications, which are outside of the scope of this document.

This document describes the Reference Architecture associated with the FAA's Automation Evolution Strategy and provides a summary of the needed design components and processes to provide a repeatable architecture design that can be used to instantiate this across the NAS. Transition efforts to evolve toward this Reference Architecture are still being defined and as the transition approach matures, more detailed execution guidance will be provided for use by NAS developer, operator, and other stakeholder communities.

1.5 Approach

This approach taken in the Reference Architecture to achieving the strategic outcomes aligns with the FAA and NAS EA best practices for how the agency should develop its mission, information, and technology architectures. Therefore, the FAA can use it consistently at various levels as well as with external stakeholders. The common approach provides integration points with other agency functions including strategic planning, capital planning, program and portfolio management, cybersecurity, and workforce development.

The team's approach to creating this Reference Architecture began with the goals, principles, and desired strategic outcomes that have been formulated by the Automation Evolution work performed so far. Additionally, the team considered the high-level architectural concepts (layered, service-based architecture) documented in previous papers and briefings that have been reviewed and accepted by the FAA Automation Evolution Strategy team [1], [3], [4].

The team expanded and refined the high-level Automation Evolution concepts to create a Reference Architecture suitable for the purposes described in Section 1.3. To accomplish that, the team drew upon:

- Knowledge of the NAS
 - Subject matter expertise on existing and planned NAS operations and systems
 - Plans and future vision for the NAS [5]
 - NAS *As-Is* and *To-Be* Enterprise Architecture

- Examples from other government agencies, in particular, the Department of Defense (DoD) Enterprise DevSecOps Reference Design [6]
- Existing *Platforms* or *Platform-like* capabilities
 - FAA Cloud Services
 - System Wide Information Management (SWIM)
 - Enterprise Information Management (EIM) Platform
 - Project Elroy/Pivotal Platform
 - Configuration, Logistics, and Maintenance Resource Solutions (CLMRS) project
 - Application-Based Capability Development (ABCD) project experience [7]
 - DoD Platform One
- Zero Trust Architecture (ZTA) principles [8]

The Reference Architecture description is a living document that is being developed iteratively. The iterative process includes identifying key questions, which may influence pathfinder activities, which may in turn, be used to refine the architecture.

1.6 Related Infrastructure Programs/Projects

There are several existing programs and projects that are already implementing aspects of the infrastructure needed by the Reference Architecture, including:

- FAA Telecommunications Infrastructure (FTI), which provides wide area networking services for the FAA.
- FAA Enterprise Network Services (FENS), which will subsume FTI.
- FAA Cloud Services (FCS), which provides FAA programs with access to commercially provided cloud services.
- National Cloud Integration Services (NCIS), which is working with FCS to establish cloud environments for NAS programs. NCIS leverages FCS for obtaining cloud services.
- SWIM, which provides standards and infrastructure for information dissemination, using publish-subscribe and request-response information exchanges. SWIM information exchange services leverage the FTI/FENS network as well as FCS and NCIS cloud services.
- EIM Platform, which provides a data archiving and analytics environment and platform. EIM leverages SWIM for accessing information from the NAS, and FCS for cloud services.
- The Integrated Enterprise Services Platform (IESP) is a virtualization platform that operates in the NAS. It can provide virtual machines (VMs) for NAS programs. These VMs come with monitoring, FTI connectivity, authentication, and backup services.

Those programs are described in Appendix A.

2 Architecture Overview

This section provides a high-level overview of the architecture, defines terms and concepts, lists assumptions, and describes key characteristics of the architecture intended to lead to the strategic outcomes stated in Section 1.3.

2.1 Architecture Vision

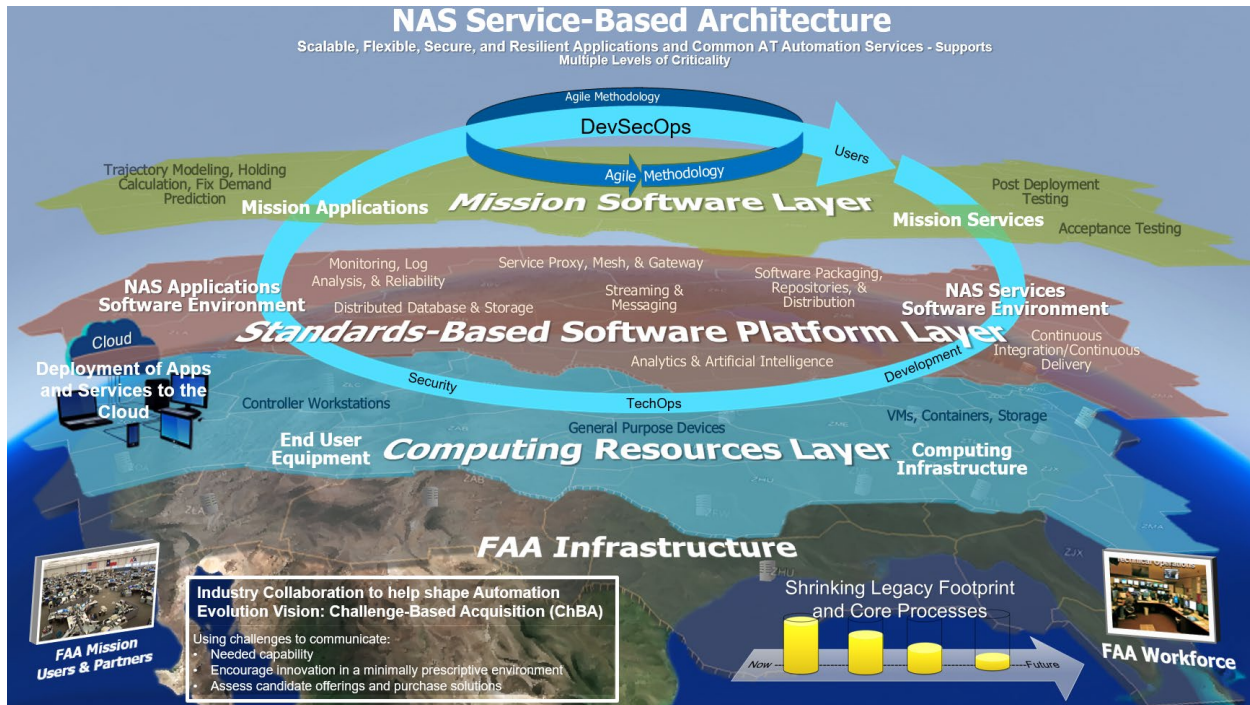


Figure 2-1. Architecture Vision

Figure 2-1 represents a high-level overview of the proposed future architecture that promotes the goals of reducing cost and increasing the speed with which the FAA can make new functionality operational. The architecture promotes those goals by breaking the NAS into loosely coupled components that can be developed and sustained independently, thereby increasing opportunities for safe incremental improvement, reuse, competition, and parallel development.

The figure depicts four architecture layers. The Mission Software Layer consists of software that is specific to the mission of the NAS, for example trajectory modeling software. The Standards-Based Software Platform Layer contains generic Information Technology (IT) software, for example distributed database software. The Computing Resources Layer contains the computing, storage, and networking components, for example workstations and server hardware. The FAA Infrastructure layer contains specialized equipment, for example instrument landing systems, and physical infrastructure, for example buildings and runways, that are included in the NAS, but do not fit clearly into the layers described above. (Because of the specialized nature of the elements in the FAA Infrastructure Layer, this layer is not addressed in the Reference Architecture.)

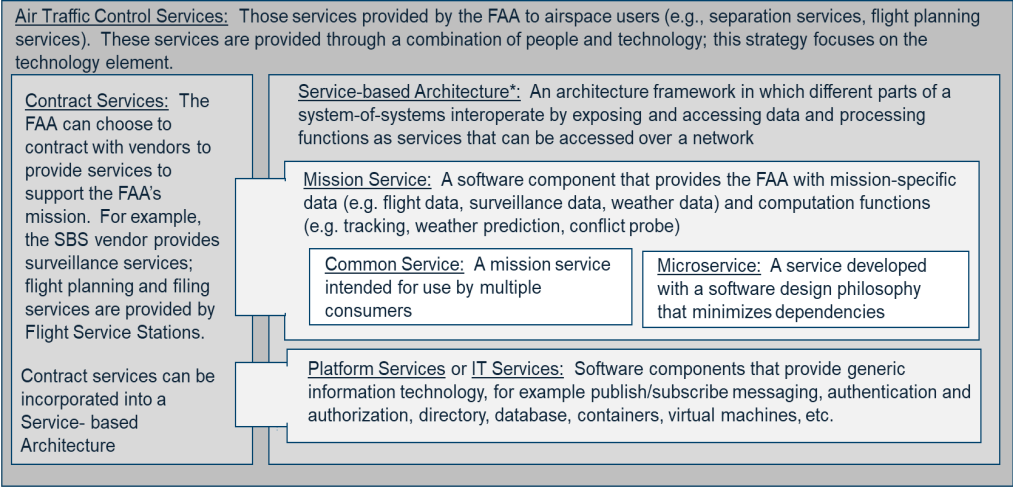
Key activities associated with each layer are shown on the figure. Agile and DevSecOps processes, tools, and methods are key aspects of the proposed architecture. The following sections in this paper present more detail on the proposed architecture.

2.2 Key Concepts and Definitions

This subsection contains key terms and definitions used in this document.

Term	Definition
Agile	An Agile approach to software development emphasizes user involvement, incremental deployment, and rapid feedback to deliver high value to users in a timely manner at a lower cost.
Cloud	Cloud is often used in the specific sense of a set of computing resources managed by an organization other than the users of those resources, and which are available for the deployment of software applications and services and the storage and access of information. Cloud resources are usually shared with other users. Resources in the cloud can be procured or released dynamically. Different cloud service models include Infrastructure as a Service (IaaS) which provides basic resources such as virtual machines (VMs) and block storage, Platform as a Service (PaaS) which provides services that replace generic information technology (IT) software building blocks such as databases and web servers, and Software as a Service (SaaS) which provides complete cloud-based software packages such as e-mail.
Container	A container packages software and all dependencies except the operating system into an easily deployed unit that will run reliably across computing environments. Because a container does not include the operating system, multiple containers can be run on the same server, and they are typically much faster to instantiate than a VM with the necessary software. Nonetheless, there may be applications, particularly during transition, where the Reference Architecture may implement VMs, which can be part of a unified PaaS platform.
Container Orchestration	Container Orchestration automates the deployment, management, and scaling of containers. A container orchestrator will typically provide a way of automatically deploying a scheduled number of a container image within a cluster of servers, will monitor the health of those containers, will restart a container image as needed, and provides for a common service Application Program Interface (API) endpoint for services running within the managed cluster. In the case of Kubernetes, a very popular container orchestrator, one or more containers are grouped together in a pod, and the pod is what is deployed.
DevSecOps	Development, Security, and Operations (DevSecOps) is an Agile methodology that refers to development, security, and operations. “Dev[Sec]Ops is a set of practices that automates the processes between software development and information technology (IT) teams, in order that they can build, test, and release software faster and more reliably.” [9] DevSecOps includes security as a critical component of the Development Operations (DevOps) practices.

Term	Definition
IaC	Infrastructure as Code (IaC) is a DevSecOps approach in which a development team includes in the source code baseline all the commands or declarations necessary to configure servers, frameworks, software libraries, and so on (anything that the mission software depends on). DevSecOps toolchains are then used to automate the deployment of the mission software, together with all the necessary infrastructure configuration and installation of software dependencies, into the development, test, staging, and production environments.
NAS Mission Software	NAS Mission Software includes NAS Mission Applications and NAS Mission Services. Mission applications are the user-facing software components that provide functions needed by FAA specialists to do their jobs. Mission services are software components that provide the data and computation services needed by the applications, via well-defined APIs. (Not to be confused with the term “Mission Support”, which is a term used to distinguish supporting functions from functions that are directly in real-time air traffic management operations.)
Pathfinder	An initial service-based capability used to gather lessons learned from the application of Agile development and acquisition processes (e.g., DevSecOps and cloud deployment). A pathfinder would help address the following: <ul style="list-style-type: none"> • FAA Agile system engineering/acquisition processes • Roles and responsibilities for government, research, and industry • Acquisition contract mechanisms • How multiple vendors can work within the framework
Program	An organizational activity with responsibility for creating mission applications and/or mission services to provide value to the FAA and aviation users. For larger efforts, a formal Program Office may be created; smaller efforts may be managed by a less formal project structure.

Term	Definition
Service	<p>The term <i>Service</i> is used in many ways. Generally, when we use the term in this document, we are referring to a software component that implements some set of related functionalities, is accessed through a well-defined interface, and is designed so it can be used by multiple clients or other services. The following illustration shows some of the other ways this term is used.</p>  <p>Air Traffic Control Services: Those services provided by the FAA to airspace users (e.g., separation services, flight planning services). These services are provided through a combination of people and technology; this strategy focuses on the technology element.</p> <p>Contract Services: The FAA can choose to contract with vendors to provide services to support the FAA's mission. For example, the SBS vendor provides surveillance services; flight planning and filing services are provided by Flight Service Stations.</p> <p>Service-based Architecture*: An architecture framework in which different parts of a system-of-systems interoperate by exposing and accessing data and processing functions as services that can be accessed over a network</p> <p>Mission Service: A software component that provides the FAA with mission-specific data (e.g. flight data, surveillance data, weather data) and computation functions (e.g. tracking, weather prediction, conflict probe)</p> <p>Common Service: A mission service intended for use by multiple consumers</p> <p>Microservice: A service developed with a software design philosophy that minimizes dependencies</p> <p>Platform Services or IT Services: Software components that provide generic information technology, for example publish/subscribe messaging, authentication and authorization, directory, database, containers, virtual machines, etc.</p> <p>Contract services can be incorporated into a Service-based Architecture</p> <p><small>*The concept is the same as <u>Service Oriented Architecture</u> (SOA), but that terminology became closely associated with specific vendor products, especially Enterprise Service Bus (ESB) products, which today are considered somewhat dated</small></p>
Software Factory	A software factory is a set of integrated software assets used to create, test, release and deploy software applications and components in a structured and repeatable way.
Zero Trust	Zero Trust Architecture is an approach to information security that improves upon perimeter-based security by requiring all resource requests to be authenticated and authorized on a per session basis regardless of their position with respect to enterprise infrastructure. As a result, breaches are more difficult to propagate, defensive operations are more effective, and granting access can be more flexible.

2.3 Guiding Principles

The Reference Architecture embodies a set of guiding principles that were stated in the FY20 Automation Evolution Strategy Briefing and white paper, with some additions from the DoD Enterprise DevSecOps Reference Design [2]. The principles are:

Enterprise Scope – The automation strategy should support the implementation of services that meet the needs of multiple programs/systems. It will adopt common tools from planning and requirements through deployment and operations. It will apply architecture at enterprise scale to achieve desired strategic outcomes.

Incremental – The automation strategy should promote the incremental development of services that can be implemented more rapidly and produce operational benefits in the near and long term. By focusing on more immediate needs and priorities, incremental development avoids unnecessary work and rework resulting from speculation about potential future operations. A *big bang* change will not be possible – we expect to

continue to operate and sustain legacy systems while the new proposed service-based architecture is instantiated.

Enable Opportunities and Expand Industry Base – The architecture should promote expansion of the industry base to provide computing resources, platform software, and mission services. When appropriate, resources and services may be outsourced to vendors offering suitable service level agreements.

Evolvable – The architecture must expect that technologies and standards will change over time. To avoid building up *technical debt*, and to increase the pace of introduction of new capabilities, the architecture must be evolvable.

Streamlined Information Technology (IT) – The architecture should automate as much of the development, testing, and deployment activities as possible. Remove bottlenecks and manual actions.

Business Sense – Apply technology and methodologies where it makes business sense to do so and when requirements can be supported.

2.4 Assumptions

2.4.1 Agile and DevSecOps

We assume that Agile and DevSecOps will be the preferred development methodologies. The shift to Agile software development is in direct contrast to the traditional “Waterfall” method of project management. Rather than focus on a fixed schedule and inflexible requirements and deliverables, Agile focuses on the following:

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

DevOps is a set of practices integrating various activities within software development and IT operations. Ideally, it can shorten systems development life cycles and provide advantages like Continuous Delivery (CD). These mechanisms pave the way for consistent and reliable continuous development, integration, and delivery as well as facilitating automated verification and testing.

DevOps is complementary with the Agile software development process; several DevOps aspects came from Agile methodology. By extending the concept of automation to include security verification into this DevOps process, we arrive at the concept of DevSecOps which is the philosophy of integrating security practices within the DevOps process.

Rationale: These methodologies are being applied and proving successful in more and more domains and have proven to dramatically reduce cost and time-to-market.

2.4.2 Transition to Internet Protocol Networking

We assume that the transition to the Internet Protocol (IP) suite from Time Division Multiplexing (TDM) circuits will continue. Currently, some TDM circuits remain in use in the NAS, including

connections from surveillance sensors to ATC systems, interfacility data transfer (IDAT) interfaces among ATC systems, and voice communications.

This includes the assumption that the IP networks will be capable of meeting availability and performance requirements of ATC services.

Rationale: The transition to IP is well underway and will only accelerate as TDM circuits become more expensive and less supported. We expect all new capabilities will be built to work with IP, and legacy information exchanges will continue to transition to using IP network services.

The Reference Architecture is network-centric in that it relies heavily on resources that are accessed over the network. There may be special cases in which all critical computing is kept local to avoid dependence on the network, but those should be minimized to obtain the benefits of the architecture. Given advances in commercially available networking services, and the requirements that have been provided to industry in the FENS solicitation, we believe it is reasonable to assume that NAS performance and availability requirements can be met at reasonable cost using commercially available network services.

The practical realization of this architecture will rely on the application of tools and methods that work in an IP networking environment. Therefore, a service-based architecture will not apply to portions of the NAS that do not use IP, although to the extent that those portions still exist, they can be accommodated by gateways and legacy system interfaces.

2.4.3 Use of Cloud for NAS Operations

We assume the FAA will be able to address concerns regarding cloud technologies in areas such as availability, security, and performance so they will be approved to use in NAS operations. Acceptable implementation may include a combination of commercially provided off-premises cloud, edge-computing, and on-premises cloud.

Rationale: Cloud technologies offer scalability and the opportunity for reduced infrastructure costs and make possible an *Infrastructure as Code* approach that is a key part of DevSecOps, and this architecture. The FAA is already making use of cloud services for administrative and mission-support functions and is planning initial steps to use cloud to support NAS operational functions.

2.4.4 Organizational Change

We assume the FAA will make changes across organizational roles, responsibilities, and activities (e.g., minimize software and systems engineering silos [10]), governance, and workforce skills based on a service-based architecture and associated methodologies (e.g., Agile and DevSecOps).

Rationale: Agile and DevSecOps rely heavily on cultural change including increased collaboration, continuous feedback, team autonomy, and increased reliance on automation. While these organizational and cultural changes may be difficult, they are necessary for Agile and DevSecOps to succeed. The FAA will also need the right organizational structures to be responsible for platform standards and implementation, as well as integration, sustainment, and operational management of common services.

2.4.5 Acquisition

We assume that acquisition processes and contracting artifacts will be tailored to allow Agile and DevSecOps methodologies to succeed. Traditional acquisition practices are aligned with a traditional “big design upfront” development philosophy that involves detailed requirements, plans, and designs as precursors to solution development. The development usually proceeds within contractor-provided development and test environments. Alternatively, Agile promotes high level requirements, plans, and architecture as precursors for an emergent design process and incremental development. A DevSecOps environment may be provided as Government Furnished Equipment (GFE). The expectation of an Agile/DevSecOps development effort should be facilitated by suitable contract content that includes objectives or high-level requirements instead of detailed requirements, frequent incremental software deliveries, metrics and progress reporting aligned with incremental development, streamlined documentation, and other differences with more traditional contract content.

Rationale: This is necessary for effective Agile and DevSecOps.

Architecture Key Characteristics

Table 2-1 contains the key characteristics of the Reference Architecture, with a brief explanation of why each characteristic is important and how it helps achieve the desired strategic outcomes.

Table 2-1. Key Characteristics of Reference Architecture

WHAT	WHY
<ul style="list-style-type: none"> ▪ Layered <ul style="list-style-type: none"> – Mission Applications and Mission Services: NAS-specific software <ul style="list-style-type: none"> • Created by Agile/DevSecOps development teams – Platform: Generic Information Technology (IT) software <ul style="list-style-type: none"> • Provided, operated, and sustained separately and used by multiple Mission Layer software development teams • Provides software factory tools for developers • Instantiates instances of mission services as needed • Mediates invocations of services and controls access – Computing Resources: End user devices, computing, storage, and networking <ul style="list-style-type: none"> • Provided, operated, and sustained separately and used by multiple development teams • Leverage cloud technologies and commodity hardware • On-demand dynamic provisioning provides resources as needed 	<ul style="list-style-type: none"> – Separates architecture elements, allowing teams to focus on what they do best: Agile/DevSecOps teams create application software to meet mission needs, platform engineers assemble off-the-shelf IT tools, and service providers offer commodity computing resources – Common platform and computing layers reduce rework, improve security, and simplify licensing and training – Allows Agile product teams to be nimble in responding to new concepts and changing environment <ul style="list-style-type: none"> • Focus on mission needs, without having to reinvent lower layers • Software factory (DevSecOps pipeline) provides repeatable processes resulting in high quality secure products – Lower layers can evolve independently as technology evolves, without impacting the Mission Layer software <ul style="list-style-type: none"> • “Tech refresh” is handled by the service provider, and costs are factored into recurring service costs, rather than requiring capital improvement funds – Platform Layer provides scalability and resilience by instantiating mission service instances as needed, leveraging elastic scaling provided by the computing resources layer
<ul style="list-style-type: none"> ▪ Service-based <ul style="list-style-type: none"> – Small software components with loose coupling and minimized dependencies. – Information exchange and interoperability via Message Bus, Service Mesh, Application Programming Interface (API) Gateways <ul style="list-style-type: none"> • Minimize databases shared across different application environments – Interoperability and integration via data standards and interface/API configuration management – Service instances run wherever needed for performance and availability, and are accessed via the network under control of Platform Layer services 	<ul style="list-style-type: none"> – Breaking large systems into smaller loosely coupled components: <ul style="list-style-type: none"> • Reduces cost and time-to-market by allowing parallel development by multiple smaller development teams • Allows increased opportunities for competition – Data standards, API configuration management, and registries/repositories mitigate the integration complexity of maintaining interoperability among mission applications and services evolving in parallel – Location independence facilitates resiliency and the ability to support contingency operations
<ul style="list-style-type: none"> ▪ Operational Monitoring and Management <ul style="list-style-type: none"> – Tools provide real-time visibility into status, performance, and availability of services, 	<ul style="list-style-type: none"> – “Technical Operations is responsible for the performance of equipment, systems, and

WHAT	WHY
applications, and supporting layers, together with the ability to manage these elements	services that affect the operation of the NAS” (FAA Order 6000.30F)
<ul style="list-style-type: none"> ▪ Secure <ul style="list-style-type: none"> – Security built into DevSecOps tools and processes – Each layer inherits controls from layers below – Zero Trust <ul style="list-style-type: none"> • Increased emphasis on end-to-end security, authentication, and authorization • Decreased emphasis on network layer security mechanisms (perimeter gateways and firewalls) – Cyber-defensive operations provide monitoring and response 	<ul style="list-style-type: none"> – Baking security into development early reduces vulnerabilities and reduces cost and time associated with obtaining authorization to operate – Ability to inherit security controls from lower layers reduces rework, cost, and time needed to obtain security approval – The current perimeter protection approach allows intrusion to propagate laterally once the boundary is penetrated – The need to create gateways and manually configure network access control at network domain boundaries increases costs and delay and creates bottlenecks – Security threats continue to grow and evolve and must be addressed during operations, as well as during development
<ul style="list-style-type: none"> ▪ Multiple Levels of Criticality <ul style="list-style-type: none"> – Provides performance, reliability, and availability commensurate with safety-critical, efficiency-critical, essential, and routine service threads 	<ul style="list-style-type: none"> – NAS includes service threads at all levels of criticality – Architecture must include the ATC systems that are the major drivers of cost and delay in the NAS

A more detailed description of each characteristic, and how it is manifested in the architecture, is provided in the following sections.

2.4.6 Layered

The layers that make up the Reference Architecture are shown in Figure 2-2.

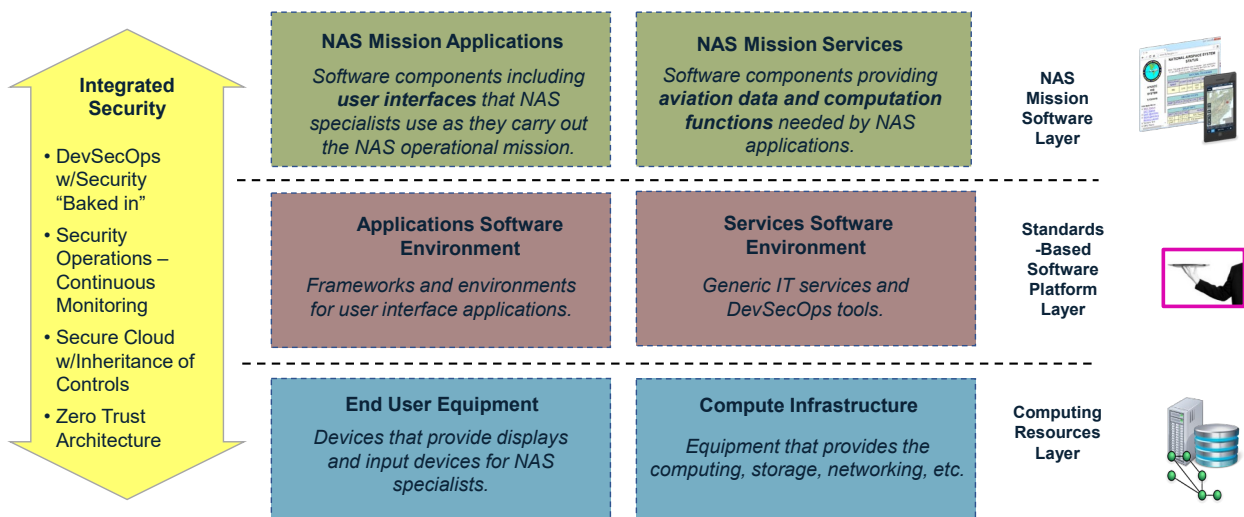


Figure 2-2. Layered Architecture – Overview

A key concept is that the different horizontal layers are intended to be acquired and sustained separately. The layers of the architecture are as follows:

- The *NAS Mission Software Layer* (or just *Mission Layer*, for short) contains NAS Mission Applications and NAS Mission Services. Mission applications are the user-facing software assemblies that provide the functions needed by FAA specialists to do their jobs. Mission services are software components that provide the data and computation services needed by the applications via well-defined APIs. Software components in this layer are likely to be specifically developed for the NAS, in contrast to generic software components in the layers below.
- The *Standards-Based Software Platform Layer* (or just *Platform Layer*, for short) consists of general-purpose IT software that provides the environment within which the mission software layer components can run. The Standards-Based Software Platform Layer contains off-the-shelf (both commercial and open-source) software components that provide generic IT that might be used in any large, complex enterprise. This layer includes DevSecOps tools that comprise the Software Factory concept, providing a repeatable process that allows mission software layer applications and services to be developed, integrated, tested, integrated, delivered, and deployed. This layer also creates the environment in which mission applications and services run, including a control plane that creates instances of mission services as needed for performance and availability. It also mediates service invocations, routing service requests, and information exchanges. The Platform Layer ensures that all accesses to resources are controlled with zero trust end-to-end authentication and authorization.
- The *Computing Resources Layer* provides the underlying computing, storage, networking, and end-user devices (e.g., workstations) to support the layers above. Like the Platform Layer, it is expected to be provided, operated, and sustained separately and support many NAS services. The computing resources layer will be built using commodity hardware (for workstations) and will use commodity cloud resources that provide on-demand provisioning.

Layering enables an important IT concept known as “separation of concerns.” Separating architecture elements into layers allows teams to focus on what they do best. Agile/DevSecOps teams create application software to meet mission needs, platform engineers assemble off-the-shelf IT tools, and service providers offer commodity computing resources.

Development teams creating mission software layer applications and services will use a Platform Layer provided to them by a separate platform engineering organization. Use of a common Platform Layer avoids proliferation of tools, simplifying licensing and training of IT staff. The availability of a robust, secure, resilient Platform Layer reduces the scope of development efforts, making it possible for smaller development teams to participate. Mission software development teams can be nimble in responding to new concepts and changing environments since they can focus on mission needs without having to expend effort reinventing the functions provided by lower layers.

The layered approach also improves NAS security. Security controls implemented and authorized in one layer can be inherited by the upper layers, shortening the process for obtaining an authorization to operate for new applications. The software factory provided by the Platform Layer ensures that security is baked into the mission software development and quality assurance (QA) process.

The lower layers can evolve independently as technology advances. For example, cloud and container technologies allow the underlying hardware of the computing resources layer to be upgraded without impacting the mission software layer. This eliminates costly technical refresh activities that currently must be built into the lifecycle of NAS systems. There will need to be policies defining minimal acceptable lag in technical debt, as well as dedicated budget for the evolution of the lower layers.

The layered architecture provides NAS scalability and resilience. Mission applications and mission services will be designed to be orchestrated by the Platform Layer, for example by instantiating new copies of services as needed for performance, leveraging elastic scaling² provided by the computing resources layer.

2.4.7 Service-based

The Reference Architecture is service-based. A service is software that implements some set of related functionalities, is accessed through a well-defined secured interface, and is designed so it can be used by multiple clients or other services. Services should be designed so they are only loosely coupled and minimize dependencies.

Shifting from acquisition of entire systems to an approach in which development teams create services and applications allows for parallel independent development of small units of functionality. Those small units of functionality can be quickly completed and made available for operational use, reducing time-to-market. That approach also increases the opportunity for industry competition. Rather than a single vendor being the only entity that can make changes to a large and complex system, that strategy emphasizes independent development teams that can create small units of functionality with well-defined interfaces, which can be sustained within a well-defined ecosystem created by the underlying Platform Layer.

In the Reference Architecture, NAS functionality (as identified in the NAS EA) is provided in the form of applications and services that communicate through service interfaces, which may be implemented via:

- Event-driven messaging on a message bus
- API Gateways
- Service Mesh

The Reference Architecture is based on a pattern that is currently being widely promoted and adopted by industry known as microservice architecture, which is an evolution of an older pattern known as service-oriented architecture (SOA). The Reference Architecture is referred to as a service-based architecture rather than a microservice architecture or SOA because we are introducing some refinements to avoid known issues with those architecture models. Contrasted with a pure microservice architecture wherein a very large number of microservices are developed, deployed, and their APIs exposed to users, the Reference Architecture exposes a more limited set of microservices from each application domain. That reduction in exposed services reduces the complexity of integration, configuration management, and sustainment. Contrasted with SOA, which is associated with older technologies such as Enterprise Service Bus (ESB) products, the Reference Architecture assumes more fine-grained services and modern

² Elastic scaling is the capability of automatically adding or reducing computing resources based on demand for those resources on a near real time basis.

technologies that emphasize scalability and support for DevSecOps methodology. These distinctions will become clearer as more detail is added in subsequent sections.

2.4.8 Operational Monitoring and Management

An overview of Operational Monitoring and Management concepts in the Reference Architecture is provided in Figure 2-3.

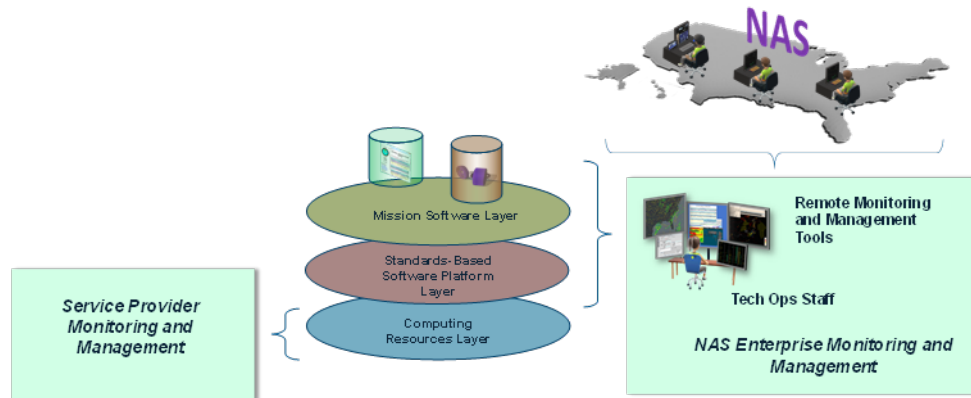


Figure 2-3. High Level Concepts for Operational Monitoring and Management

The Technical Operations (TechOps) organization is responsible for the performance of equipment, systems, and services that affect the operation of the NAS [11]. To carry out that responsibility, TechOps staff use remote monitoring and management tools. Mission services and applications must be designed to provide necessary status and performance indications, in real time, to provide data to existing monitoring and management tools, or to provide new, equivalent dashboards and status monitoring capabilities. At the standards-based software platform and computing resource layers, extensive support for monitoring and management are built into existing off-the-shelf tools. Sophisticated tools and dashboards are also provided as standard offerings along with cloud services.

In addition to FAA organic NAS enterprise management and monitoring capabilities, computing resources service providers take on responsibility for infrastructure components. For example, telecommunications service providers operate Network Operations Control Centers. Cloud service providers monitor and manage their datacenters and networks. That reduces the effort required for the FAA to monitor and manage resources at the computing resources layer, but TechOps will need to monitor cloud service quality levels to ensure that service level agreements are being met. TechOps will also need monitoring capabilities so that they are aware of any cloud service outages that would affect the FAA. In some cases, TechOps staff may be able to take corrective action (e.g., activating a contingency plan that switches to a different Cloud Service Provider [CSP]). TechOps would also be able to alert facility staff for any necessary operational mitigations. Collectively, the changes associated with the reference architecture suggest the need to re-examine the roles and responsibilities of TechOps and Second Level Engineering (SLE) and the future relationship with cloud service provider operations.

2.4.9 Secure

The Reference Architecture includes an emphasis on improved information security. While there are many aspects to better security in the proposed architecture, we highlight four interrelated facets illustrated in Figure 2-4. Those are:

- A layered architecture in which each layer provides a hardened secure foundation for the layers above
- The DevSecOps process that includes security requirements as equal to functional requirements
- A ZTA that implements strong security with granularity, visibility, and resilience against breaches, and
- Cyber Defensive Operations, including a Security Operations Center (SOC) that monitors and responds to indications of intrusion.

Those facets work with and, in many cases, will improve the existing security practices in the organization such as security awareness training, physical security, insider threat detection and mitigation, and so on.

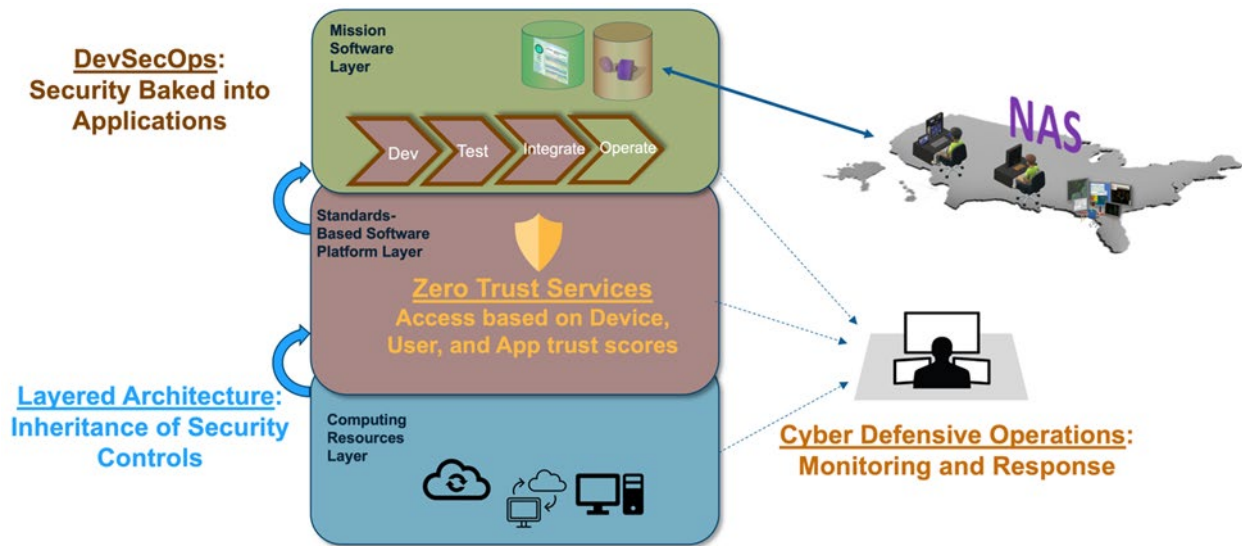


Figure 2-4. Security Overview

2.4.9.1 Inheritance of Controls

In the Reference Architecture, mission services are the focus of program efforts, building on top of lower layers provided by the enterprise to multiple programs. That approach allows the mission service to rely on the lower layer services, including the security features of those lower layers, without having to redesign them or repeat analysis or testing to validate their security features. In Figure 2-5 below we highlight how the Reference Architecture significantly relieves individual programs of security requirements, most of which can be designed, implemented, vetted, monitored, and refactored for the FAA by other organizations.

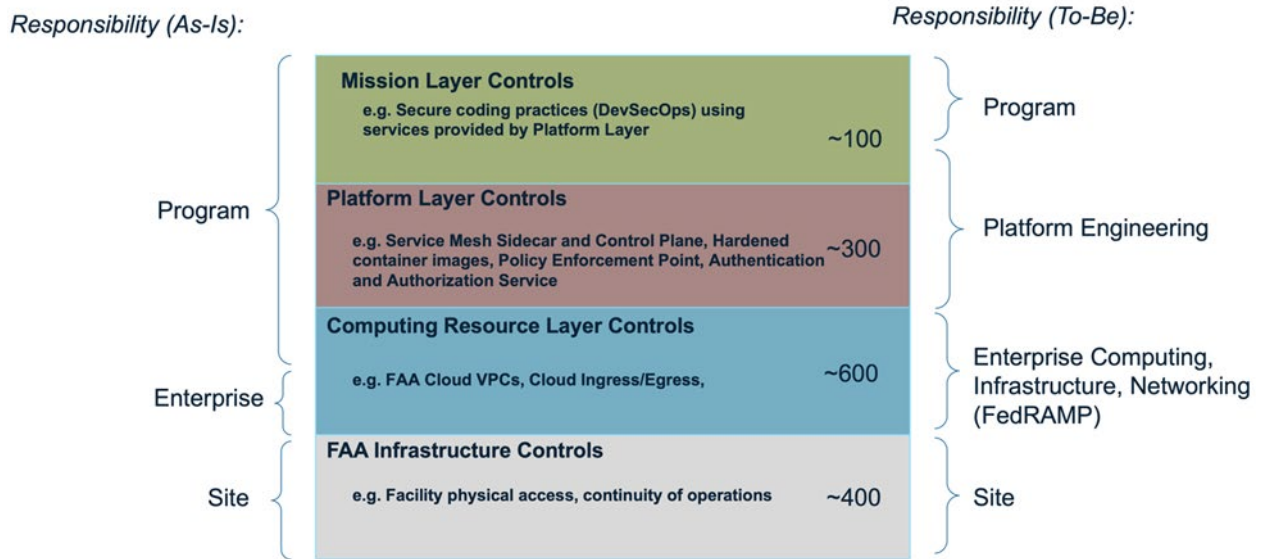


Figure 2-5. Inheritance of Controls

Any NAS system must meet National Institute for Standards and Technology (NIST) confidentiality, integrity, and availability controls according to its sensitivity and criticality. If a program were to acquire its own computing resources, it would be responsible for implementing and assessing a complete set of security controls at all layers. Currently, there are only a limited set of controls that are provided at an enterprise level. For example, the NAS Enterprise Security Gateway (NESG) provides boundary protection for the entire NAS. The NIST guidance, when applied to a specific NAS system, might result in roughly 400 controls that apply to the infrastructure, 600 to the computing resources, 300 to the platform resources, and 100 to the mission service. In the legacy approach, a program might be responsible for implementing and maintaining more than half of the controls. By contrast, if a program were able to use the approved platform, which rests on top of the approved computing resources, it would be responsible for only a fraction of the controls. Additionally, standardized containers that provide for secure communication, authorization or configuration can further reduce the security effort required of the development teams. The result is that a program would be responsible for only a fraction of even the mission software layer controls.

2.4.9.2 Security Built-in to DevSecOps

In DevSecOps, information security is baked into the development process. Legacy software development has largely treated the implementation and evaluation of the security of the system as a process independent of the development of the functionality of the software, often starting later in the *waterfall*. That often produced a deployment bottleneck as the enterprise evaluated at the end of development what changes were needed (or possible) to obtain an Authority to Operate (ATO). By contrast, the security requirements and metrics needed to determine risk and provide artifacts for ATO happen alongside the incremental, continuous development and deployment of system features from inception through the minimum viable product and beyond. Figure 2-6 below illustrates the DevSecOps processes.

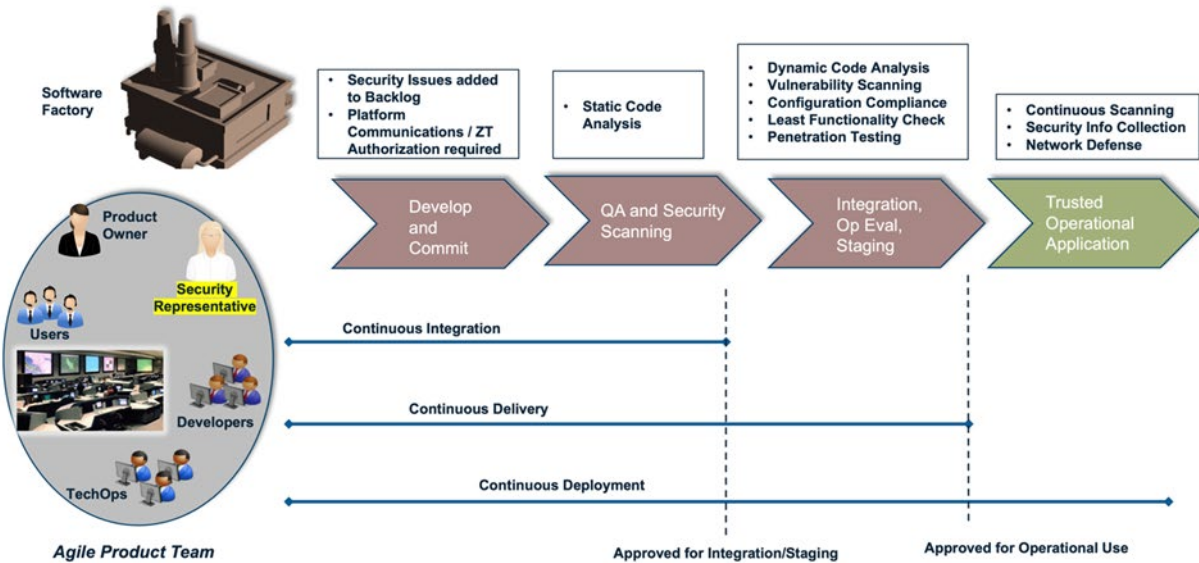


Figure 2-6. Security in DevSecOps

As illustrated above, many security features are required in the DevSecOps pipeline and are applied with each iteration as part of the process. These include but are not limited to dynamic and static code analysis, vulnerability management, risk assessment, ATO artifact generation, and signature validation. Penetration testing, continuous monitoring, and security data collection for use by the SOC occur on the deployed application or deployment images in appropriate environments.

2.4.9.3 Zero Trust

Zero Trust Architecture (ZTA) is an approach to information security that improves upon perimeter-based security by requiring all resource requests to be authenticated and authorized on a per session basis regardless of their position with respect to enterprise infrastructure. As a result, breaches are more difficult to propagate, defensive operations are more effective, and granting access can be more flexible. As security is devolved to each service and enforced by the security features of the Platform Layer, boundary protection, the paradigm of *inside/outside*,³ and the systems that define and enforce that model can be transitioned away.

Figure 2-7 illustrates the concepts of a ZTA in the Reference Architecture. Every resource request (for data or computation service, etc.) from a user or another system is authenticated and checked for authorization to access that resource. That contrasts against the model of an enterprise boundary where things on the *outside* are presumed to be untrustworthy and things on the *inside* are broadly able to communicate with each other without scrutiny.

³ The paradigm of having a trusted security perimeter where things inside the perimeter are trusted and authentication and authorization occur at that boundary. Zero trust no longer relies on authorization at the boundary, moving away from this paradigm.

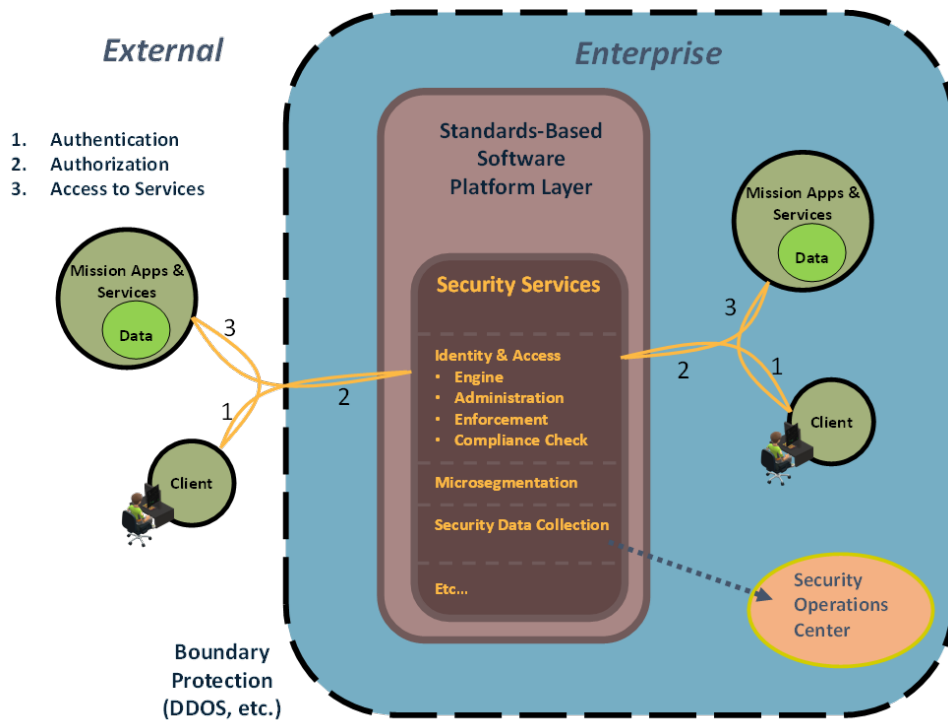


Figure 2-7. Zero Trust Concepts in the Reference Architecture

In the Reference Architecture, authentication and authorization is implemented in a uniform manner in the Platform Layer. Policy enforcement points enforce access decisions made by a policy engine. Microsegmentation prevents arbitrary communication among parts of the enterprise. Authorization policies are dynamic and adapt to threats according to rich security data collected by the platform. The policy system enforces compliance policy to reduce the risk that any of the components of the enterprise is vulnerable to exploitation. The rich information collected by the platform, and detailed knowledge of the assets and authorized communication patterns of the enterprise, is sent to the SOC, supporting effective cybersecurity defensive operations.

The Platform and Compute layer needs specific infrastructure to implement the ZTA in a secure, efficient, scalable, and resilient manner. This infrastructure will provide for trust of users, devices, applications, and networks through ZTA. Container sidecars or other standardized mediation communications implement the facilities needed to participate in the security regime. This mediation can be host agent based or a Zero Trust gateway. Host agents may be required for the compliance verification aspect of Zero Trust across all systems.

The Reference Architecture includes infrastructure, procedures, and policies to provide for the resilience of the Zero Trust communications in the face of failure modes or compromise to ensure the safe operation of the FAA mission while maintaining security.

2.4.9.4 Cybersecurity Defensive Operations

Security Operations Centers (SOCs)⁴ provide the NAS with cybersecurity defensive operations. The SOC's consist of dedicated teams of cybersecurity professionals tasked with identifying cybersecurity events of interest and determining if malicious activity is present in the FAA. To do that, SOC's collect information from end systems and network monitoring combined with threat intelligence and other sources to detect anomalous behavior and determine the cause, severity, and remediation of cybersecurity incidents. The teams use a variety of tools, partnerships, and capabilities for this work. The data provided by the architecture's security collection features supports the capabilities of the SOC's. The Reference Architecture includes a mature SOC capability which can include threat intelligence, SOC collaboration, malware analysis, threat hunting and deception operations [12].

2.4.10 Multiple Levels of Criticality

The Reference Architecture supports multiple levels of criticality, which are defined in the FAA's Reliability, Maintainability, and Availability (RMA) Handbook [13] as: safety-critical, efficiency-critical, essential, and routine.⁵ Safety, operational, and engineering analyses must be conducted consistent with the existing process. Each mission service or application must be assessed within the context of the end-to-end service threads that it supports, to determine requirements (i.e., availability and latency requirements) appropriate for the criticality of operations being supported. The Mission Layer software must be designed accordingly, and the Platform Layer and the associated Computing Resources Layer must be configured to meet the requirements. Figure 2-8 provides a high-level overview of how the different levels of criticality are supported in the Reference Architecture. The figure is explained below, and more detail is provided in subsequent sections.

⁴ There are multiple entities that have differing roles and capabilities related to security operations in the FAA. These include the Department of Transportation SOC, the NAS Cybersecurity Operations (NCO) group, security operations functions performed at a facility or system level by application owners, and third-party SOC's operated by FAA telecommunications and cloud service vendors. Further elaboration is beyond the scope of this document.

⁵ NAS systems and networks are currently grouped into network security domains that consider these same system criticality levels. However, the emphasis of this section is on RMA and performance rather than security. The Reference Architecture approach to security is based on Zero Trust principles that reduce the emphasis on network layer security domains, as discussed in the previous section.

Service-Based Reference Architecture for NAS Automation - Version 1.2

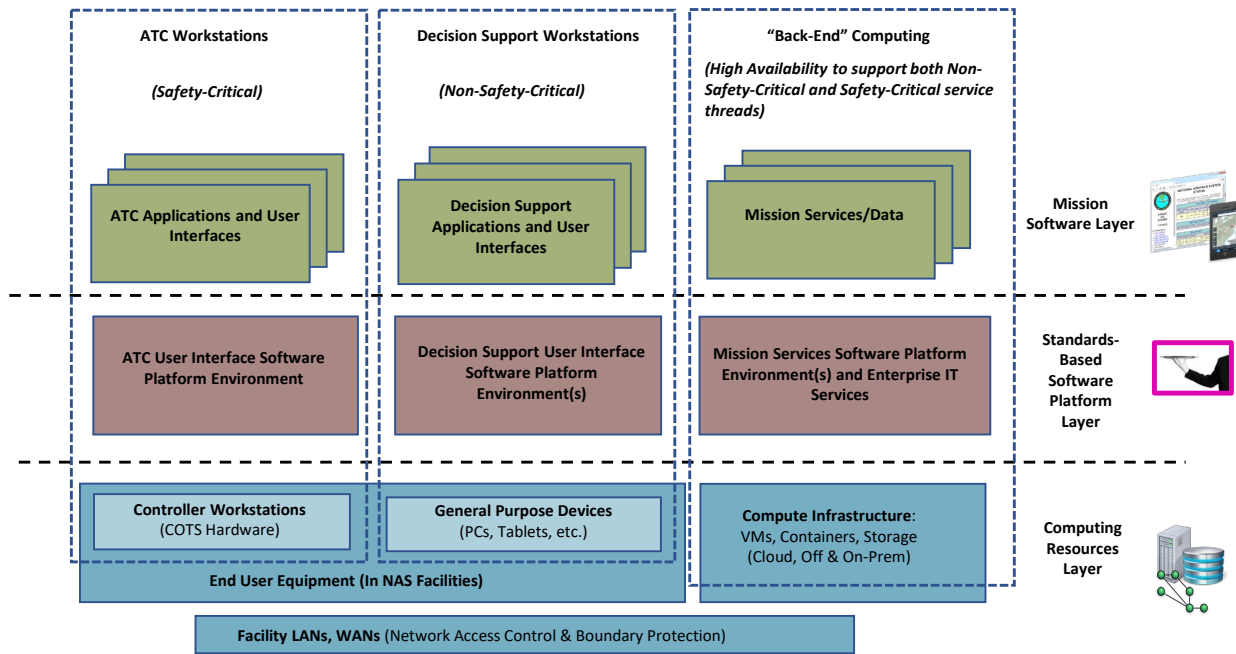


Figure 2-8. Layered Architecture Supporting Multiple Levels of Criticality

Consider first the “Back-End” Computing column. That column represents mission functions and data services executing in managed container environments provided by the Platform Layer and hosted on cloud computing infrastructure (potentially both on and off premises) provided by the Computing Resources Layer. That approach represents the current commercial mainstream, and is the expected implementation approach for routine, essential, and efficiency-critical functions. It may also be the implementation approach for safety-critical functions with sufficient infrastructure configuration and performance tuning to meet requirements that are more demanding. Considerations to be addressed within the Platform and Computing Resources layers are the number of service instances to be implemented, geographical distribution for performance and to mitigate failure impacts, load balancing for performance, service and data synchronization, and computing and communication resource prioritization.

Next, consider the left two columns, which represent the user-facing air traffic management (ATM) applications. These applications consume and coordinate mission data and computing provided by Mission Services and provide user interfaces to end users in FAA facilities. The middle column of Figure 2-8 represents the collection of Decision Support elements that are not safety-critical. These elements are expected to be implemented using commercial mainstream approaches, for example with a web browser environment providing the Platform Layer running on Computing Resources made up of general-purpose devices (personal computers [PCs], etc.). The left column of Figure 2-8 represents the collection of ATC elements that are safety-critical and must be implemented in a consistent way that recognizes that criticality. Considerations for safety-critical applications include redundancy and design for operational resilience, demanding performance requirements, and demanding requirements for data and function integrity. Meeting these requirements may require specialized Platform Layer software running on specially configured controller workstation hardware. In addition, facility-based processing, storage, and data distribution elements may be needed to support critical functions.

3 NAS Mission Applications and Mission Services Layer

3.1 Overview

The NAS Mission Applications and Mission Services⁶ Layer (or Mission Layer, for short) contains applications and services that provide functionality to support the mission of the NAS. Software components in this layer are likely to be specifically developed for the NAS, in contrast to generic software components in the layers below.

Figure 3-1 provides an overview of the Mission Layer. The elements that make up this layer are described below.

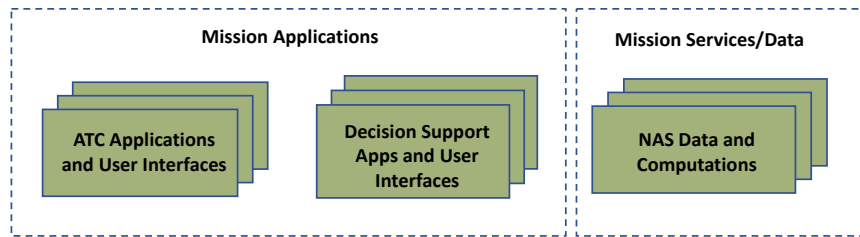


Figure 3-1. Mission Layer

Mission services are software components that provide mission-specific data (e.g., flight data, surveillance data, and aeronautical data) and computation functions (e.g., tracking, weather prediction, and conflict probe) needed in the NAS.

Mission applications are software assemblies that provide the capabilities needed by FAA specialists (e.g., controllers, traffic managers) to do their jobs. Mission applications contain a user facing component which consists of a graphical user interface that accesses mission services to obtain data and perform computations needed by the end user.

Sections 3.2 and 3.3 provide more description of NAS Mission Applications and Services, and how they fit into the overall Reference Architecture. However, the Reference Architecture only provides a general description; it does not define specifically which services and applications will be needed in the NAS.

3.2 Mission Applications

The Reference Architecture does not attempt to define specific mission applications or mission services. The Traffic Services and Mission Support Services⁷ called out in the NAS Requirements Document [15] suggest categories of mission applications that will be needed, including applications to support:

- Traffic Services
 - Flow Contingency Management
 - Separation Management

⁶ Not to be confused with the term “Mission Support”, which is a term used to refer to functions that indirectly support the mission of the NAS but are not used directly in near-real-time air traffic control and management operations.

⁷ The “Traffic Services” and “Mission Support Services” referred to in the NAS-RD are not microservices or mission services as those terms are used in the Reference Architecture. Rather they are high-level services provided by a combination of humans, operational procedures, and automation, including Mission Applications as described in this document.

- Short Term Capacity Management
- Trajectory Management
- Mission Support Services
 - Long Term Capacity Management
 - Safety Management
 - System and Services Analysis
 - System and Services Management

The Mission Applications category is divided into two sub-categories:

- Air Traffic Control (ATC) applications, and
- General decision support applications.

ATC applications provide user interfaces for Certified Professional Controllers (CPCs) to provide separation services. These software components must be integrated to provide the CPC with a user interface that is suitable for safety-critical operations.

General decision support applications provide the remainder of user interfaces used by FAA specialists while operating the NAS and performing mission support activities. This includes applications used for flow management applications, applications to allow display and entry of aeronautical information, weather information, dashboards that show the status of the NAS, and so on. These applications will be used in service threads that may be routine, essential, or efficiency-critical.

Applications in the Reference Architecture are not tied to any particular end-user computing equipment (workstation). Rather, they are built to run in a standard software environment defined by the Platform Layer, and therefore can run on any equipment that provides the necessary characteristics (display size and resolutions, input devices, etc.).

3.3 NAS Mission Services

This document does not attempt to define specific mission services, but the information services called out in the NAS Requirements Document [15] suggest categories of mission services that will be needed:

- Aeronautical Information Management Services
- Flight and State Data Management Services
- Surveillance Information Management Services
- Weather Information Management Services

The SWIM Business Services provide an existing baseline of mission services, such as:

- SWIM Flight Data Publication Service (SFDPS)
- SWIM Terminal Data Distribution Service (STDDS)
- Traffic Flow Management Data Service (TFMData)

An important aspect of the Reference Architecture is that mission services are not bundled with application software into monolithic systems, but rather run separately and are accessed over the

network. Applications use lower-level services provided by the Platform Layer to locate, authenticate, and communicate with the mission services.

Mission services will support both ATC (e.g., conflict probe and metering) and Decision Support (e.g., traffic flow management) mission applications. When using mission services to support safety critical ATC applications, care must be taken to ensure that overall service thread performance and availability requirements are met. Further work is required to ensure that the proposed architecture is compatible with NAS requirements. The expectation is that availability requirements can be met by leveraging an underlying Platform Layer that supports high availability requirements like those used in industry, as discussed in Section 4. For example, if a mission service is being used for an ATC application, then loss of availability (or connectivity) of a mission service must be detected, and switchover to an alternate instance accomplished quickly enough to prevent a safety problem.

3.4 Basic Pattern: Microservice Architecture

The Reference Architecture is based on a microservice architecture, but some modifications and extensions have been made, which is why we refer to the Reference Architecture as *service-based* rather than a microservices architecture. The basic microservice architecture is described in this section, and the modifications and extensions are described in section 3.5. For more information on microservices and their use in the context of the NAS, the reader may wish to refer to the MITRE technical report “Guidance on Flight Information Management for Microservices” [16].

In the Reference Architecture mission services are generally implemented as microservices. The microservice concept is explained in Section 3.4.1.

These microservices are assembled, usually together with user interface components, to create mission applications, as described in Section 3.4.2.

3.4.1 Microservices Concept

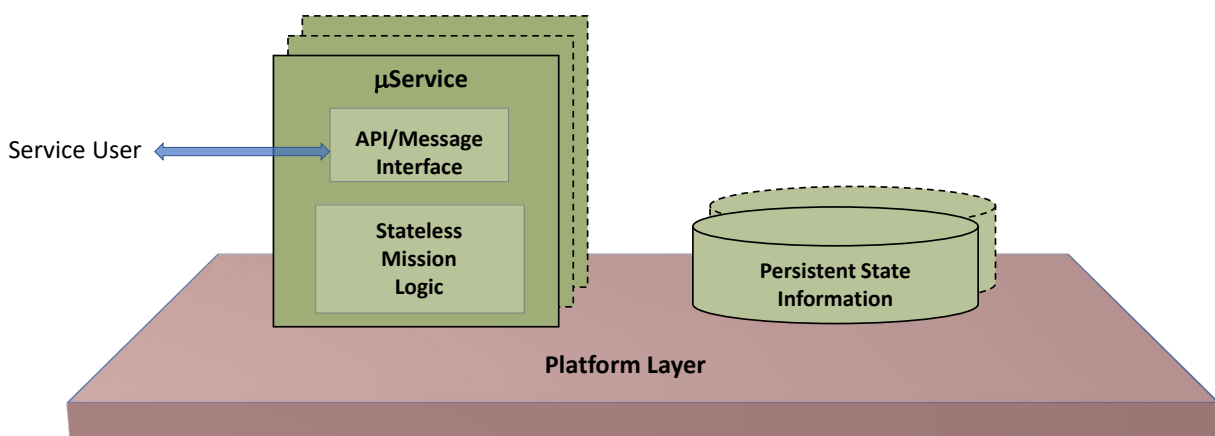


Figure 3-2. Microservice Concept

Microservices, illustrated in Figure 3-2, are basic building blocks of the Reference Architecture. This section describes the characteristics of a microservice, with a brief explanation of how these characteristics enable desired properties of the Reference Architecture.

A microservice is a software component that implements some business logic within a bounded context⁸, accessed via a well-defined interface over the network.

Access to the microservice is generally through a web service/RESTful API, an open standards-based web API⁹, or through a publish/subscribe messaging interface. Different service invocation patterns will be needed. A request/response pattern is appropriate for cases where an application needs to retrieve/pull data on demand, for example because of the user requesting new information to be displayed. A publish/subscribe pattern is often appropriate for the event-driven nature of the NAS when an event (e.g., a flight plan being amended) triggers a message that is published/pushed to multiple different systems that need to act.

Each microservice is ideally *stateless*, which means that a microservice does not keep track of the state of the user of the service. Because a microservice is stateless, an invocation of a microservice can be handled by any instance of the microservice, thus allowing multiple instances to run in parallel. Stateless does not imply that the microservice does not retain data. On the contrary, in many cases the purpose of a microservice is to keep track of the state of some real-world object or resource (e.g., a flight). This state information is persisted in an associated datastore.

A microservice runs independently in its own process, which should run separately from the datastore used for persistent state information.

The characteristics described above are important because they allow off-the-shelf software in the Platform Layer to handle lower-level concerns that should not be entangled with the business logic. For example:

- The stateless nature of microservices allows platform software to scale (up or down) the number of microservice instances as needed.
- The datastore can be replicated as needed for availability.
- The Platform Layer can mediate the API and message interfaces to ensure secure access and route service requests to service instances.

3.4.2 Assembling Microservices to Create Applications

Mission applications are created by combining a front-end with a set of back-end mission services, implemented as microservices. The front-end usually provides a user interface, and the back end provides the data and computing services that drive the user interface. This is illustrated in Figure 3-3.

⁸ The term “bounded context” is from the field of Domain-Driven Design. Very simply stated, it means that a service should do only one well-defined thing.

⁹ An example would be openAPI, which defines a standard interface to RESTful APIs.

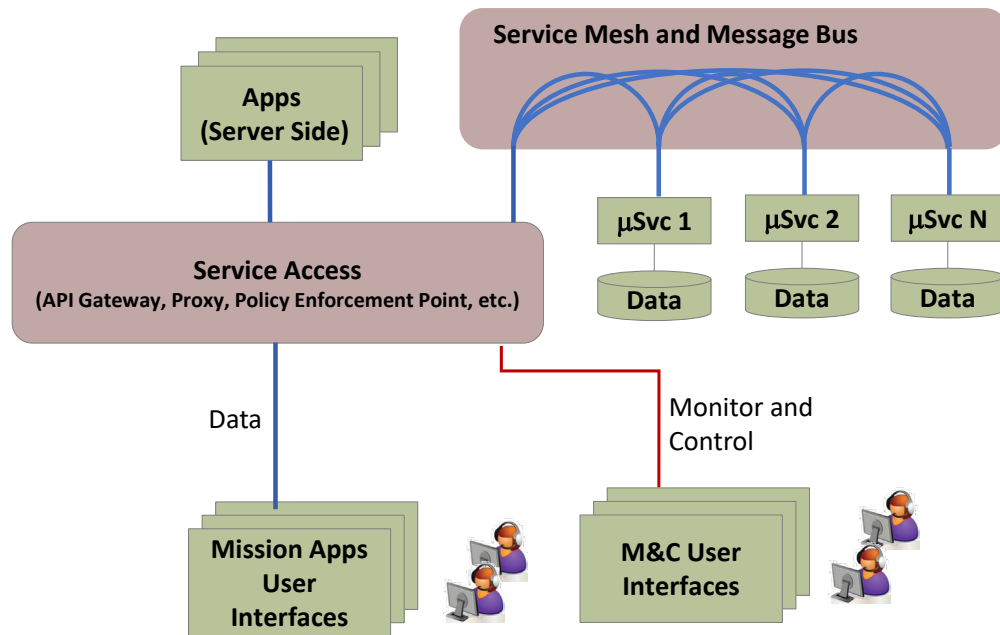


Figure 3-3. Application Assembled from User Interface and Mission Services

An important aspect of the proposed architecture is that mission services run separately and are accessed over the network, using the Platform Layer to locate, authenticate, and communicate with the service. Platform Layer elements shown in the brown rectangles in Figure 3-3 include Service Mesh, Message Bus, API Gateway, Proxy, and Policy Enforcement Point. These are described in more detail in Section 4.2.1.4.

3.5 Service-Based Architecture

The previous section described a basic microservice architecture, however, the Reference Architecture makes some adjustments, described in this section, resulting in what we call a “service-based architecture”.

This section discusses the adjustments, which include:

- Shared Mission Services versus Internal Mission Services
- Location dependencies
- Database-centric subsystems

3.5.1 Internal Mission Services Versus Common Mission Services

One of the known disadvantages of a microservice architecture is that, since microservices should be kept small, a very large number of microservices may be required, and this can be difficult to manage. If every microservice is made available for general use, then complexity grows, and it can become difficult to manage dependencies. Just as most program languages provide a means for hiding internal functions and variables, the Reference Architecture needs a way to hide microservices that are intended for use within a constrained scope. This is accomplished by exposing only a subset of the microservices as common mission services. A common mission service is a mission service exposed for use by other Agile development teams. This concept is illustrated in Figure 3-4.

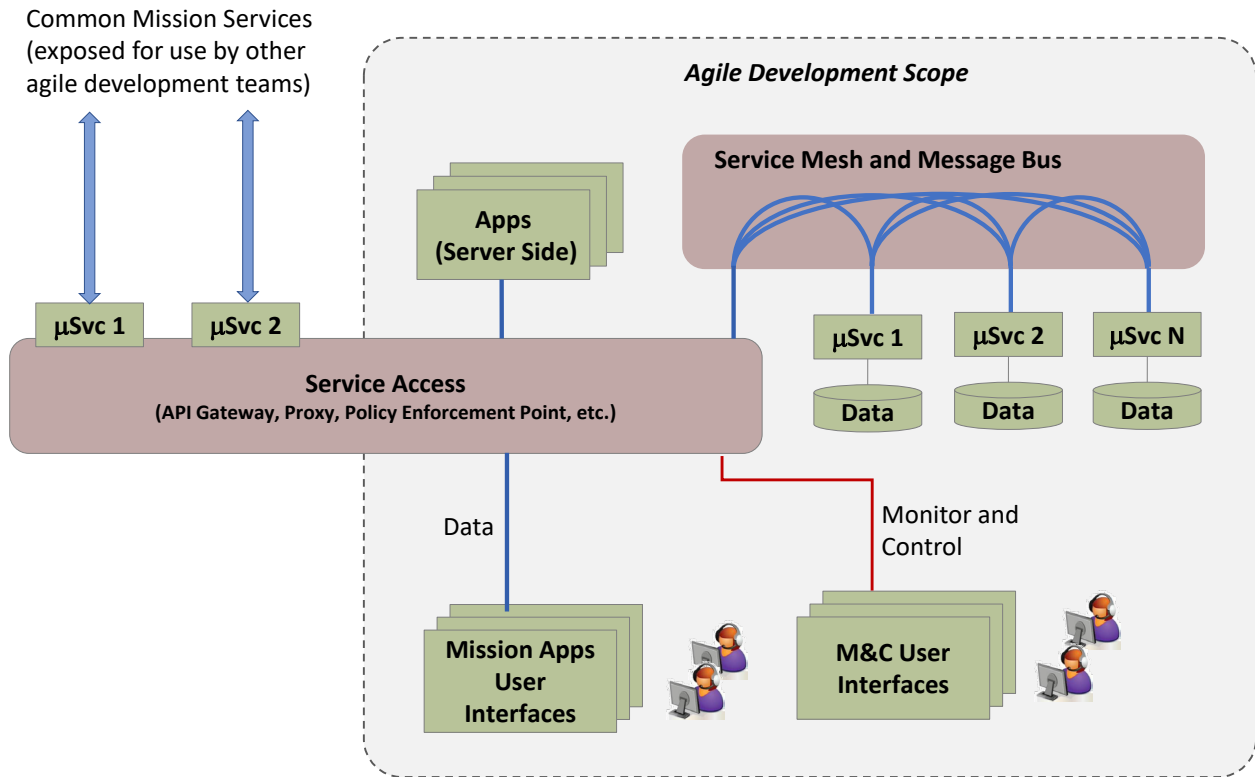


Figure 3-4. Microservices Exposed as Common Mission Services

The figure illustrates a set of applications and microservices (1, 2, ... N) that are being developed by a single development team (or perhaps a small number of teams working closely together as part of one larger development effort). These development teams may find the need to create many microservices, and some, but not all, of these microservices may be valuable to other development teams. Microservices that have broad value should be exposed as common mission services that can be used in other areas. In this example, microservice 1 and 2 are exposed via a service access point, while the remainder of the microservices are intended for local use only.

It is important to realize that the development scope boundary shown in Figure 3-4 does not represent a geographical area or facility boundary. Users in each facility should have access to applications developed independently, and any one of these applications may be drawing on common mission services that have also been developed independently.

3.5.2 Geospatial Dependencies

The Reference Architecture as described so far is location agnostic. Microservices do not have to know where they are running and instances of a microservice have no functional location dependencies. Often these assumptions are true, and any instance of a microservice being run anywhere can handle an API call to that microservice. For example, a microservice that calculates the bearing going from one latitude/longitude location to another can do so regardless of what data center it is running in, and the calculation is the same whether those points are in Alaska or Vermont.

On the other hand, there may be times where a microservice is location dependent, in one of two ways:

- A service may need to run in a particular location (at a particular facility or on specific hardware) to meet non-functional requirements for performance, availability, latency, bandwidth, workload, etc. These considerations for localization in the NAS are often characterized as performance driven.
- The functionality of the service may need to vary depending on location of the user invoking the service (e.g., which facility the user is in), or an object being referred to in the service request (e.g., the location of a flight). These considerations for localization in the NAS are often characterized as adaptation driven.

These two cases are described below.

Non-functional location dependency (performance driven). An example of non-functional location dependency would be a levied requirement that a system within a facility must continue to run even if outside network connectivity is lost. In that case, computing infrastructure would need to be located at the facility and API calls to a service would need to be routed to instances of the service running within the facility compute infrastructure.

This kind of dependency could be met by, for example, running a container orchestrator cluster on compute infrastructure located within the facility. Any requests between services within that container orchestrator would automatically be routed within the orchestrator cluster. And APIs exposed by the orchestrator would again route to within the facility. A separate API gateway would need to be running for the system at the facility, and it would be setup to route API calls to within the facility.

These kinds of accommodations inevitably involve some tradeoffs. For example, limiting API requests to being serviced by service instances running on compute infrastructure at the facility, might create the need to purchase more compute infrastructure than would be necessary if load balancing across the NAS. Consideration should be given to whether RMA, performance, and cost requirements can be met without secondary requirements requiring location-dependent computing.

Functional Location Dependency (adaptation driven). The second kind of dependency occurs when the functionality of a service is location dependent. This is an application layer design issue, not an architecture issue. It is mentioned here only to show how these kinds of issues supported within the Reference Architecture.

For example, consider a mission service that returns track positions for a controller display. There may be a requirement that the controller should see target positions only from a single radar when performing separation functions. This could be implemented by, for example, tagging track data with the source radar identifier. Then when the track position service is called, the radar ID can be included in the API, and the service would only return track data that came from the specified radar. Note that in this example, there is no need to limit where track position service is running; that can be independently adjusted as needed for non-functional requirements (performance, etc.) as described above.

3.5.3 Support for Transactions and Data-Centric Subsystems

A fundamental principle of a microservice architecture is the movement away from centralized databases to service-based data stores. Figure 3-4 illustrates this practice by showing each

microservice with its own data store.¹⁰ While this is highly desirable because it decouples data used by services, sometimes a strict insistence on this decoupling of data can lead to overly complicated software. An example of such a case is when Atomicity, Consistency, Isolation, and Durability (ACID) transactions involving several services must take place.¹¹ ACID transactions might be needed in situations such as aircraft handoff, to guarantee that one and only one controller is responsible for a flight at any given time. Shared databases often have specific features designed to simplify implementation of such transactions.

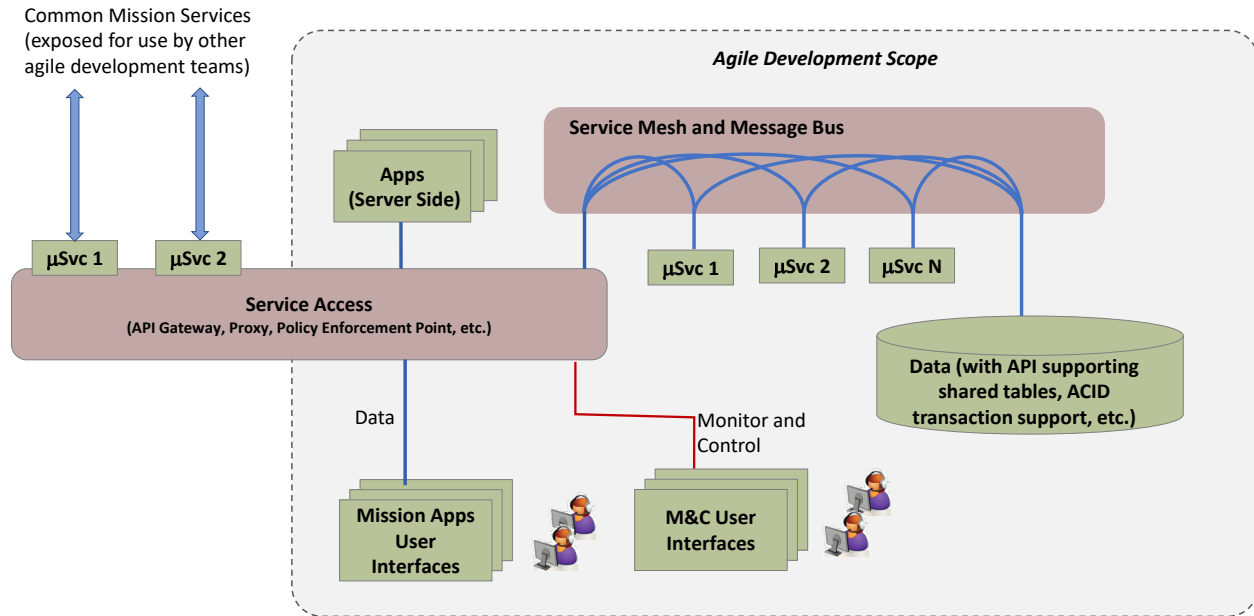


Figure 3-5. Use of Shared Data

Figure 3-5 illustrates a modification of the basic Reference Architecture to accommodate such cases. Notice how microservices 1 through N now share data that is stored within a common database (contrast this with Figure 3-4).

While this modification of the basic Reference Architecture may be necessary in some cases, it should be avoided if possible. ACID transactions come at a performance cost. Also, a shared database creates coupling between services that may make the system more difficult to modify as new capabilities are to be added. Eventual consistency should be used rather than ACID transactions when sufficient to satisfy requirements.

It is important to note that despite this modification of the basic Reference Architecture, exposed services still behave as before. Users of the exposed APIs do not need to know if microservice 1 and microservice 2 are communicating via a shared database. However, internally, those decisions may have very large impacts on cost, maintainability, and extensibility. Those decisions could also impact performance, and that can be an issue to the NAS.

¹⁰ Note that what is important is that the data that is persisted by each service is independent and stored and accessed only by that service. The database providing that persistence can be (and often is) used by multiple services. This provides for the desired decoupling of data between services.

¹¹ These are properties of a transaction that help guarantee validity of results.

3.5.4 NAS-Wide View of Service-Based Architecture

Figure 3-6 provides a NAS-Wide view in which applications and services developed and sustained independently are combined to provide the full suite of automation functions needed for the NAS.

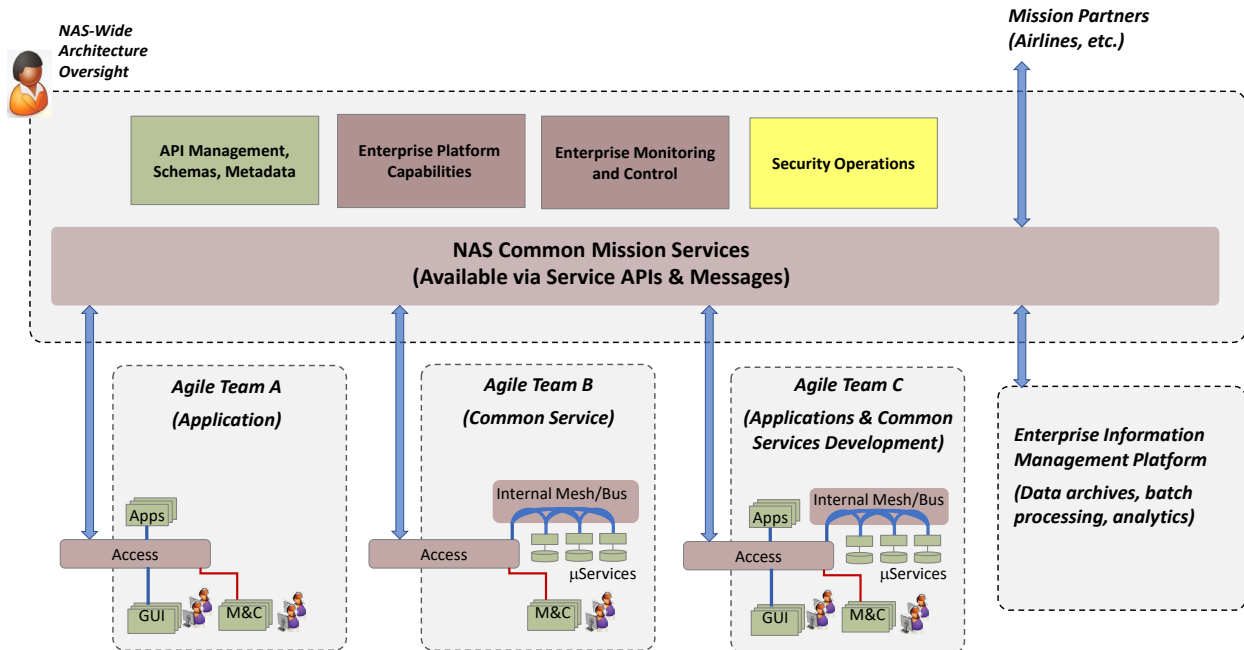


Figure 3-6. NAS-Wide View of Services and Applications

Figure 3-6 provides an example of how the results of multiple Agile development efforts are integrated to provide the automation capabilities needed to support operations. In this example, Agile team A has created applications that provide a graphical user interface (GUI), and which use common mission services provided elsewhere for the necessary back-end data and computation. By contrast, Agile team B is responsible for common mission services only (with no user interface), whereas Agile team C is developing both applications with user interfaces as well as mission services. Depending on the magnitude of the development efforts, each of these Agile teams A, B, and C, might need to consist of multiple Agile teams working together on a set of closely related capabilities.

Just as service mesh and message bus solutions are used for internal communication among components developed and sustained by a single team, they can also be used at NAS-wide scale (e.g., SWIM) to support access to common services. Mediation and transformation services, monitoring and control, and security operations will also be needed at a NAS-wide level.

Any common services need to be more extensively documented than internal services that need to be known only by teams working within a given area. Platform Layer tools (see Section 4.2.2.4) support this by providing a NAS-wide service registry, metadata, and API management.

Mediation and transformation services can also be provided as enterprise-level services. These can be used to incorporate data from legacy systems. They can also be used to solve interoperability problems and to facilitate transition to new versions of APIs and data structures.

Figure 3-6 also illustrates how the EIM Platform data lake can be accommodated by the Reference Architecture. Within the EIM platform different models of information sharing may

be used (for example, shared data accessed by multiple different services, as discussed in Section 3.5.3). However, these differences should be isolated from the rest of the NAS using APIs.

3.5.5 Monitoring and Control

The Reference Architecture provides robust facilities for monitoring and control. Mission services and applications are instrumented, and user interfaces are provided to allow TechOps staff at facilities and in the NAS Enterprise Operations (NAS EO) organization to monitor and control the automation services.

With the rich information provided, TechOps will have access to both highly granular and aggregate metrics on system performance which will provide for fast and highly informed understanding of service state and the ability to control services as needed.

3.5.6 Security Monitoring and Defensive Operations

The Reference Architecture will provide robust security monitoring and facilitate defensive operations. As described in Section 2.4.9, Mission Services and Applications will leverage the security provided by ZTA. Applications and services are created with security instrumentation by default, rather than a sensor, agent and logging infrastructure needing to be applied a posteriori.

4 Platform Layer

4.1 Overview

The Standards-Based Software Platform Layer (or more briefly, Platform Layer) consists of general-purpose IT software that provides the environment within which the Mission Layer components can run. Software in the Platform Layer is not likely to be custom developed for the NAS. Rather, this layer is expected to be assembled by licensing, configuring, and operating a suite of needed enterprise IT tools, including both commercial-off-the-shelf (COTS) as well as free and open source (FOSS) components. Cloud Platform-as-a-Service (PaaS) offerings may also make up part of this layer.

The existence of the Platform Layer relieves mission software developers from the effort and cost of selecting, licensing, configuring, and sustaining these IT components. Rather, mission software developers can focus on the application layer code that will run in the environment provided by the Platform layer, which is created, operated, and sustained separately. That reduces rework and allows TechOps and SLE staff to focus on a finite, controlled set of tools and technologies.

The Platform Layer also promotes modularity in that it decouples the Mission Software Layer from the underlying Computing Resources layer. NAS mission software application developers can develop to the platform defined by the Reference Architecture, regardless of how the underlying computing resources layer will be provided. For example, the same Platform Layer can be made available on different CSP vendor environments and in FAA on-premises environments. Inclusion of CSP-specific PaaS offerings in the Reference Architecture Platform Layer is problematic in this regard, and therefore should only be used judiciously. The ability to use multiple CSPs in the Platform Layer, without specifying particular cloud vendor capabilities, minimizes vendor lock-in while promoting interoperability.

As illustrated in Figure 4-1, the Platform Layer includes the environment within which applications (including both ATC and decision support user interfaces) are developed, as well as the environment in which mission services are developed. Instances of the Platform Layer can be created as needed to provide development, testing, staging and integration, and production environments. Those environments can be shared by multiple development teams in a *multi-tenant* model, or separate environments can be created for individual programs.



Figure 4-1. Standards-based Software Platform Layer

The following sections describe the Platform Layer in the Reference Architecture. Section 4.2 describes the Platform Layer in terms of generic *elements* that make it up, followed by a summary of preliminary recommendations for each of these elements in the Reference Architecture.

Section 4.3 describes a survey MITRE CAASD conducted of various projects and programs that used architectures or methodologies relevant to the Automation Evolution strategy. The results of the survey are contained in Appendix B.

4.2 Platform Layer Elements

This section describes the elements that make up the Platform Layer. Each of these elements represents a broad category of functional capabilities that are implemented in a wide variety of COTS and FOSS and PaaS tools or components. There are numerous components available that are evolving rapidly and often overlap or partially replace each other's functions, so a perfect categorization is not possible. Nevertheless, the Platform Layer elements described here are intended to provide a complete set of functions that will support the mission services and applications described in Section 3. Preliminary recommendations are included as to the best way to provide and use each element consistent with the architecture key characteristics laid out in Section 0, to achieve the strategic objectives described in Section 1.2. The recommendations are general; selecting specific tools for each element will be an ongoing activity separate from this Reference Architecture. We have provided examples of specific tools only as reference points to explain the elements.

4.2.1 Runtime

4.2.1.1 Software Hosting/Execution

Software hosting/execution capabilities consist of those capabilities that provide the runtime environment within which software components (services and applications) run, and manage the execution of components within the environment, including support for scaling and availability.

Example Software Runtime capabilities include:

- Support for VMs.
- Container runtime tools such as docker engine.
- Container Orchestration tools.
- Serverless computing.

Containers and container orchestration should be used where possible for running services and applications in the Reference Architecture, because it allows the Platform Layer to provide the following functions:

- Automate the deployment of containers at scale with multiple instances of a service based on schedule and load.
- Deployment of those instantiated services to multiple real or virtual machines that make up a cluster.
- Provision of a single API endpoint for those multiple instantiated services.
- Load balancing.
- Health monitoring and restart of failed service instances.

The use of container orchestrators also enables the use of service mesh technology (Section 4.2.1.4). The use of sidecar containers to provide service mesh and security functionality is encouraged.¹²

While containers and container orchestration are a fundamental part of the Reference Architecture, other approaches (e.g., running processes on VMs and Serverless Computing) should be supported as needed.

4.2.1.2 Workflow Choreography and Orchestration

Workflow choreography and orchestration capabilities manage the overall sequence of events necessary to implement some piece of business logic. In workflow orchestration one entity controls the invocation of a sequence of mission functions, whereas in choreography the sequence of mission functions is arranged without control from a single point.

Note that the term “orchestration” in this context refers to coordinating logic at the Mission Layer. This is quite different to “container orchestration” discussed in Section 4.2.1.1, which refers to managing instances of a microservice to provide performance and availability without regard to the sequence of mission logic to be executed.

Use of workflow choreography and orchestration helps decouple services from each other. For example, after one service finishes, either of two different services may need to be called, depending on the results of the first service. Rather than building that logic into the first service, which would create strong coupling, choreography or orchestration can be used to invoke the services in the right order, possibly passing the outputs of one service as inputs to the next. That minimizes the knowledge each service needs to have of other services, thereby keeping the services loosely coupled.

One risk in using these capabilities, especially orchestration, is the centralization of enterprise business logic in one place. Earlier ESB versions of these capabilities tended to suffer from performance bottlenecks and single-point-of-failure. Modern versions (e.g., Apache Airflow, Netflix Cosmos) are designed to be scalable and reduce these problems. Nevertheless, care must still be taken when using workflow orchestration to avoid centralizing the business logic in one place. Workflow orchestration risks creating bottlenecks in Agile development because it means that every service development team needs to interact with the one central team responsible for the workflow orchestration logic.

Choreography can be implemented with event-driven processing, in which events published on a message bus trigger the necessary processing. Choreography is preferred over orchestration in the Reference Architecture. Orchestration should be used with care to avoid too much centralization of business logic.

4.2.1.3 Monitoring, Log Analysis, and Reliability

Monitoring and Log Analysis capabilities provide monitoring, logging, tracing, and chaos engineering (stressing the system to ensure reliability). Examples include Prometheus, Fluentd, and Chaos Monkey. Those capabilities provide insight into how a system is behaving or, in the

¹² Some container orchestration tools such as Kubernetes allow running a group of containers together. In the case of Kubernetes, this group is called a pod. A sidecar container is one designed to run alongside other containers and encapsulate some functionality. For example, a sidecar container might implement a service mesh or security processing. The sidecar can then change without having to rebuild the main container containing a service or some other functionality.

case of chaos engineering, purposely stressing the system to ensure it behaves properly even when components fail.

The Reference Architecture log analysis capabilities can detect and report anomalous behavior and provide summary data suitable for use by human NAS operators (i.e., Technical Operations). For example, they are used to provide summaries of average response times for key services.

The Platform Layer provides standard facilities such as container sidecars and the service mesh (Section 4.2.1.4) to developers that facilitate robust and dynamically configurable logging and monitoring tools. Those will then be used by application owners, the enterprise, and information security operations.

Security data collection is required for mission services and applications. The types of data collected and forwarded to Security Operations include network flow data, authorization and authentication data from the zero trust entities, host agents, and hash checks.

4.2.1.4 Service Proxy, Mesh, and API Gateway

Service Proxy, Mesh, and API Gateway capabilities mediate service invocations. They may provide a control plane, enabling routing, load balancing, secure encrypted communications, service discovery, authentication and authorization, monitoring, and resiliency (e.g., Envoy, Linkerd, and Istio).

Capabilities that fall into this category include:

- Service Proxies (e.g., kube-proxy within Kubernetes)
- API Gateways (e.g., Kong and Mulesoft)
- Sidecar and Service Meshes (e.g., Envoy, Linkerd, and Istio)

There is some overlap between these capabilities and those that provide for container orchestration. For example, Kubernetes, a container orchestrator, proxies instances of services to a unified endpoint. Kubernetes also provides a limited form of load balancing.

The use of a service mesh ensures end-to-end encryption of messaging between services and makes it easier to implement zero trust security. Service mesh implementations may be implemented using a sidecar pattern, in which a lightweight proxy is included in the base container images within which all services run. The sidecar mediates service invocations and can enforce authentication and authorization and provide a control plane for management. That can help prevent unauthorized exfiltration of data from a service and provides for a greater level of control for the authentication and authorization for use of services. A service mesh can provide greater levels of observability and analysis since information is easier to collect about the performance of individual instances of a service, and not information aggregated from all instances of a service. A service mesh can provide for better fault handling if a service fails to return results promptly.

API Gateways provide a front end that exposes the API of a NAS service without exposing the implementation details such as the actual service that implements the API. API Gateways also provide management and monitoring and can support enterprise functions such as authentication and authorization.

The use of a Service Mesh and API Gateways is the preferred pattern in the Reference Architecture.

Service Meshes should be used to mediate and interconnect all the microservices within the bounded scope of a development effort.

API Gateways should be used to manage the microservices that are exposed by a development team for use elsewhere in the NAS, or for use by external entities.

4.2.1.5 Virtual Networking, Policy, Authentication, and Authorization

These services connect with other services and enforce access policies. Virtual networking moves the logical control of routing and traffic out of the physical forwarding devices to a dynamically updatable control plane. This facilitates ZTAs by allowing all network traffic to by default deny and whitelist flows according to authorization policy. These services also apply to orchestration systems such as Kubernetes as *network plugins* or Container Network Interfaces (CNI) that provide the security functionality. Examples include Calico, Flannel, Weave and Canal.

Authentication services establish the identity of a person or software process, and authorization services regulate access between authenticated entities. In a mature ZTA, the policy stance should by default deny access between all entities with explicit access granted based on need. Some CNI products provide these services, other examples include Amazon Web Services (AWS) Identity and Access Management (IAM), Identity Management (IdM), Istio, and Active Directory.

the Reference Architecture includes network segregation provided within the Computing Resources Layer (e.g., physical or software-defined subnets and virtual private cloud subnets), as well as virtual networking provided by the Platform Layer.

The Platform Layer provides standardized authentication and authorization mechanisms that are granular and dynamically updateable. Those mechanisms will allow the FAA to reconfigure the network efficiently and dynamically and enforce information security policies. That will facilitate transition to zero trust by allowing network micro-segmentation to prevent the lateral spread of security threats.

4.2.1.6 Distributed Database and Storage

These capabilities allow information to be persisted in a resilient and scalable manner, using structured and unstructured databases, file systems, and block storage (e.g., Vitess and Rook).

The primary use of databases in the Reference Architecture is providing persistent information for a service, allowing the service to be stateless.¹³

The Reference Architecture also uses databases to store information for auditing and later analysis. An example would be recording information for future event reconstruction (e.g., to support accident investigations). Another use is to support playback for simulation/testing.

An open issue with this Reference Architecture is the degree to which other uses might be made of such databases. For example, it might eventually be desirable to do some analytics on

¹³ A stateless service is one where any instance of that service can handle a service call or message, with no dependence on that service having handled any previous call. Any state information required by the service is either kept in a persistent data store (such as a database) or provided in the service call. There are many advantages to having stateless services including allowing load balancing of service calls across multiple instances of a service and the ability to easily restart service instances that fail. There may be specific use cases that require stateful services, but the recommendation is that stateful services should be avoided if possible.

information stored in the databases and make use of those analytics in the NAS operational environment within decision support tools.

The Reference Architecture provides a suite of tools to support relational, document based, and other common database types.

NAS mission services should be kept stateless by using Platform Layer databases and other storage services to provide persistence.

The exchange of information between different NAS mission services should use API calls or information exchanges via a message bus instead of communicating through a database. The reason for this is to avoid strong coupling between the services.

4.2.1.7 Streaming and Messaging (Message Bus)

These capabilities provide asynchronous message-based information exchange, including publish/subscribe and point-to-point message exchanges¹⁴. This document uses “Message Bus” as a generic term to include various technologies that provide streaming and messaging capabilities. (Examples include NATS, Kafka, and AMQP.)

Those mechanisms are often preferable to point-to-point communications because they enable loose coupling between services (a service does not need to know its users, and a subscriber does not need to know who publishes the information). Publish/Subscribe is also much simpler than point-to-point communications in an environment where multiple instances of a service are used to provide performance at scale, because individual messages can be sent to any service instance for processing. The asynchronous nature of messaging also allows responsive processing in an event-driven environment.

This category of capabilities also includes tools to translate between message types. For example, it may be desirable to translate an incoming message in Extensible Markup Language (XML) into JavaScript Object Notation (JSON) before putting it on the message bus.

The Reference Architecture platform includes tools that can be used for messaging/streaming within individual programs/application domains, as well as the NAS-wide message bus (SWIM).

The publishing of information to topics on a message bus is a fundamental part of the Reference Architecture. The NAS can use multiple message broker technologies. Long-haul cross-nation message broker topologies can ensure information can reliably and efficiently be delivered to geographically dispersed subscribers. Within a computing environment, microservice instances can interact in an asynchronous, decoupled manner via a highly reliable and scalable local broker technology (e.g., Kafka). Furthermore, with potential for increased numbers of sensors and other Internet of Things (IoT) devices, a highly efficient, fast broker technology (e.g., Message Queuing Telemetry Transport [MQTT]) is an important addition to the NAS architecture. Message brokers can serve as an important alternative to communicating information between services using databases. As mentioned in Section 4.2.1.2, that can also serve as important means of choreographing the use of services to complete larger functions.

The Platform Layer provides message bus tools that should be instantiated and used by development teams for internal message-based communications among Mission Layer microservices and applications. The Platform Layer will also instantiate a NAS-wide message

¹⁴ The NAS already uses Streaming and Messaging. For example, publishing and subscribing to topics on a message bus is used with SWIM.

bus for message-based communications among applications and microservices developed by disparate teams.

Information should be published to a topic on the message bus if that information is likely to be of interest to multiple services.

4.2.1.8 Analytics and Artificial Intelligence

These capabilities consist of data analysis tools. That includes big data analysis tools, as well as Extract, Transform, and Load (ETL) capabilities. The capabilities are used to find significant patterns within data sources. They are also used to report on those patterns and signal events that may be actionable.

Artificial Intelligence capabilities are becoming increasingly important for identifying patterns within data. As such capabilities become more mature, it is likely they will start to see some usage within the NAS. That will probably start with post analysis capabilities, but eventually may be used in decision support tools as well.

Analytics consist of two forms, batch processing and stream processing, as shown in Table 4-1. With batch processing, a large volume of data (e.g., data lake) is collected over time and is processed all at once. There is a delay between collecting and storing the data sets and the processing for analysis, and reporting. With stream processing, data that is continuously being ingested is processed piece-by-piece. Analysis and reporting for events of interest occur in near-real-time [18].

Table 4-1. Batch Versus Stream Processing

Batch Processing	Stream Processing
Data is collected over time	Data streams continuously
Once data is collected, it is analyzed	Data is processed piece-by-piece
Batch processing is time-consuming and is meant for large quantities of information that is not time-sensitive	Stream processing is fast and is meant for information that is needed immediately

These tools are expected to be used primarily in the mission support environment (e.g., EIM Platform) rather than within the operational NAS. However, in the future these tools may be incorporated in decision support tools and operational dashboards used in near-real time in the operational environment. These tools are also expected to be used for analyzing the NAS platform and mission services, including analyzing performance and usage patterns, as well as detecting security threats.

The Platform Layer provides Analysis tools that should be used to support batch processing as well as stream processing.

4.2.2 Development

This section addresses the non-runtime aspects of creating executable software. The development Platform Layer elements together provide the Software factory that supports Agile and DevSecOps methodologies.

4.2.2.1 Development Frameworks and Libraries

A development framework is a platform for developing software applications/services. It provides a foundation on which software developers can build programs. Development frameworks exist for various software contexts. For example, frameworks such as React and Angular, provide both an architecture model and implementation library for development of user interfaces. Similarly, server-side development frameworks are software models that make it easier to write, maintain and scale software components. They provide tools and libraries that simplify common development tasks, including routing Uniform Resource Locators (URLs) to appropriate handlers (e.g., Tomcat, Node.js, and NGINX), interacting with databases, supporting sessions and user authorization, formatting output (e.g., Hypertext Markup Language [HTML], JSON, and XML), and improving security against attacks. By constraining the developer to a common software model for performing a common task, frameworks provide a consistent approach to developing reusable, robust software functionality.

The Reference Architecture Platform Layer provides an extensive but controlled set of frameworks, libraries, and so on. Those components will need to be vetted for security supply chain concerns, licensing costs, and so on, while providing an extensive enough suite of tools to meet developer needs and maximize productivity.

4.2.2.2 Planning and Requirements Management

Requirements management is the process of documenting, analyzing, tracing, prioritizing, and agreeing on requirements and then controlling change and communicating to relevant stakeholders. A goal of the Automation Evolution is to support Agile software development, which discourages documentation of extensive detailed requirements early in the process. Nevertheless, Agile development teams still need to define and communicate and manage the functionality that is needed and planned for development. In Agile methodologies that is done using techniques such as *Epics* and *User Stories* and *Backlogs*. The following paragraphs include tools that support Planning and Requirements Management Agile practices.

As a principal capability of requirements management software, requirements traceability documents the life of a requirement. Software engineering practices for some categories of NAS software (e.g., safety-critical components) mandate that it should be possible to trace back to the origin of each requirement and every change made to the requirement. Even the use of the requirement after the implemented features have been deployed and used should be traceable.

In addition, requirements management software can support the reuse of requirements across projects, and perform requirements change impact analysis to aid software development planning and change management. Bug and issue tracking are also generally included in requirements management software.

Examples of modern requirements management tools include Jama Software, International Business Machines Corporation (IBM) Engineering Requirements Management Dynamic Object-Oriented Requirements (DOORS) Next, Atlassian's Jira, and Confluence.

To aid in the design of system and software architectures, Model Based Systems Engineering (MBSE) and advanced visualization techniques are emerging as essential tools to analyze and understand this complexity and develop solutions, including system requirements, architecture, design, interfaces, and behaviors.

Examples of modern MBSE tools include IBM Rational Rhapsody and MagicDraw/Cameo.

As with frameworks and libraries, the Reference Architecture Platform Layer provides an extensive but vetted and controlled suite of these tools to meet developer needs and maximize productivity.

4.2.2.3 Software Packaging, Repositories, and Distribution

Software packaging involves the preparation of standard, structured software artifacts targeted for automated deployment. By streamlining software configuration and deployment, software packaging can help reduce application management costs.

Container images and packages are two forms of distributing software. Containerization is a technology that packages software and all dependencies except the operating system into an easily deployed unit that will run reliably across computing environments (e.g., Docker). Packages are bundles of files that are installed by a package manager such as Red Hat Package Manager (RPM) in Red Hat Enterprise Linux (RHEL) or Advanced Package Tool (APT) in Ubuntu, which check to make sure that multiple packages use compatible libraries, do not use the same filenames, etc., before writing the files into one shared filesystem.

Packages usually come from repositories, and it is up to configuration management staff running the repositories to decide which package contains files, network port configurations, system user ID, etc., as well as which versions of programs get packaged. Packages are built from specification files that list which files should be included. Package installation technologies such as RPM and APT-get take the prepared software packages, gather, update, and install required software dependencies, and deploy the software to the target execution environment.

Software repositories also provide storage and management for various forms of software development artifacts (e.g., docker images, VM images, source code, and configuration files). Some repositories serve as (binary) artifact repositories (e.g., Artifactory and Nexus) while others (e.g., Git and Subversion) also provide version control capability at the source code level.

Automation software (e.g., Jenkins and Drone) manages and controls software delivery processes throughout the entire lifecycle, including build, document, test, package, stage, deployment, and static code analysis. The NAS will leverage such automation server technology to automate tasks related to the building, testing, and distribution or deploying of NAS software components.

Container images are the preferred form of software packaging in the Reference Architecture. The Reference Architecture Platform Layer includes software repositories, provided as an enterprise resource, containing hardened, approved software components, to be drawn upon by the Continuous Integration/Continuous Delivery (CI/CD) toolchains, as described in Section 4.2.2.5.

4.2.2.4 Application Programming Interface and Data Management

API Management includes the specification of APIs, shared data formats, schemas, and semantics. API management includes defining, controlling, and communicating the information needed to enable unrelated objects (including systems, services, equipment, software, and data) to interoperate. All those interfaces must be defined and controlled in a way that enables efficient discovery, use and change management of these systems or services. Therefore, the practice of interface management begins at design and continues through operations and maintenance.

Since a principal NAS component is its messaging architecture, the design of the domain dataspace and its associated data models is critical to its success. Because of that, tools to manage message schemas are also important. With a multitude of NAS producers and

consumers, agreement on preferred data models can be challenging. The NAS currently shares data in formats defined via XML schemas, however the Reference Architecture will include support for additional formats such as JSON and Avro which can improve efficiency and facilitate schema management.

NAS applications can declare and publish their APIs for other applications and/or external entities to discover and consume. These APIs completely define a protocol for communication and include data formats allowing for easier and more efficient consumption.

The Reference Architecture platform provides an enterprise registry that defines APIs and message schemas, together with tools for developers to update definitions (preferably automatically), with support for version control and discovery. Service APIs and message schemas that are exposed across the enterprise need to be discoverable and described in an enterprise level registry, whereas APIs and message schemas used within an application domain should be in local registries available to that development team.

4.2.2.5 Continuous Integration / Continuous Delivery Toolchain

CI/CD embody a culture, set of operating principles, and collection of practices that enable application development teams to deliver code changes more frequently and reliably. The implementation is also known as the CI/CD Toolchain.

Continuous Integration is a set of practices that encourages developers to check in the changes to source code frequently. Those changes are then integrated with the changes made by other developers. The result is integration problems are found and resolved much earlier in the process as opposed to when developers wait until their portion of the code is complete and then try to integrate their code with code produced by others. Continuous Delivery automates the delivery of applications to the infrastructure that will be used to run them. That allows for more frequent updates to applications including both bug fixes and new content. An example of a tool used for CI/CD is Drone. It is an open-source continuous integration server and is used to build and test software projects continuously.

CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of applications, from integration and testing phases to delivery and deployment. Inclusion of comprehensive testing and security scanning within the CI/CD pipeline is essential for ensuring software quality. Automated testing increases speed of innovation since developers can refactor code and immediately know if they have broken something. Code is deployed automatically and more frequently, with higher code quality and improved operations. Irrespective of system/software architecture, CI/CD is integral to modern software development practices.

The Platform Layer provides CI/CD toolchains that should be utilized by all mission software development teams. The CI/CD toolchains implement a software factory that automates configuration and artifact management, testing (including security), integration, the creation of hardened runtime artifacts (such as containers), and delivery of those hardened artifacts into both test and production environments. There will be an approved suite of tools that make up the software factory. However, differences in programming languages and development frameworks will require multiple CI/CD toolchains and development teams will need to work with the Platform Layer provider to ensure the provided toolchains meet their needs. Projects may have their own instances of the CI/CD toolchain and development environment.

4.2.3 Summary of Platform Elements in the Reference Architecture

Table 4-2 summarizes what the Reference Architecture provides for each of the Platform Layer elements.

Table 4-2. Platform Elements in the Reference Architecture

Platform Element	Service-Based Reference Architecture for NAS Automation
Runtime	
Software Hosting/Execution	Containers and container orchestration are a fundamental part of the Reference Architecture. Other approaches (e.g., VMs, Serverless Computing) are discouraged, but may be supported when needed.
Workflow Choreography and Orchestration	Use in limited contexts where valuable, however avoid centralizing enterprise mission logic in one place using these tools. Reference Architecture prefers event-driven processing and choreography patterns.
Monitoring and Log Analysis	The Reference Architecture will include a robust suite of logging and monitoring tools and services, to support both operational performance and availability management, as well as cybersecurity monitoring and response.
Service Proxy, Mesh, and Gateway	Service Mesh is the preferred pattern for Reference Architecture (provides monitoring of service instances, supports zero trust) within an application domain, with API gateways and enterprise messaging for interdomain interoperability.
Virtual Networking, Policy, Authentication and Authorization	The Reference Architecture includes network segregation provided within the Computing Resources Layer (e.g., physical or software-defined subnets and virtual private cloud subnets), as well as virtual networking and authentication and authorization mechanisms provided by the Platform Layer.
Distributed Database and Storage	The Reference Architecture platform includes an extensive suite of tools and service for persistence. The preferred pattern is use of persistence within instance of services (allowing service implementation to be stateless) rather than use of databases for communication among different services.
Streaming and Messaging	The Reference Architecture platform includes tools that can be used for messaging/streaming within individual programs/application domains, as well as the NAS-wide message bus (e.g., SWIM).
Analytics and Artificial Intelligence	Tools for analytics and AI are included in the Reference Architecture platform. Batch analysis tools are expected to be used primarily in the mission support environment (e.g., EIM Platform) but stream analysis tools may be used in near-real time in the operational environment.

Platform Element	Service-Based Reference Architecture for NAS Automation
Development	
Development Frameworks and Libraries	The Reference Architecture will allow developers to use a wide variety of frameworks, libraries, and languages to meet developers' needs. Developers will select from an extensive but controlled set of hardened, configuration-controlled artifacts.
Planning and Requirements Management	The Reference Architecture will include tools for requirements management, bug trackers, feature requests, tracking user stories, documentation, and so on.
Software Packaging, Repositories, and Distribution	Reference Architecture platform will provide enterprise repository(s) of hardened container images. Toolchains will ensure that only hardened artifacts are used.
API and Data Management	The Reference Architecture platform will provide an enterprise registry that defines APIs and message schemas, with support for automated updates, version control and discovery.
CI/CD Toolchain	All programs should use a CI/CD toolchain. The Reference Architecture platform will provide an extensive suite of CI/CD tools providing one or more software factories to meet developers' needs.

4.3 Survey of Existing Platforms

We surveyed several programs that have been applying Agile Development, DevSecOps, and/or Service-Based or Microservices architecture. Appendix B contains our survey results for:

- ABCD
- CLMRS
- EIM Platform
- Project Elroy
- DoD Platform One

For each of the above, a brief description is provided, as well as a table showing what the program is using/providing for the Platform Layer, organized according to the Platform Layer Elements breakdown in the previous section.

5 Computing Resources Layer

5.1 Overview

The Computing Resources Layer, depicted in Figure 5-1, includes end-user equipment, compute infrastructure, and local area networks (LAN) and wide area networks (WAN) that tie them together. Examples of end-user equipment would be workstations, displays, tablets, mice, trackballs, and keyboards. Examples of compute infrastructure would be servers that run services, operating systems, storage, VMs, and containers.

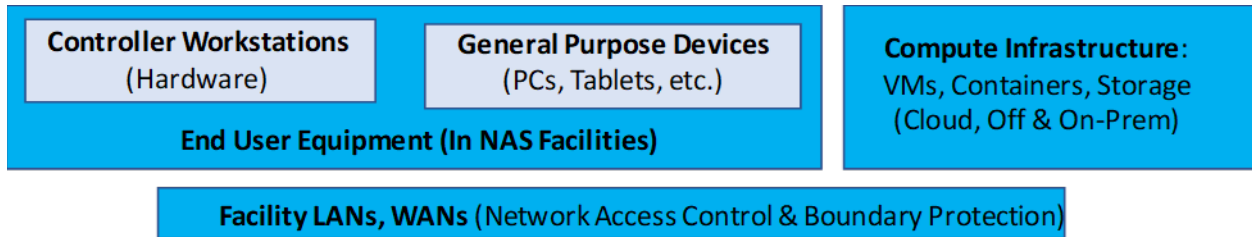


Figure 5-1. Computing Resources Layer

The Computing Resources Layer has the following key characteristics:

- **Secure:** Meeting security standards allowing NAS ATO
- **Available:** Providing redundant processing in separate locations (e.g., AWS Availability Zones)
- **Reliable:** Supporting continuity of operations (e.g., multiple cloud regions and multiple cloud providers)
- **Responsive:** Low latency and high data rate connectivity (e.g., direct connections to cloud environments from multiple NAS locations)
- **Scalable:** Able to expand or contract to continue to meet performance requirements as demand varies
- **Supportable:** Providing a support model that folds into FAA TechOps processes
- **Trusted:** Showing users and Operators the benefits of cloud and demonstrating cloud as a viable platform for the future.

The elements that make up the Computing Resources Layer are a combination of on-premises as well as off-premises resources, as illustrated in Figure 5-2. These elements are described in the following sections.

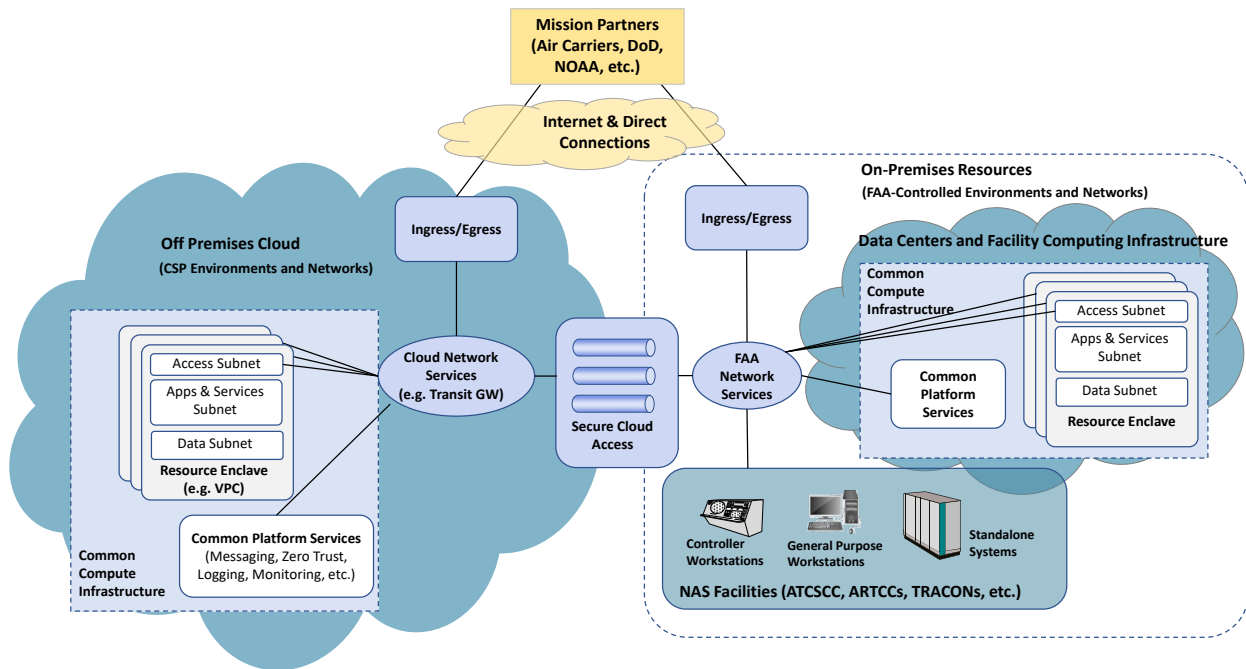


Figure 5-2. Computing Resources Elements

The common computing infrastructure is provided primarily by CSPs off-premises, as illustrated on the left side of Figure 5-2. Within these environments, resource enclaves (e.g., VPCs) are provisioned to meet the needs of programs for development, testing, staging, and operations. Platform Layer elements are instantiated within enclaves and as common services supporting all programs, as was described in Section 4. Off-premises cloud is described in Section 5.2.1.

If necessary, to meet business needs or NAS requirements (e.g., security, performance, or latency requirements), NAS common computing infrastructure can also be provisioned in on-premises data centers or facility-based computing infrastructure, as illustrated on the right side of the figure.¹⁵ Resource enclaves and Platform Layer services can then be instantiated on this infrastructure, masking computing infrastructure differences from the Mission Layer software developers. On-premises data center and facility-based infrastructure is discussed in Section 5.2.2.

While the goal is that new development will be done using the common Reference Architecture Platform Layer (Section 4) hosted on common computing infrastructure, we recognize that standalone systems such as legacy systems, sensors, and other physical equipment collocated with NAS facilities will continue to exist. Standalone systems are discussed in Section 5.3.

End user equipment (controller workstations and general-purpose workstations, mobile devices, and so on) must also be provided and sustained to support human users within NAS facilities, as discussed in Section 5.4.

Networking capabilities shown in Figure 5-2, include: NAS wide-area telecommunications (e.g., FENS), virtual networking services within the cloud, secure connectivity between the FAA

¹⁵ It remains to be seen whether on-premises data centers and facility-based compute infrastructure will be needed in the long term. Data centers providing common compute infrastructure exist in the NAS today (the Integrated Enterprise Services Platform), and back-room facility servers providing IaaS may be used as an interim step to cloud-based computing for systems such as the Standard Terminal Automation Replacement System (STARS).

premises and the cloud, and ingress and egress connectivity to mission partners. These are discussed further in Section 5.5.

This evolving NAS Computing Resources Layer demands the integration of monitoring and control strategies for diverse computing resources and services. For continuing on-premises standalone systems, monitoring and control functions will be driven by the specific operational needs of those systems, augmented with interfaces to common NAS monitoring and control functions to enable determination of NAS-wide computing resource status. Off-premises cloud computing resources will involve monitoring and control capabilities integrated with those services and aligned with the Platform services described in Section 4. When the off-premises computing involves a non-FAA service provider, expectations will be captured in service level agreements. On-premises data center and facility computing will replicate aspects of the cloud computing and Platform services with correspondingly similar monitor and control functions.

From the operator perspective, the most significant difference is the redistribution of monitoring and control responsibility and authority. Operators monitoring and controlling Platform Layer enterprise managed services and their associated cloud computing resources will have much broader influence on NAS operations. Accordingly, these operators must have a correspondingly broader perspective of the operational context for these services as well as the Platform services they all share. At the facility level, operators will monitor and control the interfaces with Platform Layer enterprise managed services and computing resources, the configuration of the local computing environment, and the configuration and status of local applications and services.

5.2 Common Compute Infrastructure

The common compute infrastructure provides the underlying servers and storage needed by the NAS Mission Services and Applications. The actual physical equipment is expected to primarily reside in off-premises facilities run by CSPs but may also reside in on-premises data centers or facility-based server rooms. In either case, those physical resources are made available as VMs and virtual storage devices using an IaaS model. In the case of on-premises data centers, hyperconverged infrastructure (HCI) can be used to unify management of system resources consisting of virtualized servers, storage, computing, and networking. HCI can be used to replace legacy infrastructure. Benefits of HCIs include increased scalability and reduced datacenter complexity, rapid deployment, and architecture flexibility. HCI can be used to build a private cloud, expand to a public cloud, and build hybrid clouds (a mix of VMs and containers) [19], [20].

The Platform Layer that runs on top of this infrastructure is provided by a CSP with a PaaS model or created by installing and configuring Platform Layer software components on the virtual infrastructure. The Platform Layer abstracts the Computing Resources Layer, so developers are provided a common environment regardless of how the compute infrastructure is hosted.

Because the Platform Layer is specifically designed to be cloud ready and usable on a wide variety of computing infrastructure resources, the actual makeup of the compute infrastructure matters little to the developer and end user of a system. This has some very practical benefits. For example, a program might stand up a development environment using an off-premises CSP for easy access and the ability to adapt rapidly to changing needs, while the production environment might be hosted on an on-premises data center. Or, one developer might stand up a development environment on their own VMs, while other developers use one created using an off-premises CSP.

The on-demand delivery of infrastructure is a key advantage of using CSPs, making it possible to scale the infrastructure to the changing demands on a program. For example, there may be less need for infrastructure when first standing up a program. As the program gets rolled out for use, there may be a much higher demand on the compute infrastructure, and that demand might vary seasonally or with special events. Using a CSP allows paying for what you need when you need it.

FAA use of off-premises cloud provided by a CSP such as AWS or Azure is already well-established for administrative systems (e.g., email), and the FAA is also beginning to use off-premises cloud under the FCS contract for Mission Support capabilities and for programs that have a public facing component (e.g., SWIM Cloud Distribution Service, and the NAS Status web pages). As the FAA gains experience working with off-premises cloud (e.g., security, monitoring and control, performance), we expect that deployment to off-premises cloud will begin to be used for NAS decision support and ATC systems. Off-premises deployment is discussed in Section 5.2.1.

However, especially in the near term, many programs will want the advantages of deploying to the cloud but for a variety of reasons will require computing infrastructure in FAA facilities, directly connected to NAS networks, power, and so on. Detail about such on-premises deployments can be found in Section 5.2.2.

5.2.1 Off-Premises Cloud

The off-premises resources (see Figure 5-2) are provided and sustained by CSPs and provisioned for the FAA using the FCS contract.¹⁶ Resource enclaves are provisioned as needed to meet the needs of FAA programs or development teams within FAA programs, providing computing environments for development, research and concept evaluation, testing, staging, and production. In AWS, for example, resource enclaves are planned to be created as VPCs following a standard FAA template, whereas in Azure, Virtual Networks (VNETs) would be used. For each FAA program or development team within an FAA program, FAA templates would exist for all stages of the engineering pipeline (i.e., development, test, integration, production. As seen in Figure 5-2, the enclave templates would include distinct sections/subnets for ingress/egress services (Access Subnet), mission and platform services (Apps and Services Subnet), and a persistent data storage subnet (Data Subnet).¹⁷ For example, an Access Subnet would host Platform Layer elements such as API gateways and Web Application Firewalls (WAFs) that expose common services. An Apps and Services subnet would host the platform elements such as container orchestration (Section 4.2.1.1), which would run containers containing instances of NAS Mission Applications and Mission Services. The Apps and Services subnet would also host Platform Layer elements such as service mesh (Section 4.2.1.4) and local message brokers. A Data Subnet would provide persistent storage for state information (Section 4.2.1.6). Finally, in addition to platform elements instantiated within resource enclaves, Common Platform Services support the entire FAA cloud environment. Examples of Common Platform Services include Streaming and Messaging (Section 4.2.1.7), Monitoring and Logging (Section 4.2.1.3), Domain Name Service (DNS) and Zero Trust security services (Section 4.2.1.5).

¹⁶ The National Cloud Integration Services (NCIS) program is in the process of providing a secure cloud environment. Much of the information in this section is based on the NCIS “Bespin” draft design materials.

¹⁷ This data subnet is not the same as the NAS data plane. The data plane in this architecture manages service invocations and event driven information exchanges using technologies such as API gateways and proxies, message buses, and service meshes. The data subnet is an enclave within the cloud environment that isolates the long-term storage from other parts of the system.

The use of an off-premises cloud is not limited to a single cloud vendor. A multi-cloud strategy is where multiple cloud environments (e.g., AWS and Azure) can be utilized in concert. There is a growing trend for organizations to adopt a multi-cloud strategy. Technologies exist (e.g., service mesh technology) that can serve as a catalyst for successful multi-cloud strategies. The following are a few advantages and some of the most common reasons organizations adopt multi-cloud:

- Risk Mitigation – create resilient architectures
- Managing vendor lock-in
- Workload Optimization – placing workloads to optimize for cost and performance
- Cloud providers’ unique capabilities – take advantage of offerings in Artificial Intelligence (AI), IoT, Machine Learning (ML), etc.

The challenges associated with transitioning to a multi-cloud architecture include:

- Determining the right cloud service for the job at hand
- Fitting the pieces together – each CSP generally has different APIs for similar services
- Managing costs in a complex environment
- Ensuring data protection and privacy
- Ensuring data consistency and integrity
- Keeping up with the rate of change

An overall multi-cloud strategy requires a thoughtful assessment about which cloud attributes best serve the FAA’s specific needs. Avoiding being locked-in to a specific CSP vendor is also an important consideration as discussed in Section 4.1.

5.2.2 Data Centers and Facility-Based Computing Infrastructure

Within Data Centers such as the NAS Enterprise Management Center (NEMC), where the IESP is located, resource enclaves can be provided with the same structure as those in the off-premises cloud environment. Platform services such as DNS, SWIM messaging, and so on can also be provided in data centers, or as needed in NAS facilities.

Currently, the IESP provides IaaS in the form of VMs and storage, using hardware installed and maintained at two NEMC locations, Atlanta and Salt Lake. The IESP could be enhanced to provide a NAS Platform Layer in these data centers. This would include functions such as infrastructure as code, container orchestration, a CI/CD DevSecOps pipeline, dynamic workloads (as services request additional resources in real time), multitenancy, separated control plane and service mesh, stateless workloads and ephemeral nodes, and availability through zones and regions. We expect application developers will want “Cloud APIs” (e.g., S3 buckets with associated authentication/auditing/monitoring).

Support for NAS automation evolution in the long term would also require that the IESP be able to scale to support NAS-wide performance requirements for mission non-critical and critical services in terms of latency, jitter, bandwidth, and compute capacity.

Racks of servers could also be provided in facility back rooms at Air Route Traffic Control Centers (ARTCCs), Terminal Radar Approach Control Facilities (TRACONs), or even Towers,

to provide facility-based common computing infrastructure. Virtualization software and NAS Platform Layer software components would then need to be installed, configured, and maintained on this infrastructure. That would allow NAS Mission Services and Applications to run locally in facilities, close to users. Whether this is necessary or cost-effective in the long term is not clear at this point. To provide platform services the back room would need to be able to provide the types of evolutions of IESP as described above and be able to do so in a seamless enterprise-wide manner.

It is also possible to purchase and install products from CSPs that create a fully managed CSP cloud on customer premises. Examples include AWS Outposts or Azure Stack. Those offerings, if used, would make CSP APIs and services available within FAA data centers or local facilities. Though the integration with and access to cloud-scale and cloud-API would be relatively seamless, along with any advantage to be gained by local network connectivity, the need for these solutions, cost, and other implications, would have to be determined.

5.3 Standalone Systems

Standalone systems include legacy systems that have not been migrated to cloud, as well as equipment that cannot be virtualized, such as sensors or equipment requiring dedicated physical interfaces (e.g., serial circuits) that are not supported in the cloud. Although these systems may not be migrated to the cloud and/or to on-premises data centers, data collection will still be required to support cyber monitoring and security investigations, performance monitoring, historical trend analyses, and other mission support functions.

Some systems that require extremely low latency, high availability, and or high levels of assurance may also be implemented as standalone systems. Examples might include landing systems, or any other system that directly affects aircraft flight.

5.4 End-User Equipment

Because of the large numbers of operational positions to support, end-user equipment represents the largest number of hardware components to be acquired and maintained. Choices made in this area will have a big impact on future operating and sustainment costs. The Reference Architecture promotes the commonality of end-user equipment enabling greater flexibility in operations and lower costs through commoditization and economy of scale in purchasing.

User equipment will reflect one of two existing themes: high-powered workstations and displays that meet safety-critical performance requirements and web/browser-based approaches for non-safety critical operations. The former involves performance demands, complexity, and domain-specific aspects that are associated with higher costs and reduced flexibility. The ubiquity of the latter and the accompanying multitude of developers and development assets are associated with lower costs, greater productivity, and more rapid development. Unless specific demands require the former, web/browser-based user interfaces will be preferred to lower the cost and leverage abundant web-based development tools and utilities.

5.4.1 Safety-Critical User Interfaces

Controller positions will use a bespoke configuration of hardware and software that meet the highest demands for availability, performance, and integrity. At the lowest level is the workstation hardware (processors, memory, interfaces) and interfaces with the system network and with user interface devices. User input mechanisms include the display, keyboard, and

trackball. In the future new user interface (UI) technologies such as speech recognition, augmented reality, or gesture controls may be added. Workstation operations are controlled by the operating system. The operating system will be a standard adopted for the NAS (currently RHEL). The operating system may be tailored to support higher levels of software design assurance associated with safety-critical operations. The workstation hosts the ATC UI application(s) and other software components for system functions such as system management and system recording, prospectively proxies for platform services. Currently ATC interface applications are large, complex, monolithic applications. For the future architecture, it is desirable to follow the service-oriented design philosophy by implementing an integrated package of user interface application components.

5.4.2 Web Browser-Based User Interfaces

When supported operations are not safety-critical, user interfaces will be standard web/browser-based designs. The utility of web/browser-based user interfaces continues to grow, enabled by the progress of browser technologies such as WebAssembly and Web Real-Time Communication (WebRTC), that expand the application of web/browser-based UI to ATM operations.

5.5 Networking

FAA Network Services provide connectivity on premises. That connectivity will be provided by FENS, which is currently in the vendor selection phase of acquisition. The FENS solicitation calls for network connectivity within Critical, Essential, and Routine network domains, as well as Edge Protection (EP) services for data flows among these domains. Mapping those network domains and communications services to the Reference Architecture remains to be completed.

Cloud networking services make use of CSP network services and infrastructure to provide connectivity between FAA elements in a controlled manner. For example, NCIS BESPIN will provision and manage an AWS Transit Gateway that allows the FAA to provide and control connectivity among resource enclaves.

Cloud Access connects the FAA's on-premises network and the FAA's cloud environment using, for example, AWS Direct Connect or Azure ExpressRoute services, combined with network layer protection and monitoring. Cloud Access will be instantiated at as many NAS locations as needed for performance and availability.

Ingress/Egress provides access to mission partners (e.g., air carriers) via the Internet. Ingress/Egress services from the on-premises environment. This is currently provided by the NESG and other components that provide boundary protection and connectivity to mission partners via the FAA's Internet Access Gateways (IAGs). The IAGs were built by the FAA in accordance with Trusted Internet Connection 2.0 (TIC 2.0) guidance from the U.S Department of Homeland Security (DHS).

Ingress/Egress in the cloud environment will be provided by cloud-based services at the internet edge that provide protections equivalent to the NESG and the IAG. Cloud-based Ingress/Egress services allow NAS services to be securely exposed to mission partners via the internet,¹⁸ and to

¹⁸ Partners requiring high availability connectivity to FAA cloud-based services can configure their own direct access to the CSP environments as needed. Direct connections between the FAA and trusted government partners (e.g., DoD) are also accommodated, but not discussed further here.

Service-Based Reference Architecture for NAS Automation - Version 1.2

allow NAS applications to access external resources. Ingress/Egress is protected by security controls consistent with DHS guidance (currently TIC 3.0 but expected to continue to evolve).

6 Surveillance Services Use Case

6.1 Background

6.1.1 Surveillance Use Case Purpose

The Surveillance Services use case informs perspectives regarding how mission services should be defined and how those mission services relate to other Reference Architecture elements. Surveillance is selected as a use case topic because it involves many properties of concern to determining the effective scope and applicability of the Reference Architecture (e.g., safety criticality, demanding performance, and aspects unique to ATC operations). Specific use case objectives include:

- Explore influences on how to define and scope services (e.g., coupling, performance requirements, change isolation, process attributes, and software magnitude).
- Solidify candidate NAS Mission services.
- Identify dependencies among Surveillance Services and other Mission services.
- Lay the groundwork for assessing the feasibility of cloud-based efficiency-critical and safety-critical services.
- Assess the performance needs for surveillance services and associated applications. Determine if cloud services can meet these needs or whether a specific NAS Data Center solution acting like a cloud would be required.
- Clarify expectations of supporting Platform Layer services.
- Identify NAS-unique influences that differentiate the Reference Architecture from more canonical service-based architecture models.

6.1.2 Current Surveillance Data Processing

Surveillance Data Processing (SDP) provides an integrated picture of controlled airspace used primarily for ATC separation. SDP is one of many integrated capabilities provided by a facility's ATC system. Each instance of SDP is customized for a facility's operation. Surveillance inputs include dedicated connections to each radar and to ADS-B and Wide Area Multilateration (WAM) services. Surveillance information is shared with additional NAS applications and external users for processing and display of positional information and trajectory modelling. Multiple post-operational functions use and store position information independently.

Sensor data is supplied to SDP functions using IP, serial communications, or serial over IP communications from sensor point to SDP location based on the SDP's/ATC Facility's geographic area of responsibility. This inflexible design is challenging during facility failure conditions and contingency/resiliency operations. It is costly to maintain circuits to support pre-planned facility reconfigurations during failure conditions, which may or may not adequately address the multitude of failure scenarios that may be encountered.

New sensor types offer improved accuracy and update rates and are provided to ATC separation systems for use. However, downstream applications are limited to the published post-SDP System Track for use at slower update rates than provided by the source. Downstream

application missions may benefit from access to more accurate and more frequently updated surveillance information.

The ATC separation systems are not the authoritative surveillance source. The ATC separation systems are authoritative for pairing sensor track information to a specific flight. Today, sensor data can only be consumed at the whole sensor level as opposed to subscribing to specific tracks.

Today ATC automation systems can process sensor track information in slant range coordinates and in spherical coordinates based on World Geodetic System 1984 (WGS 84). Spherical coordinate data supports the five-mile separation common in En Route operations. In Terminal operations, three-mile separation may be supported by operating in fusion mode (utilizing all available surveillance sources) or by single sensor slant range processing. In En Route operations, three-mile separation is supported by operating using the default track-based display (utilizing only three-mile separation qualified surveillance sources in qualified coverage airspace) which provides the three nautical mile (NM) target symbol. Efforts are ongoing to perfect surveillance data fusion and evolve a more performance-based specification to support certification.

In addition to current SDP, future operational concepts suggest the need to accommodate new surveillance processing requirements. For example, supporting the operations of new entrants such as space and Unmanned Aerial System (UAS) operators may involve new surveillance technologies and new approaches to integrating surveillance for situation awareness. New surveillance approaches may be needed to support operational concepts such as dynamic airspace that offer greater ATC operational flexibility.

The potential development of future mission level Surveillance Service(s) is an opportunity to meet the objectives of 1) relax the constraints of existing SDP while providing performance that continues to meet fundamental safety objectives and enhances NAS operations, 2) provide the means to more easily accommodate new technologies and operations, and 3) provide Surveillance Services in a manner consistent with the Reference Architecture described in Sections 1 through 5.

6.2 Surveillance Services Overview

6.2.1 Surveillance Use Case Scope and Assumptions

For this use case, some simplifying assumptions are made as follows:

- Surveillance sensors are data sources that will be considered operational infrastructure, similar to the communications network. The sensors are outside of the scope of the automation processing that is the focus of the Reference Architecture.
- Distributed surveillance data is IP-based, uses All Purpose Structured Eurocontrol Surveillance Information Exchange (ASTERIX) protocol, and is distributed via a Surveillance Data Network (SDN). The SDN employs mechanisms to prioritize surveillance data distribution and minimize latency in real-time operations (e.g., differentiated network domains). SDN may utilize FENS and other networks (e.g., third party surveillance networks for the UAS/UAS traffic management [UTM] community) for the exchange of surveillance data.

- Surveillance source types considered are radar (cooperative [secondary surveillance radar (SSR)] and non-cooperative [primary surveillance radar (PSR)]), ADS-B, and WAM. Conceptually, a placeholder is included to represent a future new surveillance source.
- Multiple instances of each service are assumed to promote availability and to balance processing loads.

For this use case, the scope of the Surveillance Services is interpreted to be input processing and quality assurance functions associated with surveillance input processing, tracking, and surveillance data distribution. Separation assurance functions (e.g., Conflict Alert, Minimum Safe Altitude Warning, and airspace alerts) are not included among Surveillance Services but are considered mission services that consume surveillance data.

6.2.2 Surveillance Services Context

The Surveillance Services are provided within the context of the layered architecture depicted in Figure 6-1 and described previously in Section 0. Considering the “Back-End” Computing column first, the column represents mission and data services that are hosted in cloud technologies (potentially on- or off-premises) focused on managed container environments representative of the current commercial mainstream. Surveillance Services are a collection of the Mission Services/Data of the figure. Among the Mission Services/Data, Surveillance Services have attributes that include:

- They are safety-critical, with associated requirements and performance sensitivity.
- They exhibit a streaming data mode of operation.
- They are common mission services. Ideally, the scope will be NAS-wide, but might also be partitioned in different ways (e.g., defined geographical regions).

Dependencies among services are explicitly accommodated by mechanisms such as the service mesh described in Section 4.2.1.4. The service mesh may need to accommodate ATC-specific business logic. Additional detail is provided in Section 6.3.1.

Service-Based Reference Architecture for NAS Automation - Version 1.2

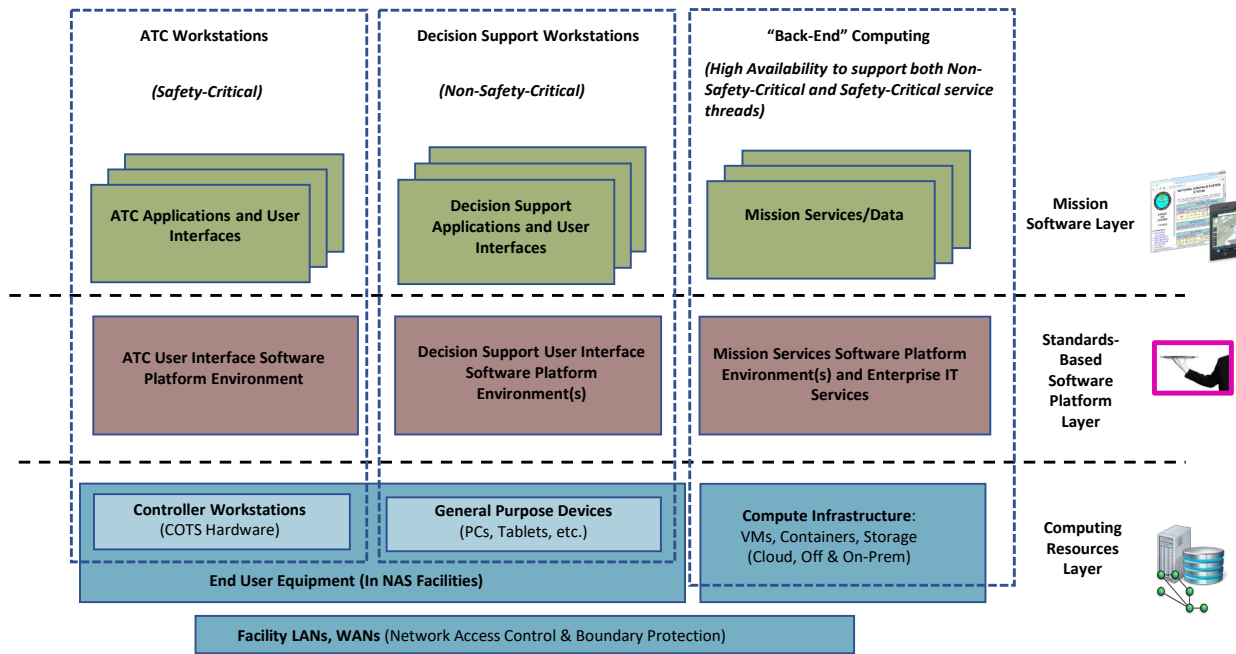


Figure 6-1. Layered Architecture

The Software Platform Layer services must support the robustness, integrity, and performance requirements of safety-critical Mission Services. While many Software Platform services may commonly apply, customization may be needed to support safety-critical operations. Additional detail is provided in Section 6.3.2.

Ideally, the Compute Infrastructure for Surveillance Services will conform to the container environment defined in Section 2.2 and described in Section 4.2.1. Safety-critical requirements drive the need for multiple instantiations of this environment with physical diversity to mitigate potential failure modes. It is anticipated that commercial cloud service providers will be able to meet those needs, but multiple alternatives including on-premises cloud implementations and NAS Data Centers must be considered. See Section 6.3.1.2 for additional detail.

The two leftmost columns of Figure 6-1 represent the coordination and consumption of services that will occur at FAA facilities using on-premises computing resources. ATC and Decision Support user interface components will use processors, displays, and input mechanisms at each controller position, similar to current operations. Differences in design between ATC and Decision Support positions can be expected based on the differing criticalities involved. Ideally, facilities will have a compute infrastructure that mirrors common mission infrastructure (i.e., managed container environment), but on a diminished scale. Application-specific servers and storage may also be provided based on operational need. Additional detail is provided in Section 6.3.

6.3 Surveillance Services Detail

6.3.1 Surveillance Services Mission Layer

A fundamental aspect of the service-based architecture is the determination of services. Principles applied to determine services involve a combination of general principles that apply to any service/microservice approach and domain-specific aspects that reflect the context of the

operations to which the service applies. The following list represents influences in defining NAS common mission surveillance services:

Separation of functions/concerns. A service should be captured by a minimal set of software components that reflect specific functions and use of data.

Minimal coupling. To the extent possible, relationships between services should be minimized and services made as independent as possible.

Criticality. NAS services have performance requirements that reflect the importance of the service to NAS operation and must conform to a DO-278A assurance level that characterizes the scope of configuration management, testing, documentation, and other aspects of development that apply. Current ATC automation systems reflect assurance level (AL) 4, with an AL3 exception for air-ground data communications. Because higher performance requirements and assurance levels involve greater and more costly development efforts, services should be composed in a way that minimize the level of applicable performance requirements and assurance level.

Legacy design experience. A NAS common mission surveillance service must have capabilities that meet or exceed the surveillance capabilities of existing ATC automation systems. These ATC automation systems exhibit common design patterns that already reflect many of these principles and may be leveraged as a basis for surveillance service definition. Surveillance and Broadcast Services (SBS) represents a model of a common mission surveillance service.

Operational scope. NAS operations are differentiated into operational domains (e.g., en route, terminal, surface, and oceanic) that reflect different operational contexts, standards, procedures, and performance needs. While the intention is to define common mission surveillance services that apply to all, it may occur that operational differences warrant associated partitioning of surveillance services to meet domain-specific needs.

Scalability. A desired property of service-based architecture is the ability to respond to operational demands by increasing the instances of a service and balancing the applied load among them. A NAS common mission surveillance service must be able to accommodate the workload represented by the number of sensors, targets, and tracks occurring in operation.

Anticipated change pace. Stability of a service-based architecture is promoted by separating the services that are stable and seldom change from the services that require frequent change. While typically interpreted to refer to functional changes associated with evolving capabilities, NAS operations involve episodic updates (i.e., adaptation) to be accommodated.

Software magnitude. To provide the most effective development, operation, and maintenance, the best size of a service software component involves tradeoffs between the complexity of the service environment to be managed (i.e., the number of services and service dependencies to be coordinated) and the complexity of each individual service software component to be developed and maintained.

Resilience and disaster recovery. The scope and distribution of services influences the resiliency of ATC operations and the potential to reconstitute operations in response to disasters.

Observability. A well-defined service should promote effective monitoring and control of its function, performance, and accessibility within the context of the operational environment.

Service-Based Reference Architecture for NAS Automation - Version 1.2

Considering existing ATC automation designs as a guide, tentative surveillance services of radar processing, ADS-B processing, WAM processing, tracking, and surveillance data distribution are identified. Rationale for these services is summarized in Table 6-1.

Table 6-1. Surveillance Services Rationale

Service Definition Influence	Prospective Common Mission Service				
	Radar	ADS-B	WAM	Tracker	Distribution
Separation of Functions	Common radar interface and protocol (ASTERIX CAT048)	Common interface and protocol (ASTERIX CAT033)	Common interface and protocol (ASTERIX CAT010)	Common interface and protocol (ASTERIX CAT062)	Manages service consumer subscriptions
	<p>Integrated processing of primary and secondary radars</p> <p>Radar specific QA</p> <p>Radar processing could be partitioned into more “micro” services with specific functions such as:</p> <ul style="list-style-type: none"> • sensor status • collimation • registration • CAT048 parsing • performance analysis (e.g., QARS) • other 	<p>ADS-B specific QA</p> <p>ADS-B processing could be partitioned into more “micro” services with specific functions such as:</p> <ul style="list-style-type: none"> • sensor status • CAT033 parsing • performance analysis • other 	<p>WAM-specific QA</p>	<p>Fused track solution integrates sources</p> <p>Tracking services could be differentiated to optimize for source characteristics, but that would complicate what constitutes the “best” or most “authoritative” solution for consumers</p>	<p>Filters streams by subscription attributes and policies (e.g., security, privacy)</p> <p>Distribution services could be differentiated by consumer characteristics (e.g., NAS vs non-NAS, safety, and efficiency-critical vs essential or routine)</p>

Service-Based Reference Architecture for NAS Automation - Version 1.2

Service Definition Influence	Prospective Common Mission Service				
	Radar	ADS-B	WAM	Tracker	Distribution
Minimal Coupling	<p>Coupled to Tracker, and Distribution Services. Coupling is unidirectional – Radar out to other services</p> <p>Coupling with ADS-B may be desired for QA purposes</p>	Coupled to Tracker, and Distribution Services	Coupled to Tracker and Distribution Services	Coupled to Radar, ADS-B, WAM, and Distribution Services	<p>Coupled to Radar, ADS-B and Tracker services.</p> <p>A Distribution service harmonizes access to surveillance information, reducing required service couplings between surveillance service consumers and sources.</p>
Criticality	Critical	Critical	Critical	Critical	Critical
Legacy Design Experience	Integrated radar processing; regional sources and automation-specific processing. Radar specific functional threads and software component(s)	NAS-wide processing (SBS); regional (service volumes) and automation-specific automation processing. In automation, ADS-B specific functional threads and software component(s)	Regional coverage through SBS infrastructure; automation-specific automation processing	Regional, automation-specific processing. In automation, tracking specific functional threads and software component(s)	<p>SBS provides ADS-B distribution by static subscription (SVs)</p> <p>Legacy automation systems provide system-specific surveillance data distribution to external users (e.g., EDDS, STDD, handoffs)</p>

Service-Based Reference Architecture for NAS Automation - Version 1.2

Service Definition Influence	Prospective Common Mission Service				
	Radar	ADS-B	WAM	Tracker	Distribution
Operational Scope	May be NAS-wide, Regional, or Facility-specific based on operational situation and performance	May be NAS-wide, Regional, or Facility-specific based on operational situation and performance	May be NAS-wide, Regional, or Facility-specific based on operational situation and performance	May be NAS-wide, Regional, or Facility-specific based on operational situation and performance	May be NAS-wide, Regional, or Facility-specific based on operational situation and performance
Scalability	Operational demand	Operational demand	Operational demand	Operational demand	Operational demand
Anticipated Change Pace	Low	Low	Low	Low	High
Software Magnitude	TBS	TBS	TBS	TBS	TBS
Resilience	Design dependent	Design dependent	Design dependent	Design dependent	Design dependent
Observability	Service-specific	Service-specific	Service-specific Shared ADS-B infrastructure	Service-specific and integrated service M&C	Service-specific and integrated service M&C

Additional context for the identified Surveillance Services is depicted in Figure 6-2.

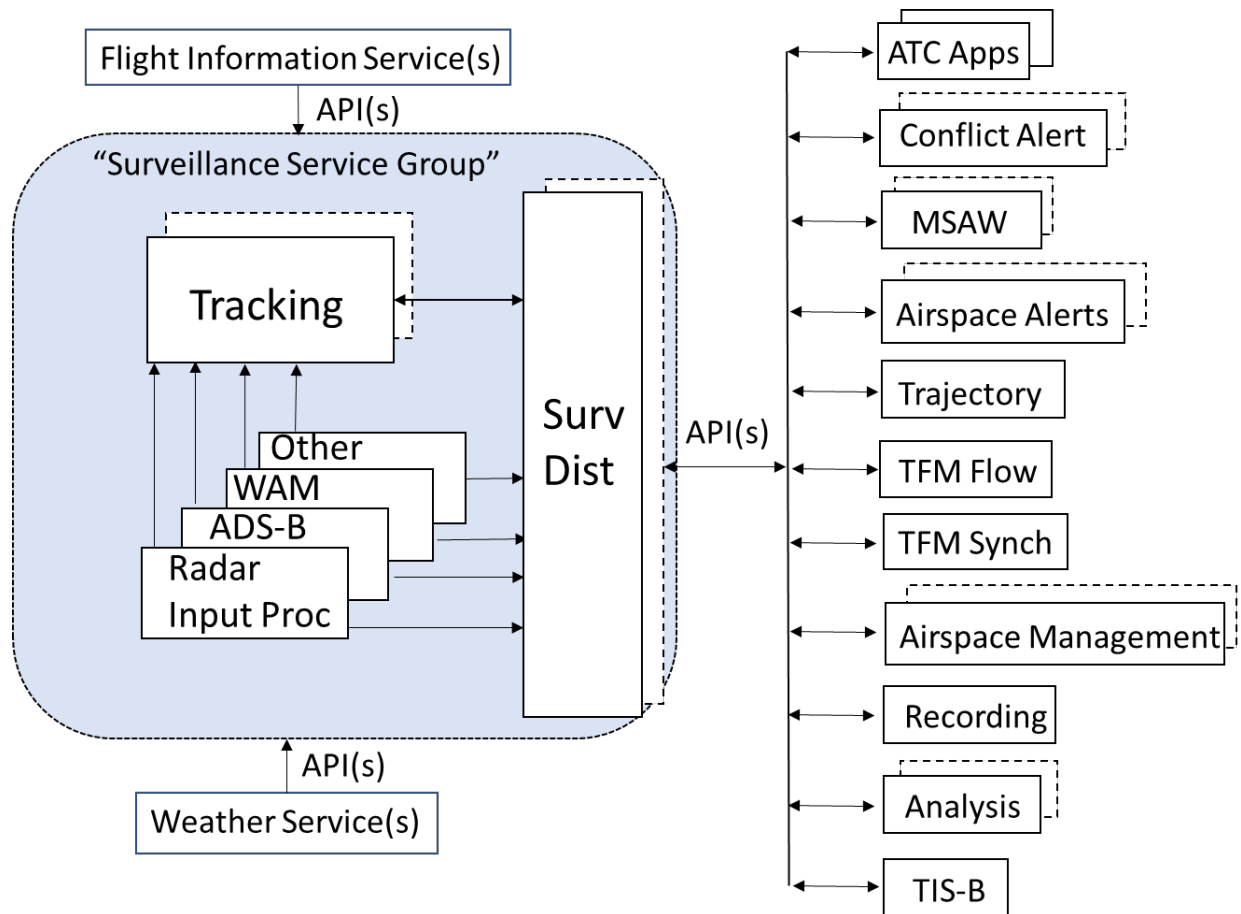


Figure 6-2. NAS Surveillance Services

In the figure, the output of the Surveillance Service Group, represented by the collection of services in the blue area, is viewed by surveillance data consumers as a single service mediated by the Surveillance Distribution Service. Collectively, those services provide the surveillance data to the non-comprehensive list of services and applications depicted on the right. The Surveillance Services are accessed by one or more APIs structured to provide a data stream meeting the specific operational needs of each of the consuming services/applications. Further description and rationale for this model is described below.

In ATC automation designs, the SDP is roughly decomposed into surveillance source input processing, source-specific quality assurance functions, tracking, and mechanisms for both internal and external data distribution. In the NAS, sources of surveillance data include radar (primary and secondary), ADS-B, and WAM. Those sources reflect distinct mechanisms of surveillance operation and associated sensor modes/states, different performance characteristics, and different interface protocols (i.e., different ASTERIX Categories defined for the purpose).

Accordingly, the Surveillance Services model in Figure 6-2 depicts those sources as distinct input processing services. Radar input processing functions include message processing, target data filtering, coordinate conversion, Mode C altitude pressure correction, functions that enhance target fidelity such as ground clutter filtering and beacon reflection and ring-around suppression,

and processing of multi-level weather intensity. Output quality is assured by mechanisms addressing both individual sensor performance (e.g., sensor status reports, permanent echo verification, test targets, sensor data counts, and radar message errors) and collective sensor performance (e.g., collimation and registration). Radar source input processing ensures that the quality and depth of surveillance coverage is preserved through mediation of overlapping sensor coverage and through effective sensor failure and recovery mechanisms. The primary output of the Radar Input Processing service is a stream of target reports to the Tracking service and the Surveillance Distribution Service in a standard coordinate system such as WGS 84 and standard protocol, most likely ASTERIX. A secondary output to the Surveillance Distribution Service is the multi-level weather intensity produced by select primary radars with weather capability. Although conceptually represented as a single service, expected workload, network topology, and resilience arguments may suggest regional or even facility-specific instances of this surveillance service with corresponding geospatial boundaries.

ADS-B input processing shares some similarities with radar processing, but also exhibits some differences. ADS-B functions include message processing, target data filtering, and coordinate conversion. Output quality is assured by mechanisms addressing sensor performance (e.g., sensor status reports and test reports), collective sensor performance (e.g., service volume message counts), and report quality (e.g., Navigation Integrity Category (NIC), Navigation Accuracy-Position (NACp) value for reported position accuracy and integrity, and Navigation Accuracy-Velocity (NACv) for velocity accuracy). Unlike current radar processing capabilities, current ADS-B processing is divided between the surveillance broadcast service and the ADS-B report processing organized by service volume that occurs in ATC systems. In this service-based architecture context, the ADS-B processing service is assumed to encompass both. The output of the ADS-B Input Processing service is a stream of target reports to the Tracking service and the Surveillance Distribution Service using the standard coordinate system (e.g., WGS 84) and standard protocol (e.g., ASTERIX).

WAM is a ground-based, distributed, cooperative surveillance system that uses a minimum of three receivers to determine aircraft positions by calculating the differences in transponder signal arrival times. WAM leverages SBS infrastructure, adding radio stations to existing ADS-B station locations and implementing multilateration processors to SBS control stations. Target reports are distributed in a standard protocol (e.g., ASTERIX). Automation systems process WAM input as virtual radar target reports.

In addition to radar, ADS-B, and WAM input processing services, additional surveillance input processing services are foreseen to support integration of new entrants in the NAS. For example, scheduled to begin operation in 2021, the Space Data Integrator (SDI) will process received vehicle data from space launch and reentry operators based on vehicle telemetry. The source and prospective application of this information suggest a distinct input processing service will be required. Similarly, proliferating UAS operations and concepts for Extensible Traffic Management (xTM) have the potential to generate a need to integrate UAS surveillance into ATC operations in a manner yet to be determined.

Received surveillance data is processed by tracking functions to provide the best estimates of target properties including position, velocity, and maneuver occurrences. Tracking functions include track initiation, report to track correlation, update, coast and termination, altitude validation, and track-to-flight plan association. In existing ATC automation systems, the tracking function integrates the available surveillance sources to provide a “fused” output. In the service-based architecture model, a similar Tracking Service is envisioned that integrates the outputs of

Radar, ADS-B, and WAM input processing services in a manner similar to existing ATC automation capabilities, but also includes the potential of additional Tracking Service options, represented by the dashed shadow box. Because tracking performance involves tradeoffs between characteristics of both the surveillance sensors involved and the targets being surveilled, it may occur that new surveillance capabilities and new NAS entrants warrant development of tracking services better optimized for those characteristics. Tracking Service output is provided to the Surveillance Distribution Service and is expected to conform to a standard coordinate system and protocol (e.g., WGS 84 and ASTERIX). In addition to the Radar, ADS-B, and WAM processing inputs, the Tracker Service has inputs from a Flight Information service, providing aircraft identification and intent information to support track and flight data pairing. Inputs mediated by the Surveillance Distribution Service may also be necessary. For example, if Surveillance Service Groups are implemented regionally, then instances of these tracking services will need to share information. Also, existing track controls and interactions among tracking and separation management applications suggest that the APIs may need to involve transactions between ATC applications and tracking services.

A Surveillance Distribution Service ensures that surveillance information including both tracks and target reports are provided to requesters based on expressed need. Surveillance Distribution Service functions include managing subscriptions/requests for surveillance information from consuming services and applications, managing service level access controls, filtering data for operational context (e.g., service volume, surveillance source, and target class), filtering data for operational sensitivities (e.g., national security operations), and other functions to be determined. There is a potential for multiple distribution services. For example, non-critical consumers of surveillance services such as analysis applications and consumers external to NAS may have different performance requirements than ATC operations (e.g., latency) that warrant differentiating the distribution services.

Figure 6-2 includes a sample of prospective consumers of surveillance services. Foremost among those are ATC Applications. Those are the necessary counterparts to the facility-based ATC automation of today. ATC Applications are expected to provide the following:

- User interfaces (display, keyboard, trackball, or other input mechanisms) for controllers, supervisors, traffic managers, and technical support personnel.
- NAS common mission service proxies – In the service-based architecture, many of the functions performed in today's ATC automation systems are allocated to common mission services, such as the surveillance service group. These functions will be replaced in ATC Applications by a common mission service that establishes and manages connections with common mission services, receives and processes common mission service data in accordance with published APIs, and cache and store enterprise service data as necessary to preserve state, provides access to recent data history, and promotes resilience.
- Other ATC Application elements – In addition to the elements for user interfaces and common mission service proxies, there will likely be components dedicated to integration functions. These functions involve integrating the inputs from the Surveillance Service Group and other common mission services, interacting with controller, supervisor and TechOps workstations, and providing functionality that is unique to the domain or facility. Service integration may involve applications developed for the purpose, may leverage service mesh capabilities of the Platform Layer, or both. ATC Application

elements also include monitor and control capabilities for the local information technology environment.

Among the services consuming surveillance data are decision support applications for separation management. Those include Conflict Alert (CA), Minimum Safe Altitude Warning (MSAW), restricted airspace alerting functions, and other separation management support functions that are presently included in the SDP subsystems of ATC automation. Those are represented as common mission services because they reflect distinct software components in existing automation and consume surveillance information with little, if any, transactions back with surveillance components. In all cases, the potential for multiple services is depicted, recognizing that operational distinctions between domains (en route, terminal, surface, and ocean) may drive a need for service distinctions.

A critical service for NAS operations is Airspace Management. Airspace Management ensures that all regions of airspace for which U.S. ATC is responsible are assigned to an operational ATC unit. The service also ensures that all eligible aircraft receive services and that responsibility and eligibility for communications and aircraft separation are uniquely assigned to a controller. Existing NAS-wide Controller-Pilot Data Link Communications (CPDLC) session management may be a model for aspects of this service.

Other services are also dependent on the Surveillance Service. These include Traffic Flow Management services, represented as distinct Strategic Flow and Synchronization services consistent with NAS EA, a Trajectory Service, likely to be one of a set of Flight Information-related services, and Analysis Services that use surveillance information for a variety of purposes. A common mission Recording Service is also shown, representing the probable utility of logging functions implemented as part of the service environment. In addition to the common mission level, recording services may be distributed throughout the NAS IT environment and may be implemented as part of the Platform Layer. The Traffic Information Broadcast Service represents another potential class of consumers.

6.3.1.1 Surveillance Service Platform Layer

Surveillance Services use the services of the Platform Layer as described in Section 4. Because the Surveillance Services are safety-critical and demand the highest performance among Mission Services, there are some notable distinctions identified below.

Runtime

Software Hosting/Execution. For Surveillance Services, there are two aspects – the execution environment for the common mission Surveillance Services and the execution environment for ATC Applications in NAS operating facilities. The latter are additionally partitioned into the local server environment for applications and service clients and the user interface applications running at user operational positions. Both the common mission Surveillance Services and the local server environment for applications and service clients in NAS facilities will be hosted in a managed container environment, consistent with the recommended Reference Architecture pattern.

Monitoring and Log Analysis. Monitoring and logging will apply at both the common mission and local facility levels.

Service Mesh, Proxy, and Gateway. Service meshes, proxies, and gateways will be involved in Surveillance Service, but the details are to be determined. The mechanisms used may depend on the configuration of services and hosting environments. For example, the constituent

Surveillance Services may reside in the same hosting environment, suggestive of a service mesh-based relationship. Consumption of the Surveillance Services is projected to be mediated by a distribution service, which is suggestive of an API gateway. Criticality of service is an influence as is geographic distribution (e.g., ATC Applications are expected to be hosted in many distributed ATC facilities). In the NAS facility processing environments, co-location of service clients and applications suggest a service mesh.

Virtual Networking, Policy, Authentication and Authorization. Virtual network capabilities that connect services and enforce access policies are expected to apply. Distinctions between safety and efficiency-critical applications and services and those of lesser criticality suggest isolating VNETs based on criticality.

Distributed Database and Storage. For the Surveillance Services, there are operational data stores associated with each service and expectations of a common mission level database for recordings for analysis applications. Examples include:

- For radar, there is a database of adapted radar sensors and associated information that support the quality assurance functions of collimation and registration, adapted zones to filter clutter, a map of sort cells that determine preferred sensors at each point in space, and other required adaptations. Short term storage of target data supports the quality assurance functions and reporting mechanisms such as QARS. There are adapted parameter values that govern functional behavior.
- For ADS-B and WAM, there are simpler data stores of adapted sensors and associated information.
- For the Tracking Service, there is a data store of tracks and track history as well as adapted parameter values that influence tracking functional behavior.
- The Surveillance Distribution Service has data stores that include subscription information and access permissions, defined and assigned service volumes, and other adapted configuration information.
- At the common mission level, all received target reports, tracks, and data output by the data distribution service are recorded and stored for analysis.

Streaming and Messaging. The Surveillance Services involve both streaming and messaging services. Messaging services would be employed for the publish/subscribe mode of operation of the distribution service. The subsequent delivery of target and track data to service consumers is inherently a streaming service. Within the Surveillance Services Group, information is streamed among the constituent services.

Development

Development Frameworks and Libraries. Because the Surveillance Services are safety-critical, there is motivation to promote or restrict software development to programming languages and frameworks with suitable properties. In practice, subsets of major system programming languages have been defined that ensure safety conventions are observed. Those include MISRA-C, MISRA-C++, SPARK Ada, and JSR-302 Safety-Critical Java (SCJ). Those language subsets promote determinism in operation through mechanisms such as restricted memory management functions (e.g., no garbage collection). Rust is a relatively new programming language defined for safe systems.

Planning and Requirements Management. Effective requirements management is necessary for Surveillance Services for two reasons: to meet the demands of DO-278A compliance and other safety assurance processes and to ensure that the legacy capabilities being superseded by the Surveillance Services are adequately replicated to support operations across the domains. It may not be the case that requirements must continue to be expressed in the same way, managed in a traditional database structure such as DOORS, or documented in a formal requirements specification. Contemporary workflow tools such as Jira and modeling tools such as Cameo include approaches for managing requirements in the context of Agile software development and system architecture and modeling, respectively.

Software Packaging, Repositories, and Distribution. There is nothing inherently different for the Surveillance Services, but safety-critical software packaging involves greater expectations of configuration management integrity.

Application Programming Interface and Data Management. For the Surveillance Services, the APIs associated with the Surveillance Distribution Service will be critical to the operation and performance of applications and services consuming surveillance data.

Continuous Integration / Continuous Delivery Toolchain. There is nothing inherently different for the Surveillance Services, but greater emphasis on configuration management is expected. Traceability between requirements, however expressed, and testing is paramount.

6.3.1.2 Surveillance Services Infrastructure Layer

Figure 6-3 represents the primary elements of the Surveillance Service infrastructure. Going from the bottom to the top of the figure, these elements include:

- Surveillance sensors (shown are CARSR, ASR, and ADS-B sensors)
- SDN
- NAS-wide hosting of common mission services, including the constituent services of the Surveillance Service
- IT infrastructure hosting service clients and ATC applications on premise in FAA facilities
- Remote ATC User workstations connected to FAA facilities for hosted services and applications.

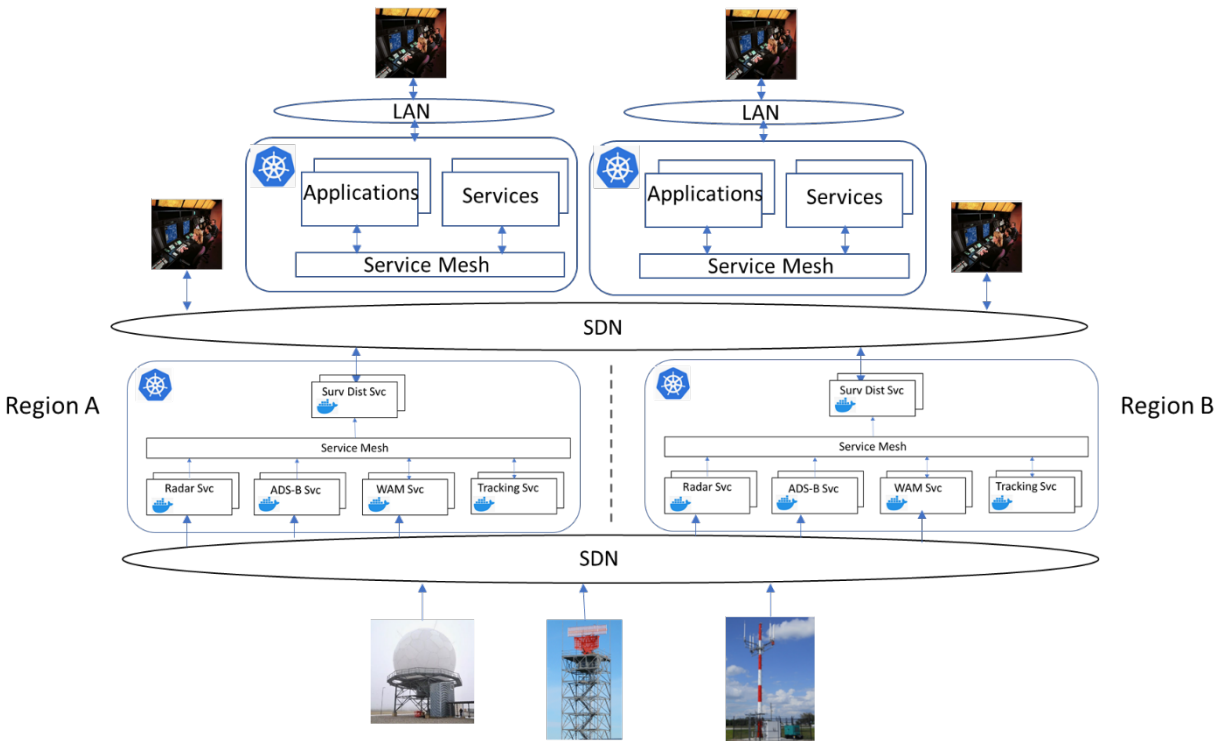


Figure 6-3. Surveillance Services Infrastructure

Surveillance sensors number in the hundreds and will communicate to common mission surveillance input processing services through the SDN using ASTERIX protocols. The SDN ensures that each sensor has redundant, physically diverse data paths to each instance of surveillance input processing service.

Common mission surveillance services support safety-critical NAS-wide operations. Surveillance services are projected to be hosted in managed container processing environments. In the figure, the environments are represented as Docker containers managed by Kubernetes, the most popular utilities for the purpose currently. To meet the required .99999 availability of surveillance services and dependent applications supporting separation management, associated mission services, applications, and communication elements must meet an even higher standard of availability and involve multiple levels of redundancy. Each service is allocated to a container. Multiple containers/service instances operate concurrently to ensure that no single failure of an instance/container can disrupt service availability. Kubernetes manages container start, stop, and failover.

Redundancy is provided for the NAS common mission level processing environment itself. Physically independent and geographically diverse IT environments (represented in the figure as Regions A and B) exist to ensure that even failure or destruction of a common mission processing environment would not result in loss of common mission services. Conceptually, that corresponds to the operating region/availability zone approach employed by cloud service providers. In each environment, services communicate locally via a service mesh. The common mission processing environments also ensure that other aspects of performance are satisfied, such as load-balancing among service instances, and environments are configured to ensure minimum latency in surveillance service processing. Surveillance Services output, streaming target and track data, is distributed to other mission services operating within the local IT

processing environment and to surveillance service clients and applications within ATC facilities.

The IT processing environment within each ATC facility, if needed, is a smaller scale version of the common mission processing environments. Surveillance services are hosted in managed container processing environments, also represented as Docker containers managed by Kubernetes. Containers host common mission service clients as well as ATC applications that integrate the inputs from mission services and support user interaction via user workstations. Support may be provided for workstations operating remotely.

The consistency of the described IT processing environments, common mission and local, offers many degrees of freedom with which to work as NAS automation needs and capabilities evolve. As confidence with cloud service performance is established, cloud services may supplement or replace on-premises assets. Specific services and ATC applications may be either harmonized or differentiated to meet operational needs. Processing environments, common mission and local, can be scaled to support arbitrary degrees of ATC service consolidation.

6.3.2 Surveillance Services Challenges

The Surveillance Services challenges identified to date include:

- Service availability management. Managed container IT environments, in the cloud or on-premises, need to demonstrate in combination with redundancy strategies that service availability exceeding .99999 can be met.
- Latency. The Reference Architecture includes processing overhead associated with layer services that impact latency. It must be demonstrated that the Surveillance Services can meet separation management performance requirements at controller workstations.
- Certification. Procedures exist to certify safety-critical services and systems in operation. Counterparts to these procedures need to be defined that accommodate a distributed service architecture.
- Controller UI and service interactions. A service-based architecture works best when constituent services are as independent as possible. Existing ATC user interfaces include interactions with system functions that may be difficult to replicate while maintaining the desired degree of service independence.
- Location-based influences on operations. There are existing NAS functions that are location based (e.g., beacon code management). These functions must be accommodated or superseded by common mission services.
- Software design assurance. DO-278A defines software design assurance levels and associated objectives applicable to NAS systems/services. Means to satisfy those objectives are suggested (e.g., plans, tests, configuration management), although alternative approaches may be offered. Suitable means must be found to meet applicable design assurance objectives within the Reference Architecture.

Appendix A Related Infrastructure Programs/Projects

This appendix provides a brief description of some FAA infrastructure programs that are relevant to the Service-Based Reference Architecture.

A.1 FAA Telecommunications Infrastructure (FTI) and FAA Enterprise Network Service (FENS)

FTI is the telecommunications infrastructure that supports all wide area communications needs for the entire FAA. The Operational IP (OPIP) network in the NAS Operations domain is among the various services FTI provides.

FTI also provides the NESG, which is the FAA boundary solution. The NESG serves as a security gateway between NAS systems connected to the OPIP network and all other systems.

FENS is the planned successor to FTI which will subsume FTI functions when the FTI contract end.

A.2 FAA Cloud Services (FCS)

The FCS program has established cloud environments that are available for use by FAA programs. These environments are created using commercial cloud services by Amazon (AWS) and Microsoft (Azure). Currently, FCS cloud environments are being used for Mission Support applications, most notably the EIM Platform, described below. FCS currently provides Federal Information Management Act (FISMA) Moderate services, but FISMA High services are planned. Both Government Community Cloud and Commercial Cloud services are available for Mission Support applications, with plans to have the same available to support NAS operations.

A.3 Integrated Enterprise Services Platform (IESP)

The IESP is a virtualization platform that operates in the NAS. It can provide VMs for NAS programs. These VMs come with monitoring, FTI connectivity, authentication, and backup services. IESP can be used for the consolidation of equipment for existing programs or avoiding the need to buy equipment (e.g., servers) for new programs.

The IESP has primary and backup data centers at the Atlanta (ATL) and Salt Lake (SLC) National Enterprise Monitoring Centers, respectively. Several NAS programs including Aeronautical Information Management Modernization (AIMM) (for Aeronautical Common Service deployment) use the IESP currently and more are planning to use the platform in the near future.

A.4 National Cloud Integration Services (NCIS)

NCIS is an FAA program that supports integrating Air Traffic Organization applications into the cloud. NCIS supports programs managed by the Program Management Organization (AJM) that are transitioning to FCS. NCIS advises program offices on cloud technologies for optimization and cost efficiency. Goals of NCIS include:

- Commoditize hardware across the agency
- Leveraging economies of scale

- Supporting an on-demand self-service model for computing
- Providing rapid elasticity – the ability to automatically scale up and down in response to system load for optimal user experience and cost
- Providing measured service – in which the FAA pays for consumed resources only
- Shifting IT spending to operational expenditures (OpEx) rather than capital expenditures (CapEx).

A.5 System Wide Information Management (SWIM)

SWIM is a NAS-wide information system that supports Next Generation Air Transportation System (NextGen) goals. SWIM facilitates the data sharing requirements for NextGen, providing the digital data-sharing backbone of NextGen. SWIM enables increased common situational awareness and improved NAS agility to deliver the right information to the right people at the right time. This information-sharing platform offers a single point of access for aviation data, with producers of data publishing it once and users accessing the information they need through a single connection.

SWIM provides standards and infrastructure to facilitate the dissemination of information from producers to consumers, using both event-driven as well as request/response information exchange patterns. The SWIM infrastructure includes the NAS Enterprise Message Service (NEMS), which comprises a set of nodes that provide message broker and web service interfaces. SWIM standards define protocols and message formats that information producer and consumer systems use to connect to NEMS nodes and exchange information. SWIM also provides the NAS Services Registry and Repository (NSRR), which lists all the information services and provides links to documentation on how to receive and process the information provided by these services.

NAS systems connect to SWIM over the FTI OPIP network, while airlines and other users external to the FAA access SWIM via the NESG, as well as introduced cloud-based distribution via the SWIM Cloud Distribution Service (SCDS).

A.6 SWIM Cloud Distribution Service (SCDS)

SCDS is a cloud-based distribution service that was established leveraging FCS and AWS. It facilitates the distribution of SWIM data to external users via the NESG and is set up to move data only in one direction (from the NAS Operations environment to external users) per security requirements. All publicly available SWIM data is available for access via SCDS in a user-friendly simple fashion resulting in an improved user experience. Improved automation has resulted in lower cost for service delivery.

A.7 Enhanced SWIM Cloud Service

The SWIM Program is working to take the SCDS experience to another level by implementing Enhanced SWIM Cloud Service (ESCS) which will replace SCDS and provide improved services such as bi-directional information exchange and introduction of a cloud-based security gateway that is compliant with Trusted Internet Connection (TIC) 3.0 as specified by the Department of Homeland Security (DHS). That allows a more direct access to “SWIM in the cloud” by SWIM users that are external to the FAA.

A.8 Enterprise Information Management (EIM) Platform

The EIM Platform is a capability that leverages FCS to ingest and store data from various sources in the FAA including SWIM for the creation of a single data lake users can access securely for data retrieval, manipulation, and analysis. The platform also provides a collection of tools for FAA users to employ for their analysis. Presently, the EIM Platform is built on AWS, which is one of the cloud service providers under FCS and mainly supports Mission Support domain needs. There are many features provided by the EIM Platform including an *application mall* that programs in Mission Support can use for hosting applications.

Appendix B Platform Survey

This appendix contains a summary of information gathered after conducting a survey about existing platforms. That information was used to help validate that the platform elements listed in Section 4 of the document were sensible and that important platform elements were not missing.

B.1 Application Based Capability Development (ABCD)

Application Based Capability Development (ABCD) is a MITRE project focused on learning how Agile and DevSecOps processes can be applied in an FAA context. ABCD envisions a framework designed to:

- Deliver useful new Traffic Flow Management (TFM) information applications in six months or less.
- Enable TFM users to compose their workspace from a collection of available components (“widgets”) that are useful in multiple TFM contexts.
- Establish common data and computational capabilities where all applications share the same “truth” and capabilities can be used together without data mismatches.
- Involve users throughout the process, so that new capabilities meet operational requirements from the start.
- Automate testing, security scanning, and deployment using modern Development, Security, and Operations (DevSecOps)¹⁹ tools to minimize defects.
- Support both desktop and mobile users.

The ABCD Mission Applications developed to date include a Time-Based Flow Management (TBFM) Airborne Delay Monitor, a Pathfinder²⁰ Coordination application, a Departure Fix Usage Visualizer, and an Electronic Flight Strip application. Backing these browser-based applications are a set of microservices including a Flight Information service, a Flight Position service, a Traffic Management Initiatives (TMI) service, and so on, as well as “translator” services that function to translate data from SWIM formats to Avro (Apache Hadoop) formats used within ABCD. These applications and services run in near-real-time and are designed to handle the performance loads created by the entire set of flights active across the NAS.

The ABCD software currently comprises prototypes used for demonstration purposes, however if a technology transfer is completed these applications and services would be used by FAA

¹⁹ “DevSecOps” is a variant of “DevOps” (Development and Operations) that includes integrated security scanning tools. DevOps tools [8] are designed to automate much of the software development, integration, testing, and deployment process to improve delivery speed and reduce problems.

²⁰ The term “Pathfinder” here refers to an FAA operational process/procedure that uses a flight designated as a “Pathfinder” to explore whether a departure route should be reopened for general use. (Note that the term “pathfinder” is also used in other contexts to describe a project done to learn from, in order to reduce a technical or process risk.)

specialists such as traffic management coordinators to support NAS service threads considered efficiency-critical.

The ABCD applications and services currently run on VMs within the MITRE IT environment, however the Infrastructure as Code methodology and tools would make it possible to quickly build and deploy ABCD to other environments, such as FCS/AWS. More information on ABCD can be found in [7].

An overview of the ABCD operational architecture is provided in Figure B-1, and an overview of the ABCD development architecture is provided in Figure B-2.²¹

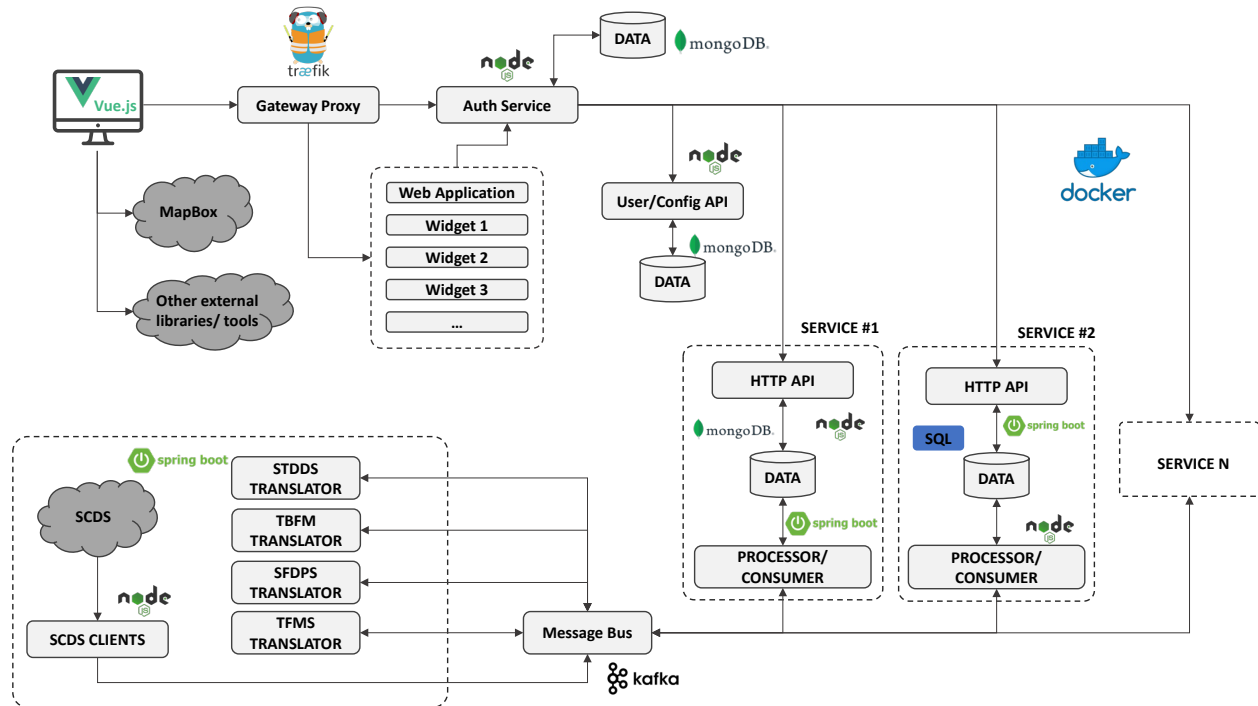


Figure B-1. ABCD Operational Architecture

²¹ Product names, logos, brands, and other trademarks featured or referred to within the documentation are the property of their respective trademark holders. These trademark holders are not affiliated with the author or this project in any way.

Service-Based Reference Architecture for NAS Automation - Version 1.2

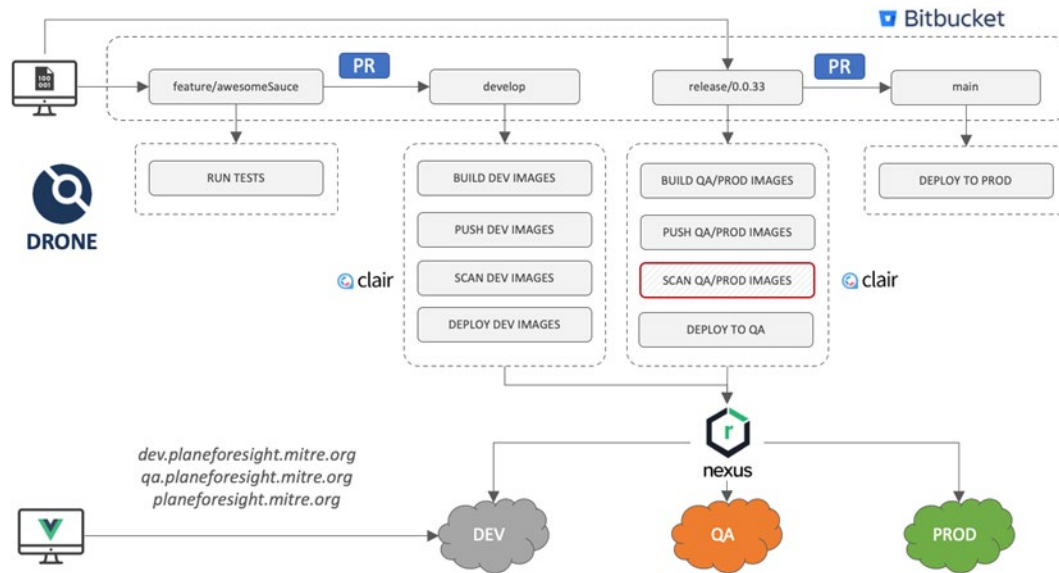


Figure B-2. ABCD Development Architecture

Table B-1 shows the elements of the Platform Layer in this Reference Architecture that are addressed in the ABCD runtime and development architectures shown in the above figures.

Table B-1. Platform Layer Elements in ABCD

Platform Element	ABCD Coverage
<i>Runtime</i>	
Software Hosting/Execution	ABCD service components are packaged using Docker. However, some elements (e.g., Kafka message broker) run on VMs for performance reasons. Docker Swarm used to allow multiple instances of each service to be instantiated as necessary to meet performance needs.
Workflow Choreography and Orchestration	n/a
Monitoring and Log Analysis	Prometheus, Conductor
Service Proxy, Mesh, and Gateway	Traefik proxy for routing service invocations to instances
Virtual Networking, Policy, Authentication and Authorization	Vault (for storing credentials), Docker Swarm virtual networking
Distributed Database and Storage	MongoDB, clustered SQL database
Streaming and Messaging	Solace client used to subscribe to live data from SWIM. Kafka broker used for internal high-performance message communications.
Analytics and Artificial Intelligence	n/a

<i>Development</i>	
Development Frameworks and Libraries	SpringBoot, Node.js, Vue.js (for GUIs)
Planning and Requirements Management	Jira, Confluence
Software Packaging, Repositories, and Distribution	Nexus, Docker registry, NPM proxy
API and Data Management	Swagger for documenting APIs, AVRO schemas, Manual documentation of Kafka topics
CI/CD Toolchain	Drone, Bitbucket, Maven, Clair, Jenkins

B.2 Configuration, Logistics, and Maintenance Resource Solutions

Configuration, Logistics, and Maintenance Resource Solutions (CLMRS) includes a group of FAA programs: Configuration Management Automation (CMA), Logistics Center Support System (LCCS), and Automated Maintenance Management System (AMMS), which are being developed by the CLMRS team using an Agile approach (Scaled Agile Framework) and DevSecOps methodology. CLMRS is envisioned to revolutionize the way the Agency leverages information, beginning with logistics, maintenance, and configuration management.

CLMRS users are the TechOps workforce responsible for logistics, maintenance, and configuration of the NAS. CLMRS software runs in the FAA's Mission Support domain.

CLMRS systems run on the AWS cloud environment provided by FCS.

More information on CLMRS can be found in [21].

Table B-2 shows how the elements of the Reference Architecture Platform Layer are addressed in the CLMRS environment.

Table B-2. Platform Layer in CLMRS

Platform Element	CLMRS Coverage
<i>Runtime</i>	
Software Hosting/Execution	CRI-O, Kubernetes
Workflow Choreography and Orchestration	N/A
Monitoring and Log Analysis	Prometheus, Fluentd, Managed AWS Elastic
Service Proxy, Mesh, and Gateway	HAProxy

Platform Element	CLMRS Coverage
Virtual Networking, Policy, Authentication and Authorization	Open vSwitch
Distributed Database and Storage	etcd
Streaming and Messaging	AWS SNS/SQS
Analytics and Artificial Intelligence	
<i>Development</i>	
Development Frameworks and Libraries	C#, Java, Python, PL/SQL, Angular v2, Golang
Planning and Requirements Management	Jira (primary), Confluence, Doors (mapped from Jira)
Software Packaging, Repositories, and Distribution	Artifactory, Quay
API and Data Management	3scale API Management
CI/CD Toolchain	Bitbucket with Git, looking for approval to use GitHub, TeamCity, Selenium (web), UTPLSQL (Oracle SQL), WinAppDriver (Windows client), Quay (container vulnerabilities)

B.3 Enterprise Information Management (EIM) Data Platform (DP)

The purpose of the EIM Platform is to provide FAA users, departments, and programs with an effective and efficient environment to perform post-operational data analysis and provide information management support functions using FAA NAS, mission support, administrative and other data. EIM DP contains a common *unified data layer*²² and hosts shared, common enterprise information management capabilities, processes, products, and tools. It runs on AWS GovCloud West IaaS provided via FCS and is accessible via the FAA Mission Support Network.

EIM DP, along with its tools and capabilities, is primarily a data lake containing NAS data. It can be used by programs to create new Mission Applications that leverage the data ingest and storage capabilities of EIM DP and use the EIM DP tools to analyze and present data to meet the needs of users. The end users of EIM DP include analysts that use the provided tools to process, analyze, and view NAS data in various ways, either directly using EIM DP tools (e.g., querying datasets using Presto and Hive query engines via the Hadoop User Experience web interface), or

²² This unified data layer is not the same as the NAS data plane mentioned in the Reference Architecture. This term is used to be consistent with how EIM DP is presented in its documentation. The NAS data plane manages service invocations and event driver information exchanges. The EIM unified data layer contains a data lake that users can query directly.

by viewing or downloading displays and reports produced by Mission Applications that are hosted in the EIM DP environment (e.g., Accessible Rich Internet Applications [ARIA]).

EIM DP varies from the service-based approach assumed by this Reference Architecture in that it provides Mission Applications and end users with direct access to a common data layer, rather than having a defined set of Mission Services that provide access to specific data and functionality via service interfaces. That is natural given the purpose of EIM DP, and it does not preclude using EIM DP to create new mission services that provide access to raw or processed data from the data lake via service interfaces such as REST APIs.

More information on EIM DP is available in [22] and [23].

An overview of the EIM DP runtime architecture is provided in Figure B-3. and an overview of the development architecture is provided in Figure B-4.

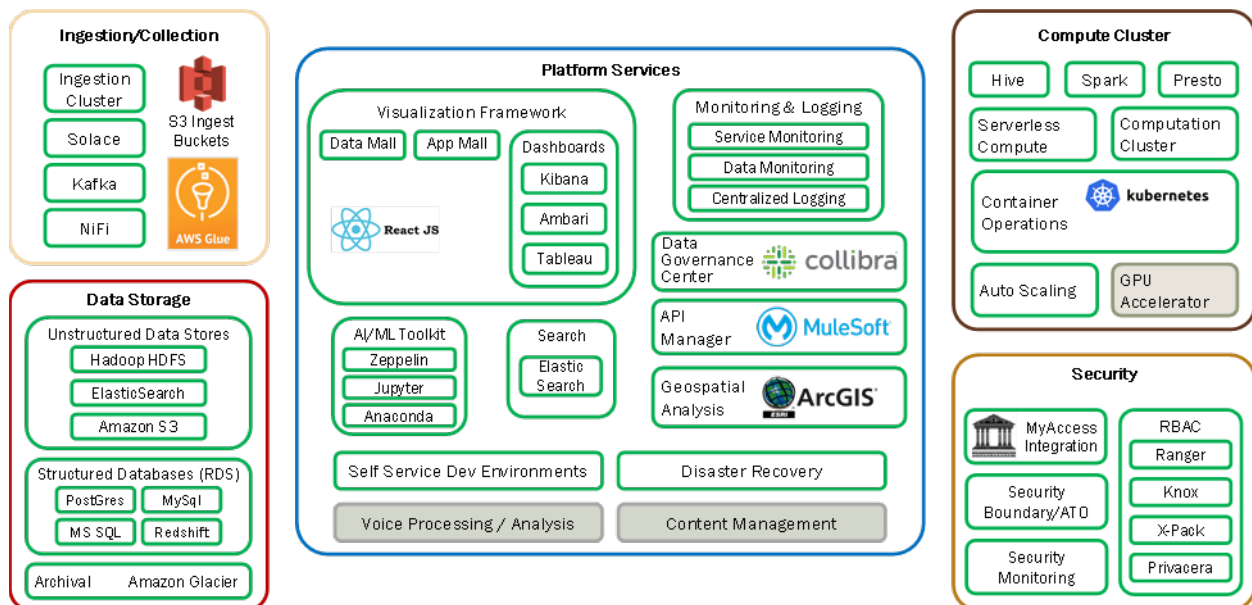


Figure B-3. EIM Platform Runtime Architecture

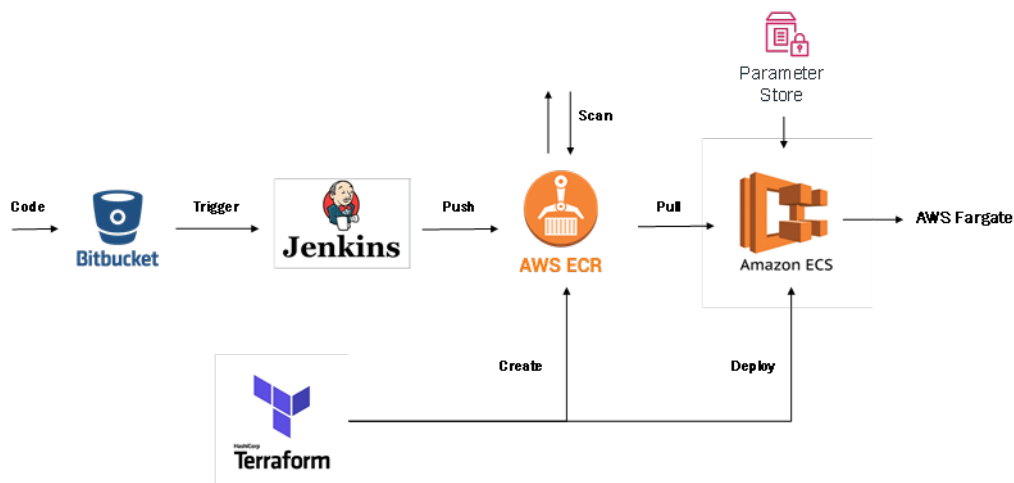


Figure B-4. EIM Platform Development Architecture

Table B-3 shows how the elements of the Reference Architecture Platform Layer are addressed in the EIM Platform environment.

Table B-3. Platform Elements in EIM Platform

Platform Element	EIM Platform Coverage
<i>Runtime</i>	
Software Hosting/Execution	Software that makes up the EIM Platform, and software created by other programs that use the EIM Platform, may be installed directly on AWS VMs or may be containerized. Containerized software may run in Docker containers on VMs or may use AWS Elastic Container Service (ECS), with ECS clusters run by AWS Fargate serverless computing or deployed onto elastic compute instances (VMs) managed by application owners. Kubernetes or OpenShift is planned for the future. EIM Platform runs an analytics compute cluster, using AWS Elastic Map Reduce (EMR), a hosted Hadoop Cluster, and an ArcGIS Geospatial compute cluster. EMR clusters can also be provisioned for user programs, which then communicate with EIM Platform to retrieve data over internal AWS circuits enabled by VPC peering. AWS Athena and Lambda can be used for serverless computing. EIM Platform also provides a presentation layer that allows users to run tasks to analyze data using scripting languages or using provided dashboards and visualizations.
Workflow Choreography and Orchestration	Tools such as NiFi, AWS Glue used to orchestrate data ingest workflows.
Monitoring and Log Analysis	No mention of this topic in the documentation, although LogStash is listed as an available tool.
Service Proxy, Mesh, and Gateway	Amazon tools. Not using service mesh. APIs hosted in Mule runtime instances on EIM Platform, as well as APIs developed using Java/Springboot service (presumably running in EC2 instances.)
Virtual Networking, Policy, Authentication and Authorization	AWS VPCs and Transit Gateway managed by FCS. IAM, AWS Security Groups, Azure AD, Cylance Protect, My Access, Ranger, Knox, X-Pack, Privacera, Collibra, Anypoint Platform
Distributed Database and Storage	Amazon RDS (Postgres, MySQL, MS SQL Server, RedShift), Unstructured (Dynamo DB, S3, HDFS, ElasticSearch), Amazon Glacier, Athena
Streaming and Messaging	Apache Kafka, Apache NiFi, and Solace (SWIM client) are used for data ingest. Stream Sets, AWS Glue, Hortonworks DataFlow, Apache Sqoop used within analytics environment.
Analytics and Artificial Intelligence	Hadoop, Spark, Map Reduce 2, Pig, Apache Hive, Apache Flink, Presto, Hortonworks Data Platform, Jupyter, Hue, Anbari, Kibana
<i>Development</i>	
Development Frameworks and Libraries	Eclipse+plugins, ArcGIS, React.JS

Planning and Requirements Management	Jira, Confluence
Software Packaging, Repositories, and Distribution	Artifactory, AWS Elastic Container Registry (ECR). NAR, JAR, property files. Amazon Machine Image (AMI). Chocolatey for Windows packages.
API and Data Management	MuleSoft API Anypoint Portal, Collibra data governance
CI/CD Toolchain	FAA Bitbucket, Jenkins, Maven, SonarQube, Clair, Fortify, Nessus, Ansible, Terraform

B.4 Elroy

Elroy is an FAA project to explore Rapid Development and Deployment, to:

- Answer the call for innovation,
- Modernize the FAA’s non-safety critical software development methods and capabilities to deliver capabilities faster,
- Influence Automation Evolution and Pathfinder strategy,
- Reduce costs using Cloud service and other cost saving measures, and
- Deliver more user value.

An overview of the Elroy system is provided in Figure B-5.²³

²³ Source: Briefing by Shirley Burgess, 26 October 2020.

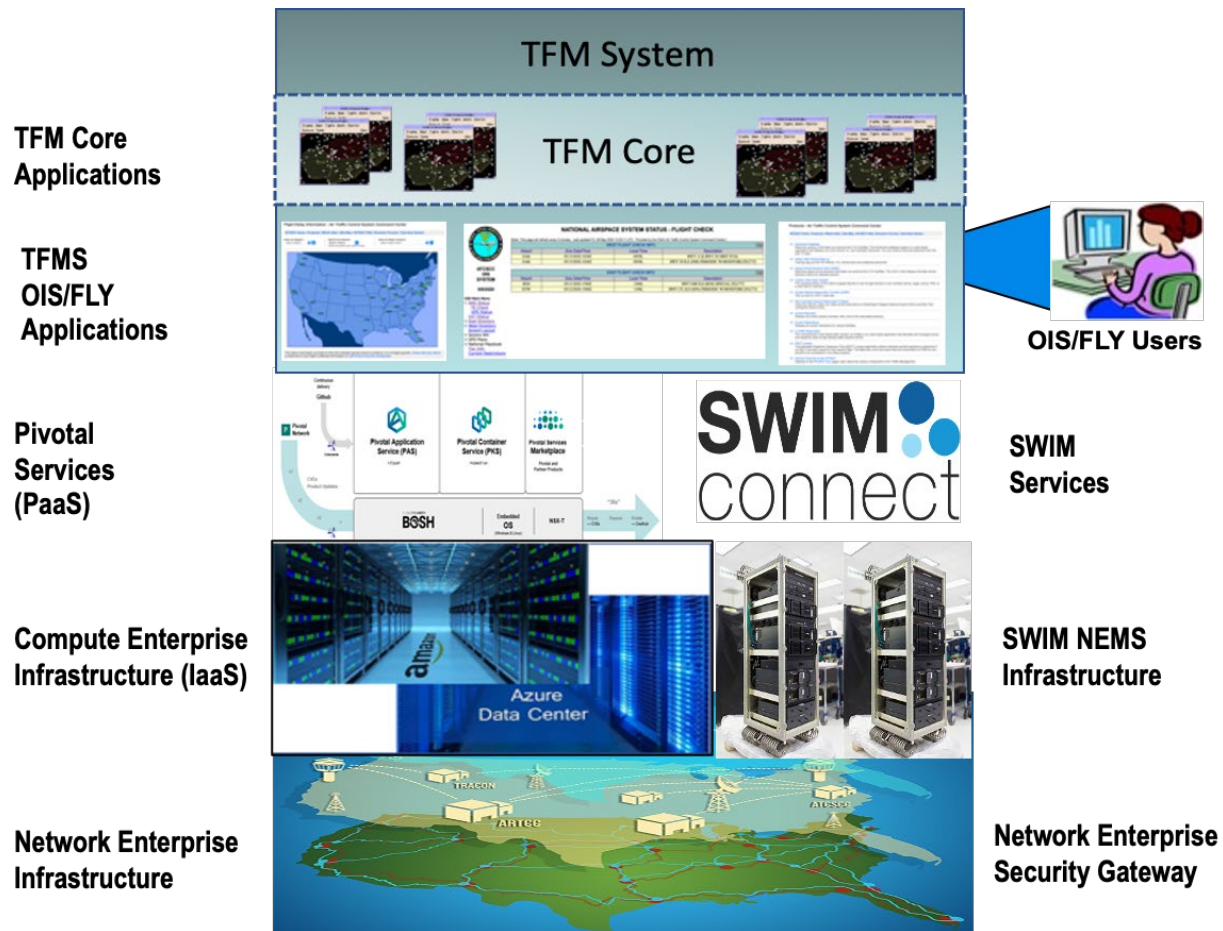


Figure B-5. Project Elroy System Overview

An overview of the Project Elroy Platform Layer is provided in Figure B-6.

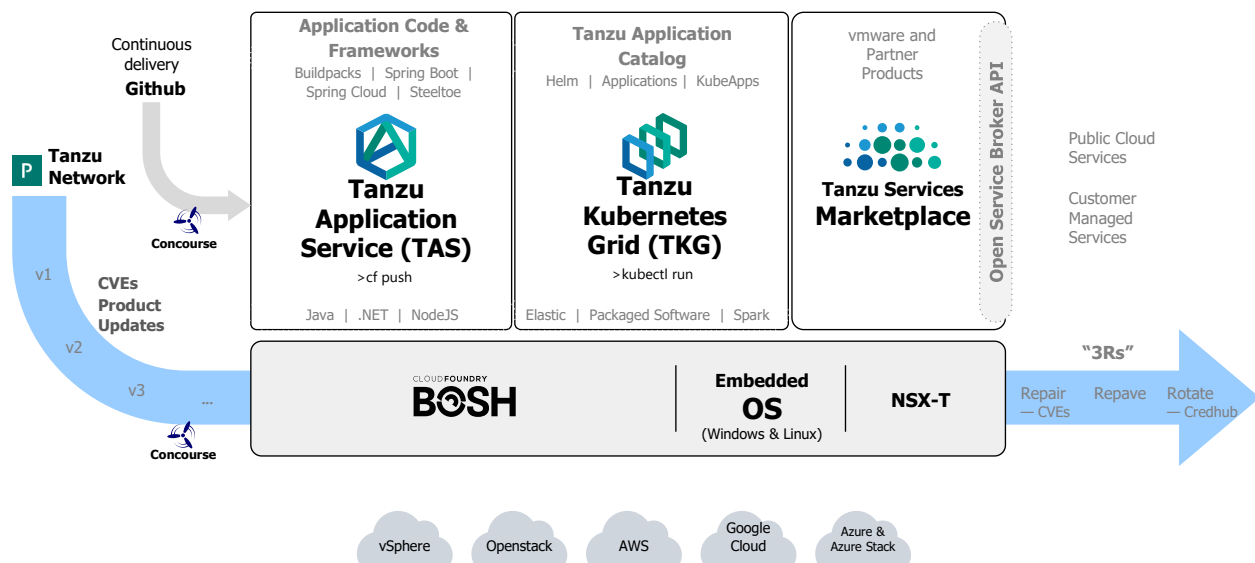


Figure B-6. Project Elroy Platform Overview

shows how the elements of the Reference Architecture Platform Layer are addressed in the Project Elroy Platform environment.

Table B-4. Elroy Platform

Platform Element	Elroy Platform Coverage
<i>Runtime</i>	
Software Hosting/Execution	Containers and Kubernetes (Pivotal’s “Droplets” and “Stem Cells” and Tanzu Kubernetes Grid), Bosh
Workflow Choreography and Orchestration	n/a
Monitoring and Log Analysis	Open-source tools, Healthwatch, considering Dynatrace
Service Proxy, Mesh, and Gateway	GoRouter, Contour
Virtual Networking, Policy, Authentication and Authorization	UAA (User Account and Authentication service, Cloudfoundry authentication component), CredHub, FAA Azure AD, AWS IAM (for ops team)
Distributed Database and Storage	PostgreSQL, S3
Streaming and Messaging	Solace JMS client, Solace VMR
Analytics and Artificial Intelligence	n/a
<i>Development</i>	
Development Frameworks and Libraries	Buildpacks (e.g., Node.js, NGINX, Spring cloud services)
Planning and Requirements Management	Jira
Software Packaging, Repositories, and Distribution	Harbor, Artifactory, S3
API and Data Management	n/a
CI/CD Toolchain	Concourse

B.5 Platform One

The DoD Enterprise DevSecOps Initiative is a DoD Joint Program that is bringing Enterprise IT Capabilities to programs across the DoD with Cloud One and Platform One. Cloud One provides access to cloud computing services, and Platform One is the first DoD-wide approved DevSecOps managed service. Platform One brings timeliness and modularity and enables reuse by providing a collection of approved, hardened Cloud Native Computer Foundation (CNCF)-compliant Kubernetes distributions, infrastructure as code playbooks, and hardened containers that implement a DevSecOps platform compliant with the DoD Enterprise DevSecOps Reference Design.

The DoD Chief Information Officer (CIO) is pushing for broad adoption across a wide range of programs creating software for a wide range of missions. The DoD Enterprise DevSecOps Reference Design and Platform One are currently being used on software for the F-35 and F-16 aircraft, the Air Operations Center (AOC), the Advanced Battle Management System (ABMS), and Ground Based Strategic Deterrent (GBSD), and more.

Platform One services are intended to be hosted on Cloud One. In some cases, a dedicated instance of the Platform One tools is instantiated, using Cloud One resources, for a particular program or development team. This model is suitable for large teams/programs that need a dedicated enclave. In other cases, a shared enterprise environment is provided and managed by the Platform One team and used by multiple different programs or development teams. This model is suitable for smaller or medium sized teams.

The DoD has made a great deal of information on the Enterprise DevSecOps initiative²⁴ and Platform One readily available. Documents, briefings, and videos are available at [24] and [25].

An overview of the layers that make up the DoD Enterprise DevSecOps Layers is provided in Figure B-7. The top layer in the figure corresponds to the “Mission Software Layer” in our Reference Architecture. The middle three layers collectively make up “Platform One” and correspond to the “Open Software-Based Platform Layer” in our Reference Architecture. The “Infrastructure Layer” at the bottom of this figure corresponds to the “Computing Resources” layer in our Reference Architecture.

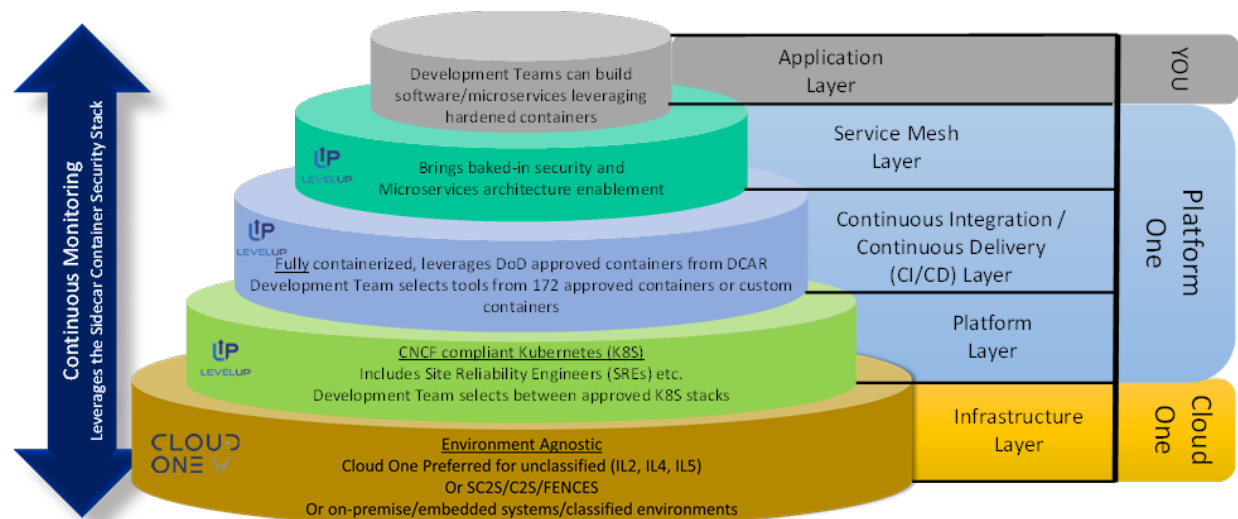


Figure B-7. Overview of DoD Enterprise DevSecOps Layers

Table B-5 shows how the elements of the Reference Architecture Platform Layer are addressed in the DoD Platform One environment.

²⁴ DoD Enterprise DevSecOps Initiative is a joint program with OUSD(A&S), DoD CIO, U.S. Air Force, DISA and the Military Services

Table B-5. Platform Layer Elements in DoD Platform One

Platform Element	Platform One Coverage (examples only)
<i>Runtime</i>	
Software Hosting/Execution	Containers managed by technologies such as Kubernetes or Openshift are the preferred means of providing the execution environment in Platform One. Direct hosting on VMs is also supported but discouraged.
Workflow Choreography and Orchestration	Nothing listed
Monitoring and Log Analysis	A variety of tools are supported, including Sensu, EFK, Splunk, and more. (Prometheus, used in ABCD, is included.)
Service Proxy, Mesh, and Gateway	ISTIO Service Mesh is an important part of Platform One. API gateways including Kong, Azure and AWS API are used. HA Proxy, Envoy, etc.
Virtual Networking, Policy, Authentication and Authorization	Virtual networking is included within the Container Management tools, and enclaves can be created on Cloud One. Security tools are provided for both static and dynamic analysis, as well as operational monitoring.
Distributed Database and Storage	An extensive set of relational and unstructured persistence tools/services are provided including SQL Server, MySQL, MongoDB, Redis, Elasticsearch, and so on.
Streaming and Messaging	Kafka, Flink, Nats, RabbitMQ, and ActiveMQ are supported. (However, unclear if any of these are provided as cross-DoD enterprise messaging services comparable to SWIM in the FAA.)
Analytics and Artificial Intelligence	Kubeflow for AI/ML. A wide variety of tools/services for analytics, including Tableau and Kibana for visualization, Hadoop, Elasticsearch, and Oracle Big Data for unstructured data analysis.
<i>Development</i>	
Development Frameworks and Libraries	Nginx, Apache2
Planning and Requirements Management	Jira, Confluence, etc.... Pivotal Tracker
Software Packaging, Repositories, and Distribution	Artifactory, Nexus. For containers, IronBank (DCAR)
API and Data Management	Nothing listed
CI/CD Toolchain	Jenkins, Github Government, GitLab, Maven, Gradle, Ant, Cucumber, J-unit, Clair, Ansible, Terraform, Helm

Appendix C References

- [1] The MITRE Corporation, "Automation Evolution Strategy," The MITRE corporation, McLean, VA, 2020.
- [2] U.S. Department of Defense, "DoD Enterprise DevSecOps Reference Design Version 1.0," Department of Defense Chief Information Officer, 2019.
- [3] C. Andrews, D. Chaloux, W. Heagy, M. Liggan, O. Olmos and D. Thomson, "National Airspace System Automation Evolution Strategy: Coordination Draft," The MITRE Corporation, McLean, VA, 2020.
- [4] The MITRE Corporation, "Automation Evolution Strategy Work Plan Status," The MITRE Corporation, McLean, VA, 2020.
- [5] The Federal Aviation Administration, "2035 Vision for Air Traffic Management Services," The Federal Aviation Administration, Washington, DC, 2020.
- [6] U.S. Department of Defense, "DoD Enterprise DevSecOps Reference Design Version 1.0," Department of Defense Chief Information Officer, 2019.
- [7] The MITRE Corporation, "Application Based Capability Development Framework: An Agile Process and Software Framework for Rapid Implementation of Traffic Flow Management Capabilities," The MITRE Corporation, McLean, VA, 2020.
- [8] P. Koka, B. McKenney, S. Paul, and D. Thomson, "FAA Zero Trust Architecture and Implementation Analysis," The MITRE Corporation, McLean, VA, 2020.
- [9] Atlassian, "DevOps: Breaking the Development-Operations barrier," 2020. [Online]. Available: <https://www.atlassian.com/devops>. [Accessed 2 April 2020].
- [10] Stevens Institute of Technology, International Council on Systems Engineering (INCOSE), Systems Engineering Research Center, and School of Systems and Enterprises, "Report of the Workshop on The Relationship Between Systems Engineering and Software Engineering," 12-14 June 2014. [Online]. Available: https://www.incose.org/docs/default-source/newsevents/report.pdf?sfvrsn=5c0db4c6_0. [Accessed 20 April 2021].
- [11] Federal Aviation Administration, "6000.30F - National Airspace System Maintenance Policy," FAA, 22 April 2013. [Online]. Available: https://www.faa.gov/regulations_policies/orders_notices/index.cfm/go/document.information/documentID/1020968. [Accessed 23 March 2021].
- [12] The MITRE Corporation, "Ten Strategies of a World-Class Cybersecurity Operations Center," MITRE, Bedford, MA, 2014.
- [13] Federal Aviation Administration, "FAA Reliability, Maintainability, and Availability (RMA) Handbook, FAA RMA-HDBK-006C, V1.1," 2015.
- [14] Atlassian, "Scrum of Scrums," Atlassian, 2021. [Online]. Available: <https://www.atlassian.com/agile/scrum/scrum-of-scrums>. [Accessed 19 February 2021].

- [15] Federal Aviation Administration, "National Airspace System 2025 Top Level Far Term System Requirements Document Version 3.0," FAA, Washington, DC, 2014.
- [16] David L. Bloom, Steven R. Bodie, Nicholas T. Hamisevich, Dr. Stephane L. Mondoloni, Brian T. Simmons, Tejal N. Topiwala, "Guidance on Flight Information Management for Microservices, MTR210465," The MITRE Corporation, 2021.
- [17] T. Olavsrub, "What is a data architect? IT's data framework visionary," 20 October 2020. [Online]. Available: <https://www.cio.com/article/3586138/what-is-a-data-architect-its-data-framework-visionary.html>. [Accessed 19 March 2021].
- [18] C. Tozzi, "Precisely," 25 March 2020. [Online]. Available: <https://www.precisely.com/blog/big-data/big-data-101-batch-stream-processing>. [Accessed 29 January 2021].
- [19] VMWare, "Hyperconverged Infrastructure," VMWare, Inc., 2021. [Online]. Available: <https://www.vmware.com/products/hyper-converged-infrastructure.html>. [Accessed 20 4 2021].
- [20] NUTANIX, "What is Hyperconverged Infrastructure?," Nutanix, 2021. [Online]. Available: <https://www.nutanix.com/hyperconverged-infrastructure>. [Accessed 20 April 2021].
- [21] The Federal Aviation Administration, *Configuration, Logistics, and Maintenance Resource Solutions (CLMRS) 101 (Briefing)*, Washington, DC: The Federal Aviation Administration, 2020.
- [22] General Dynamics Information Technology, "FAA EIM Platform Developers Guide, Version 13," General Dynamics, 2020.
- [23] Federal Aviation Administration, "EIM Platform Government Furnished Equipment -- Appendix B Design Guide," FAA, Washington, DC, 2020.
- [24] U.S. Air Force Assistant Secretary of Acquisition, "DoD Enterprise DevSecOps Initiative and Platform One," [Online]. Available: <https://software.af.mil/dsop/documents/>. [Accessed 20 January 2021].
- [25] U.S. Air Force Assistant Secretary of Acquisition, "Platform One: DoD Enterprise DevSecOps Services," [Online]. Available: <https://software.af.mil/dsop/services/>. [Accessed 20 January 2021].
- [26] IBM, "Advantages of XML," IBM, [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzamj/rzamjintroadvantages.htm. [Accessed 17 March 2021].

Appendix D Acronyms

Term	Definition
ABCD	Application Based Capability Development
ADS-B	Automated Dependent Surveillance-Broadcast
AIMM	Aeronautical Information Management Modernization
AL	Assurance Level
AMPQ	Advanced Message Queuing Protocol
API	Application Program Interface
APT	Advanced Package Tool
ARSR	Air Route Surveillance Radar
ARTCC	Air Route Traffic Control Center
ASR	Airport Surveillance Radar
ASTERIX	All Purpose Structured Eurocontrol Surveillance Information Exchange
ATC	Air Traffic Control
ATL	Atlanta
ATM	Air Traffic Management
ATO	Authority to Operate
AWS	Amazon Web Services
CA	Conflict Alert
CAT	Category
CI/CD	Continuous Integration/Continuous Delivery
CLMRS	Configuration, Logistics, and Maintenance Resource Solutions
CNI	Container Network Interfaces
ConOps	Concept of Operations
ConUse	Concept of Use
COTS	Commercial off-the-Shelf
CPC	Certified Professional Controllers
CPDLC	Controller-Pilot Data Link Communications
CSP	Cloud Service Provider
CSS-FD	Common Support Services-Flight Data
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations

Term	Definition
DHS	Department of Homeland Security
DNS	Domain Name Service
DoD	Department of Defense
DOORS	Dynamic Object-Oriented Requirements
DTD	Document Type Definition
EA	Enterprise Architecture
EDDS	En Route Data Distribution System
EIM	Enterprise Information Management
EIM-DP	Enterprise Information Management Data Platform
EP	Edge Protection
ERAM	En Route Automation Modernization
ESB	Enterprise Service Bus
ESCS	Enhanced SWIM Cloud Service
ETL	Extract, Transform, and Load
FAA	Federal Aviation Administration
FCS	FAA Cloud Service
FENS	FAA Enterprise Networking System
FISMA	Federal Information Management Act
FOSS	Free and Open-Source Software
FTI	FAA Telecommunications Infrastructure
FY	Fiscal Year
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IAG	Internet Access Gateway
IAM	Identity and Access Management
IDAT	Interfacility Data Transfer
IdM	Identity Management
IESP	Integrated Enterprise Services Platform
IMS	Information Management Service
IoT	Internet of Things
IP	Internet Protocol

Term	Definition
IT	Information Technology
JSON	JavaScript Object Notation
LAN	Local Area Network
MBSE	Model-Based System Engineering
MISRA	Motor Industry Software Reliability Association
MQTT	Message Queuing Telemetry Transport
MSAW	Minimum Safe Altitude Warning
NACp	Navigation Accuracy Category-Position
NACv	Navigation Accuracy Category-Velocity
NAS	National Airspace System
NAS-RD	NAS Requirements Document
NCIS	National Cloud Integration Service
NEMC	NAS Enterprise Management Center
NEMS	NAS Enterprise Messaging Service
NESG	NAS Enterprise Security Gateway
NextGen	Next Generation Air Transportation System
NIC	Navigation Integrity Category
NIST	National Institute for Standards and Technology
NSRR	NAS Service Registry and Repository
OPIP	Operational IP
PaaS	Platform as a Service
PSR	Primary Surveillance Radar
QA	Quality Assurance
QARS	Quick Analysis of Radar Sites
RA	Reference Architecture
RHEL	Red Hat Enterprise Linux
RPM	Red Hat Package Manager
RTCA	Radio Technical Commission for Aeronautics
SBS	Surveillance and Broadcast Services
SCDS	SWIM Cloud Distribution Service
SCJ	Safety Critical Java
SDN	Surveillance Data Network

Term	Definition
SDP	Surveillance Data Processing
SFDPS	SWIM Flight Data Publication Service
SLC	Salt Lake City
SLE	Second Level Engineering
SNS	Simple Notification Service
SOA	Service Oriented Architecture
SOC	Security Operations Center
SQS	Simple Queue Service
SSR	Secondary Surveillance Radar
STDDS	SWIM Terminal Data Distribution Service
SWIM	System Wide Information Management
TBFM	Time-Based Flow Management
TDM	Time Division Multiplexing
TechOps	Technical Operations
TFDM	Terminal Flight Data Management
TFM	Traffic Flow Management
TFMData	Traffic Flow Management Data Service
TFMS	Traffic Flow Management System
TIC	Trusted Internet Connection
TIS-B	Traffic Information Service - Broadcast
TRACON	Terminal Radar Approach Control
UAS	Unmanned Aerial System
UI	User Interface
URL	Uniform Resource Locator
USAF	United States Air Force
VM	Virtual Machine
VNET	Virtual Network
VPC	Virtual Private Cloud
VPN	Virtual Private Network
WAF	Web Application Firewalls
WAM	Wide Area Multilateralation
WAN	Wide Area Networks

Term	Definition
XDF	Extensible Data Format
XML	Extensible Markup Language
xTM	Extensible Traffic Management
ZTA	Zero Trust Architecture