MASTER THESIS

CONNECTED DEFENCE: NEXT-GENERATION DATA PLATFORM FOR MILITARY INTELLIGENCE AND OPERATIONS

VALENTIN PFEIL

University of the Bundeswehr Munich,
Department of Computer Science,
Institute for Software Technology

SUPERVISED BY
PROF. DR. WOLFGANG HOMMEL,
DR. KARL FÜRLINGER

JUNE 25, 2025

Ideas alone have little worth. The value of an innovation lies in its practical implementation.

— WERNER VON SIEMENS

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Prof. Dr. Wolfgang Hommel for his commitment over the past months. Especially for proofreading, discussions and efforts to create a proper working atmosphere.
There is also much gratitude for giving me the opportunity and trust to do my master's thesis under his supervision. He has always been a source of inspiration and motivation for true passion. And even so, he gives the best example of how to stay grounded despite great success.
I would also like to express my sincere gratitude to Dr. Karl Fürlinger for his support. His guidance on my path helped me to gaint clarity, orientation and specialisation, particularly in critical areas.

Then, I would like to thank M. Sc. Alexander Vogel, M. Sc. Christian Nilsson, M. Sc. Frank Keienburg, as well as the entire *OmniAware* team, for their valuable support. The Cloud Native Hyperscalers subpractice of Capgemini's Business Unit Germany enabled the collaboration with the University of the Federal Armed Forces in Munich. This cooperation was made possible through Capgemini's involvement in the Strategic Collaboration Agreement with Amazon Web Services, which also provides funding support for innovative projects such as this one. Without Capgemini's strategic contribution and the backing of the Strategic Collaboration Agreement, the development of a more authentic and immersive product would not have been achievable.

My thanks also go to my friends, colleagues and comrades, who are not just well-meaning observers but companions who have experienced the same challenges, failures and successes. We have come a long way together. Notably, one of my dearest friends and colleagues Christoph, who excelled in machine learning, showed me what it takes to push your limits and guided me on my path to cloud engineering.

I would also like to emphasise the unconditional love and support I have received from my family. In particular, my grandparents have inspired me to be the best version of myself. My aunt and cousins have always given me advice and support. Last but not least, my uncle Yakup has made me interested in computer science, guided me through it and has always been by my side as a role model but also as a mentor.

ABSTRACT

The increasing complexity of multinational defence operations demands secure and scalable systems for managing data under strict sovereignty and security requirements. This study presents a Proof of Concept for a Connected Defence platform, focusing on the design and implementation of the core system architecture as the central element of the project. Based on the North Atlantic Treaty Organization Architecture Framework Version 4, the core integrates cloud-based infrastructure, advanced security protocols and standardised interfaces to enable secure, efficient and interoperable data exchange across diverse stakeholders.

The core system is designed as a robust and modular foundation, emphasising flexibility and scalability through the integration of cloud and edge computing technologies. A key focus is the implementation of advanced security mechanisms to ensure data sovereignty, confidentiality and compliance with stringent defence regulations. Additionally, the development of standardised and extensible interfaces plays a critical role in enabling seamless communication and interoperability between diverse components and systems.

While optional extensions, such as a sensor module, a situational awareness platform and a digital twin simulationssystem, are outlined as potential future components, the Proof of Concept focuses exclusively on validating and implementing the core system. This foundation provides the necessary infrastructure to later incorporate real-time visualisation, predictive simulations and operational data processing, demonstrating the platform's potential for supporting advanced defence operations.

By leveraging high-performance computing alongside cloud and edge technologies, the Proof of Concept addresses key challenges in modularity, secure communication and scalability. This research establishes a clear and actionable architectural blueprint, showcasing the feasibility and technical robustness of the core system as a critical step toward developing a next-generation Connected Defence platform.

CONTENTS

1 INTRODUCTION

The increasing complexity of modern defence operations demands secure, scalable and interoperable information technology (IT) infrastructures to support mission-critical applications. Multi-domain operations — spanning land, air, sea, space and cyber — require highly resilient and adaptable computing frameworks capable of processing and analysing vast amounts of data in real time. Current defence IT architectures often suffer from fragmentation, limited interoperability and security vulnerabilities that can significantly impact operational effectiveness and strategic decision making [23], [26].

Cloud computing has emerged as a key enabler of modern defence capabilities, providing elastic computing, scalable storage and a globally distributed network infrastructure. The ability to efficiently handle large workloads while maintaining data sovereignty and security is critical for defence applications. Cloud architectures facilitate mission-critical operations by providing high availability, automated resource management and the ability to deploy advanced security controls such as encryption, access control policies and zero-trust architectures [31], [41].

Leading cloud providers, such as Amazon Web Services (AWS), have tailored solutions for the defence sector, ensuring compliance with military security requirements while maintaining the benefits of cloud scalability and efficiency. In the US, AWS GovCloud, for example, provides an isolated cloud environment that meets stringent regulatory and compliance standards for defence and government applications. In addition, AWS Nitro Enclaves provide confidential computing capabilities that enable the secure execution of sensitive workloads within cryptographically attested enclaves, mitigating the risk of data exposure and unauthorised access [14], [37].

Recognising the increasing demand for digital sovereignty within Europe, AWS has introduced the AWS European Sovereign Cloud, one or many independent cloud regions within the European Union (EU) designed to meet European data residency, compliance and operational control requirements. It is set to launch by the end of 2025. This initiative ensures that European governments, defence agencies and critical industries retain full control of their data while benefiting from AWS global infrastructure and advanced security features. By leveraging physically and logically separate cloud regions, AWS ensures compliance with strict European regulations while enabling secure multi-domain operations [65].

However, despite the advantages of cloud adoption, integrating cloud computing into defence environments introduces new challenges related to security, compliance and interoperability. Defence organisations require stringent guarantees of data confidentiality, integrity and sovereignty, particularly when operating in coalition-based or untrusted environments. Trusted Execution Environments (TEE) and confidential computing play a critical role in addressing these challenges by ensuring that data remains protected even during processing through hardware-enforced security mechanisms. These developments set the stage for exploring secure and compliant defence cloud architectures [56].

Capgemini is a global leader in consulting, technology services and digital transformation with a strong focus on innovation and modern solutions for various industries, including defence. With a presence in over 50 countries, Capgemini provides strategic guidance and technological expertise to drive digital transformation and operational excellence.

Within Capgemini, the Business Unit (BU) Germany plays a significant role in delivering tailored solutions for German enterprises and public sector organisationations. The BU Germany is structured into multiple business lines that focus on specific technological and industry-oriented domains.

The research is embedded within the Application Business Line (ABL) Practice Cloud and Custom Application (C&CA), which specialises in designing, developing and implementing scalable cloud architectures and customised software solutions. C&CA focuses on integrating advanced cloud technologies, ensuring compliance with industry standards and optimising operational efficiency for mission-critical applications. By leveraging Capgemini's global expertise and local market knowledge, the Cloud and Custom Application supports clients in achieving secure, interoperable and high-performance cloud infrastructures tailored to their specific operational needs.

As part of its strategic initiatives in cloud computing and digital transformation, Capgemini has established a long-term Strategic Collaboration Agreement (SCA) with AWS. This agreement strengthens the partnership between the two companies, enabling joint efforts to drive innovation, accelerate cloud adoption and develop industry-specific solutions leveraging AWS's advanced cloud services. The SCA focuses on enhancing cloud security, scalability and operational efficiency while supporting enterprises in their transition to cloud-native architectures. Additionally, the SCA provides funding mechanisms to support research and development projects that align with its strategic goals.

A key aspect of the SCA is the expansion of Capgemini's AWS Centers of Excellence (CoEs), which provide expertise and support for clients in various industries, including defence and public sector operations. Through this collaboration, Capgemini and AWS facilitate the adoption of modern cloud technologies, including confidential computing, artificial intelligence (AI) -driven analytics and resilient cloud infrastructures. The SCA also enables the deployment of solutions that embed services such as AWS GovCloud and AWS Nitro Enclaves to ensure compliance with stringent regulatory requirements, particularly for highly sensitive and mission-critical applications.

By leveraging the SCA, Capgemini is positioned to support organisations in achieving secure, scalable and compliant cloud solutions. This collaboration aligns with the broader goals of digital sovereignty, operational agility and advanced cloud security, providing a strong foundation for modern defence and enterprise cloud environments. Furthermore, the funding mechanisms provided by the SCA directly contribute to research initiatives such as this thesis, whose PoC aligns with the agreement's objectives, allowing AWS to support the project's funding and technical enablement.

HENSOLDT, a key customer of Capgemini in the defence sector, is a German defence technology company specialising in sensor solutions for surveillance, reconnaissance and electronic warfare. The company develops advanced systems for land, air, sea and cyber operations with a strong focus on sensor fusion and AI-driven analytics.

One of HENSOLDT's latest innovations is Ceretron, a sensor suite designed for real-time (RT) data fusion and enhanced situational awareness in complex operational environments. As cloud computing plays an increasing role in defence applications, integrating scalable cloud solutions could further enhance Ceretron's capabilities by enabling advanced data processing, AI model updates and secure, cross-platform data sharing. This aligns with HENSOLDT's digital transformation strategy, leveraging cloud technologies to optimise operational efficiency and decision-making in mission-critical scenarios.

This thesis aims to investigate the design and validation of a secure, interoperable and NAFv4-compliant defence cloud architecture leading to the first research question.

RQ1: How can a cloud-native defence architecture be designed to ensure compliance with the NATO Architecture Framework Version 4 (NAFv4) while supporting secure and scalable mission-critical operations?

Ensuring compliance with NAFv4 in a cloud-native defence environment requires a structured architectural approach that aligns with NATO's predefined viewpoints, including operational, systems and technical perspectives. A defence cloud must facilitate multi-domain integration, interoperability with coalition networks and mission assurance while maintaining a modular and scalable system design. The architectural design must incorporate Infrastructure-as-Code (IaC) principles, containerised workloads and dynamic orchestration mechanisms that adhere to NAFv4-defined interoperability standards. Additionally, the architecture must support secure information exchange across different classification levels while enforcing zero-trust security models and role-based access control (RBAC). Ensuring resilience against cyber threats and operational disruptions requires embedding fault tolerance mechanisms, redundant data pathways and decentralised decision-making processes.
Chapter 2 outlines the theoretical foundations of NAFv4, detailing how its principles guide modern defence IT infrastructures, while Chapter 3 explores how cloud-native technologies can be integrated into an NAFv4-compliant design without compromising operational efficiency.

RQ2: What are the key security challenges in defence cloud infrastructures and how can a confidential computing-based security model be validated to ensure compliance with defence security standards?

Defence cloud infrastructures are prime targets for cyber espionage, data exfiltration and advanced persistent threats (APT) due to the sensitivity of mission-critical workloads. Traditional encryption mechanisms protect data at rest and in transit, but ensuring data-in-use confidentiality remains a challenge. Confidential computing, leveraging TEE, provides an additional layer of security by isolating sensitive workloads at the hardware level. However, validating its effectiveness in a defence context requires assessing remote attestation protocols, enclave integrity verification and cryptographic key management strategies. Compliance with defence security frameworks such as the National Institute

of Standards and Technology (NIST) Confidential computing guidelines, International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 27001 and military-specific risk assessment models must be ensured. Additionally, real-world adversarial testing is necessary to evaluate security validation methodologies through penetration testing, compliance verification and cryptographic performance assessments.

While confidential computing addresses data-in-use security challenges, interoperability between cloud, edge and high-performance computing (HPC) environments introduces additional complexities. This requires investigating how defence systems can securely exchange data and workloads across diverse computing paradigms while maintaining operational efficiency, leading to the third research question.

RQ3: How can interoperability between cloud, edge and HPC environments be ensured in a defence cloud infrastructure while maintaining security and operational efficiency?

Modern military operations demand real-time data fusion across cloud, edge and HPC infrastructures, creating significant interoperability challenges. Cloud services provide scalable computing power, edge devices enable real-time battlefield analytics and HPC environments facilitate computationally intensive simulations. Seamless integration across these environments requires low-latency communication channels, secure federated identity management and cross-domain authentication. Additionally, differences in security postures, data formats and orchestration frameworks must be reconciled to ensure uninterrupted data flow. Secure API gateways, standardised message-passing protocols (e.g. Message Queuing Telemetry Transport (MQTT), Google Remote Procedure Call (gRPC) or Data Distribution Service (DDS) and distributed ledger-based access controls can mitigate interoperability risks while ensuring secure cross-domain operations). The architecture must also incorporate self-adaptive networking mechanisms capable of dynamically routing workloads between cloud, edge and HPC nodes based on network congestion, computational demand and security policies.
Chapter 4 describes the technical implementation of these interoperability mechanisms, while Chapter 5 evaluates their performance through empirical stress tests, latency benchmarking and resilience assessments under mission-critical conditions.

To answer these research questions, this thesis adopts a design science methodology, combining theoretical analysis with a practical PoC implementation. The PoC demonstrates the feasibility of a confidential and secure cloud architecture, evaluating its security properties, scalability and performance impact in a realistic defence scenario. By addressing the challenges of secure cloud adoption in defence, this work aims to contribute to the development of next-generation defence IT infrastructures, capable of providing trusted computing environments in coalition-based and sovereign military contexts [19], [32].

2 BACKGROUND

Summary: This chapter provides the theoretical and methodological foundation for the design and implementation of a secure, interoperable and sovereign data platform tailored to defence applications. It consolidates the architectural paradigms, compliance frameworks and technological principles upon which the system architecture developed in Chapter 3 is built. Rather than detailing the system design itself, which follows in later chapters, this section focuses on formal modelling methodologies, core security mechanisms and deployment paradigms that inform the architectural decisions and implementation artefacts.

The foundational elements covered herein — namely the NATO Architecture Framework Version 4 (NAFv4), Confidential Computing mechanisms and distributed cloud-edge computing fabrics — establish the structural underpinnings for the design realisation presented in subsequent chapters. These concepts are essential for understanding the platform's alignment with defence-grade standards, mission assurance requirements and operational interoperability objectives.

The chapter begins by positioning *OmniAware* as a federated and extensible defence data platform developed under Capgemini's Strategic Collaboration Agreement (SCA) with AWS. The platform is aligned with the alliance-wide objectives of digital sovereignty, real-time mission orchestration and secure coalition interoperability. Two operational scenarios — Platform Health Monitoring (PHM) and the Contextual Image Verification System (CIVS) — are introduced as concrete instantiations of the architectural design, each representing distinct mission needs and data processing requirements. These use cases serve as functional drivers for the capability mapping, service orchestration and deployment modelling performed in later chapters.

Building upon this applied mission framing, the NAFv4 section introduces the formal methodological approach that governs architectural modelling throughout the thesis. Emphasising traceability, semantic rigour and interoperability, NAFv4 provides the structural basis for decomposing strategic capability goals into technical deployment artefacts. The thesis operationalises NAFv4 using the ArchiMate modelling language and Archi tool, ensuring NATO-compliant artefact generation and alignment with coalition governance frameworks such as Federated Mission Networking (FMN).

The second major section is dedicated to *confidential computing* and *data sovereignty*, which are presented as foundational security primitives for modern military cloud systems. This segment elaborates on hardware-based Trusted Execution Environments (TEEs), remote attestation, cryptographic key provisioning and enclave integrity guarantees. Through an in-depth analysis of AMD SEV-SNP, Nitro Enclaves and associated key management flows, the thesis builds a security framework capable of withstanding adversarial, jurisdictional and coalition-based trust threats. These mechanisms are tightly mapped to regulatory mandates such as STANAG 4774/4778, *AC/322-D(2021)0032-REV1* and national cloud compliance requirements.

The final section introduces the operational computing architecture — comprising cloud, fog and edge tiers — necessary for the deployment of attested services, sovereign processing nodes and decentralised mission logic. It highlights how cloud-native infrastructures, when extended via fog and edge paradigms, can meet the latency, resilience and autonomy demands of modern tactical environments. Virtualisation, containerisa-

tion and Infrastructure-as-Code (IaC) are discussed as enabling technologies for secure workload orchestration, particularly in coalition and disconnected deployments.

Taken together, these pillars — formal architectural methodology (NAFv4), verifiable security mechanisms (confidential computing) and distributed deployment fabrics (Cloud, Fog, Edge) — provide the theoretical and technical baseline upon which the design and implementation in Chapters 3 and 4 are constructed. This background ensures that the proposed platform is not only technologically sound, but also strategically aligned with emerging defence doctrine and alliance-wide interoperability standards.

2.1 OMNIAWARE

The *OmniAware* Connected Defence Platform is being designed as a sovereign, federated and extensible architecture to address the complex demands of secure and interoperable data management in modern military environments. This thesis contributes to its conceptualisation and realisation as part of a broader initiative led by Capgemini under the Strategic Collaboration Agreement (*SCA*) with Amazon Web Services (AWS).

Within the scope of this thesis, *OmniAware* serves as a vehicle for demonstrating how NATO-aligned architecture models, confidential computing primitives and mission-oriented deployment patterns can be operationalised in a secure and sovereign manner. Rather than presenting a fully finalised system, the work focuses on modelling a compliant and deployable architecture prototype grounded in real-world requirements and defence-grade standards.

The platform targets dual-use scenarios within defence and homeland security and is aligned with Capgemini's strategic objectives under the AWS SCA. It is developed iteratively across multiple architectural layers, leveraging real-time orchestration, confidential analytics and policy-enforced data exchange across sovereign cloud, fog and edge deployments.

At its core, *OmniAware* aligns with Capgemini's strategic objectives under the SCA with AWS and is supported as part of Capgemini's Defense Europe initiatives. The platform targets dual-use contexts within defence and homeland security and has been modelled to support real-time mission orchestration, confidential analytics and zero-trust communication patterns across fog, edge and sovereign cloud deployments.

The project timeline is structured around quarterly milestones, aligned with the SCA fiscal calendar:

- **Q1 (Jan-April)** focused on foundational artefact development. These included the architecture-centric Press Release and Frequently Asked Questions (PR/FAQ) document [75], a comprehensive AWS aligned remediation plan derived from the Well-Architected Framework (WAF) Review, a capability and feature breakdown, a reference architecture structured around the AWS WAF and stakeholder-facing planning artefacts. Together, they established the baseline for a compliant and secure Minimum Viable Product (MVP).

- **Q2 (May-July)** marks the transition from documentation to implementation. The objective is the realisation of the *OmniAware* MVP. This includes developing deployable services, formalising technical artefacts for the AWS Foundational Technical Review (FTR) and preparing materials for the Field Ready Kit (FRK) — a strategic enablement package for AWS internal sales enablement and co-selling support. The FRK includes a Sales Brief, Solution Brief and a co-branded consulting offer.

Two distinct scenarios were derived from the project's mission framing and are modelled using Business Process Model Notation (BPMN). These are:

- **Platform Health Monitoring (PHM)**, which enables real-time analytics and anomaly detection on vehicle telemetry data using secure cloud-edge pipelines and attested processing workloads [78].
- **Contextual Image Verification System (CIVS)**, a service that validates tactical imagery against authoritative external references, supporting plausible inference under degraded trust conditions in reconnaissance missions [76].

The PHM scenario provides real-time vehicle condition monitoring and predictive maintenance capabilities within mission-critical contexts. Leveraging sophisticated analytics and artificial intelligence methodologies, this scenario realises advanced telemetry processing and anomaly detection. Specific capabilities include [79]:

- Continuous collection and sophisticated analysis of vehicle sensor telemetry (engine diagnostics, positional data via GPS, etc.).
- Contextual enrichment using external data sources such as terrain profiles, traffic intelligence and meteorological data.
- Integration and structured classification of optional radio audio transmissions via voice intelligence algorithms.
- Real-time anomaly detection leveraging advanced statistical models and machine learning.
- Context-aware anomaly validation employing artificial intelligence to distinguish critical from non-critical deviations.
- Persistent and auditable storage of anomalies and event data for operational assessment and forensic analysis.
- Visual dashboarding solutions for immediate operational status visibility and decision support.
- Decision-making augmentation concerning vehicle readiness and operational deployment.
- Utilisation of Natural Language Processing (NLP) techniques to corroborate sensor-detected anomalies via crew communication analysis.

The PHM scenario thus ensures operational reliability and enhanced decision-making accuracy, effectively contributing to overall mission assurance and readiness.

The CIVS scenario addresses tactical and strategic requirements for validating and classifying reconnaissance imagery within coalition and multinational environments. This use case implements sophisticated cross-domain validation techniques and robust classification according to NATO-defined security standards. The key capabilities encompass [77]:

- Real-time ingestion of imagery data streams enriched with precise metadata (GPS coordinates, timestamps, camera orientation).

- Automated classification of imagery in accordance with established NATO security categories (CAT-1 Unclassified, CAT-2 Restricted, CAT-3 Secret).
- AI-driven pattern recognition for extracting weather-related features directly from imagery.
- Systematic cross-validation of observed visual data against authoritative external weather data sources.
- Immediate automated alerting upon detection of discrepancies between visual and official environmental data.
- Enforcement of rigorous role-based access control (RBAC) to maintain strict security and confidentiality of sensitive imagery.
- Secure interfaces enabling authorised users to investigate anomalies in classified imagery safely.
- Provision of validated environmental intelligence to support high-level strategic and tactical decision-making processes.

Through its robust validation and security mechanisms, the CIVS contributes significantly to mission integrity by ensuring reliable environmental context verification, thereby enhancing operational decision-making confidence.
These scenarios have been methodically modelled using the Business Process Model Notation (BPMN) and defined ontologies to achieve a rigorous semantic structure. Consequently, they underpin the capability mapping used in subsequent architectural development phases compliant with NAFv4. By adhering strictly to formalised viewpoint-driven methodologies outlined by NATO [74], *OmniAware* aligns operational functionality with technical infrastructure requirements. It leverages contemporary technology stacks including Kubernetes for container orchestration, confidential computing frameworks (notably AMD SEV-SNP) and zero-trust security models for data sovereignty and compliance assurance.

This thesis contributes a validated architectural model demonstrating how sensitive, multi-domain data flows can be securely and efficiently managed, thereby addressing gaps identified in current literature and practice.

By employing rigorous architectural methodologies, *OmniAware* provides practical insights into the secure orchestration of mission-critical defence systems, thereby significantly advancing the field of secure cloud computing in defence contexts. This thesis positions itself as a contributing artefact within the broader architectural design initiative under the AWS-Capgemini collaboration, focusing on NATO-aligned capability modelling and secure, deployable system design.

2.2 NATO ARCHITECTURE FRAMEWORK VERSION 4

In order to build secure, modular and interoperable cloud-native infrastructures for modern military operations, a structured and formally governed architectural methodology is indispensable. Within this thesis, the NATO Architecture Framework Version 4 (NAFv4) is adopted not merely as a referential document, but as the foundational modelling framework to structure, trace and govern the system architecture. It offers the methodological rigour required for aligning mission planning with technical implementation across federated, multi-domain and sovereign defence platforms.

NAFv4 represents the latest iteration of NATO's enterprise architecture framework and introduces a paradigm shift from document-centric architecture towards formal, model-driven engineering. It builds upon the legacy of frameworks such as DoDAF and MODAF, but extends them by fully integrating formal meta-modelling languages such as the Unified Architecture Framework Domain Meta-Model (UAF-DMM) and ArchiMate. This model-centric philosophy enables semantic consistency, traceability and governance across all phases of the architectural lifecycle [53], [74].

At the core of NAFv4 is its viewpoint-based modelling structure. Viewpoints represent categorised stakeholder perspectives that guide the development of system views, each focused on specific architectural concerns. The five primary viewpoint classes are: Concept (NCV), Logical Specification (NLV), Service Specification (NSV), Physical Resource (NPV) and Architecture Foundation (NAV). These layers enable the decomposition of complex systems from strategic intent to deployable infrastructure, ensuring that high-level capability goals are methodically refined into technical designs and validated deployments.

The Concept Viewpoint defines operational goals, high-level capability groupings and mission scenarios. It provides the entry point for capability-driven planning and stakeholder alignment. The Logical Specification Viewpoint then describes functional interactions, logical processes and information flows independent of implementation technologies. The Service Specification Viewpoint shifts the focus towards modular and reusable services, interface definitions and orchestrations. Next, the Physical Resource Viewpoint grounds the architecture in concrete infrastructure components such as compute nodes, networks and deployment topologies. Finally, the Architecture Foundation Viewpoint provides meta-information, traceability artefacts, compliance annotations and mappings to external standards such as Unified Architecture Framework (UAF) and The Open Group Architecture Framework (TOGAF) [53].

NAFv4's viewpoint logic is systematised in its official Viewpoint-Model Matrix. This artefact structures the required model types across architectural layers and semantic intents, guiding modellers in developing traceable, semantically valid artefacts. Figure 2.1 illustrates the full matrix, which has been applied throughout this thesis as a methodological blueprint for architectural design.

To implement these models formally, this thesis employs the ArchiMate modelling language (version 3.2), which aligns directly with NAFv4's layered viewpoint structure [52]. ArchiMate supports the representation of business, application and technology layers and is enriched by constructs for strategy, motivation and physical deployment. It offers the expressiveness and traceability required to represent NATO-compliant architectures from capability maps to containerised deployments.

The modelling process is supported by the open-source tool **Archi** (version 5.5.0), which provides native support for ArchiMate 3.2, viewpoint-based organisation, layered visualisation and semantic validation. Compared to other modelling tools such as Sparx Enterprise Architect, Archi was selected due to its open and extensible modelling format, native ArchiMate 3.2 support, active community development and seamless integration with version control systems. These factors made it particularly well-suited for a proof-of-concept architecture that required transparency, portability and reproducibility across iterative modelling cycles. Archi allows modellers to construct artefacts aligned with NAFv4 logic, maintain traceability across viewpoints and document architectural assumptions in a collaborative, version-controlled environment.

To ensure methodological consistency, the *ArchiMate Modelling Guide for NAFv4* [74] was adopted. This guide, published by NATO's Architecture Capability Team, defines mappings between NAFv4 artefacts and ArchiMate element types. It provides detailed

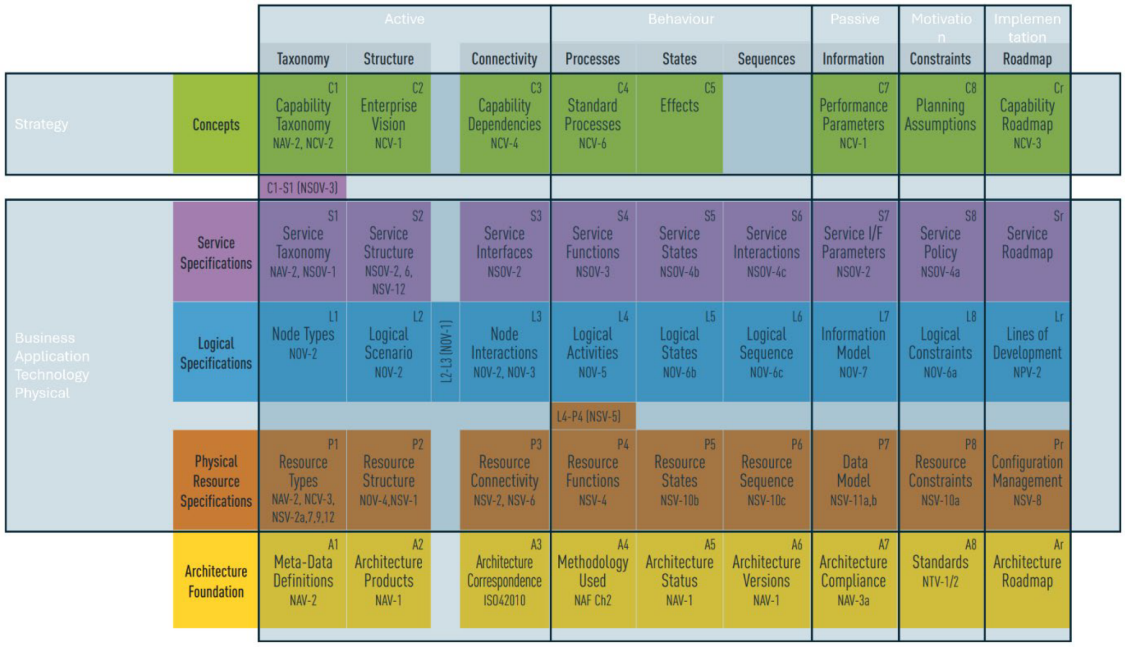


Figure 2.1: NATO Viewpoint-Model Matrix — Structuring Architectural Semantics [74]

specifications for modelling capabilities, interfaces, nodes and resources and defines visual conventions for mandatory versus optional elements.

NAFv4 Name	NAFv4 ArchiMate Name	ArchiMate Name
The name provided in the NAFv4 framework documentation.	The name of the specialized ArchiMate element created for the purpose described in this document.	The name of the 'parent' element from the ArchiMate specification from which the NAFv4 specialization is derived

Figure 2.2: Mapping Schema NAFv4-Archimate - Example [74]

Figure 2.2 shows a mapping example, in which ArchiMate elements such as `Application Component`, `Service Interface` or `Technology Node` are aligned with standardised viewpoint artefacts such as `NSV-4` or `NPV-6`. This mapping logic was implemented directly within Archi using dedicated folders and structured views per viewpoint class, allowing NATO-compliant decomposition of the *OmniAware* platform architecture.

To reinforce the traceability between strategic intent and operational execution, Figure 2.3 provides an example of a `C4 Viewpoint`, in which a capability defined in the NCV is semantically linked to operational activities defined in the NLV, closing the feedback loop between planning and execution.

By integrating NAFv4, ArchiMate and Archi in a tightly coupled methodology, this thesis achieves not only compliance with NATO modelling expectations, but also a practical workflow for building federated, sovereign and adaptable architectures. Each artefact — whether a capability map, logical service flow or deployment node — is part of a validated and reproducible model structure that supports auditability, lifecycle governance and architectural reuse.

In summary, NAFv4 is not merely referenced, but fully operationalised within this research. Its layered viewpoint system, formal artefact taxonomy and integration with

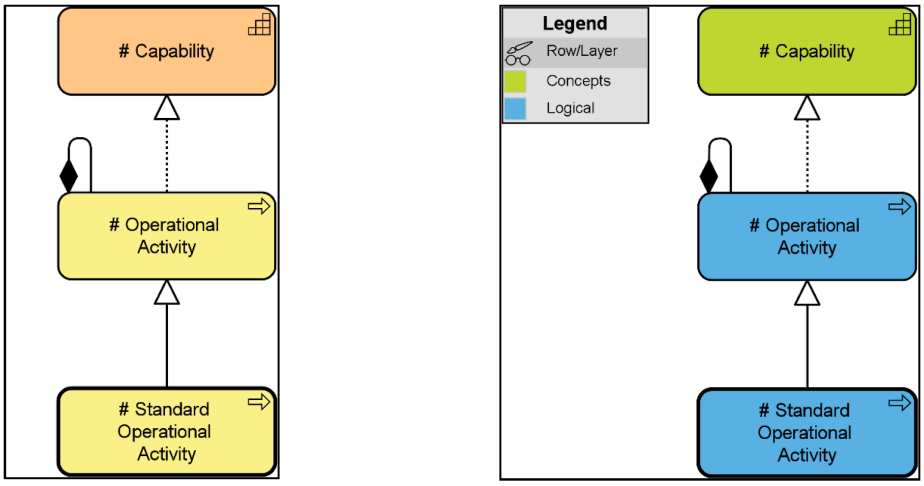


Figure 2.3: C4 Viewpoint, Mapping of NAFv4 Capability to Operational Activity - Example [74]

modelling standards like ArchiMate form the backbone of the *OmniAware* platform's architectural design. This ensures that the resulting system is not only technically feasible, but also methodologically grounded, traceable and aligned with alliance-level architectural doctrine.

From a historical perspective, the NATO Architecture Framework originated as a derivative of the U.S. Department of Defense Architecture Framework (DoDAF) and the UK's Ministry of Defence Architecture Framework (MODAF), both of which emphasised structured capability-based planning and enterprise-level interoperability. Earlier versions of NAF, particularly v3, were predominantly document-driven and lacked the semantic precision required for modern, automated and federated system design. As digital operations and coalition-based military engagements grew more complex, the need for an integrated, machine-processable architecture standard became evident [53].

NAFv4 addresses these needs by introducing a formal model-driven engineering paradigm, underpinned by the integration of the Unified Architecture Framework Domain Meta-Model (UAF-DMM) and the ArchiMate specification. The most prominent structural enhancement is the two-dimensional classification of architectural artefacts using "Subjects of Concern" (e.g. Capability, Service, Resource) and "Aspects of Concern" (e.g. Behaviour, Structure, Information, Roadmaps). Each artefact — such as `NSV-4` or `NPV-6` — is mapped along both dimensions to clarify its scope, intent and relationship within the broader system model.

This methodological matrix enables rigorous model design that supports semantic traceability between different abstraction levels. For example, a capability model (`NCV-2`) is explicitly linked to logical activities (`NLV-4`), service compositions (`NSV-5`) and physical node deployments (`NPV-6`), thereby ensuring continuity from strategic intent to implementation. These relationships are critical in defence architectures, where architectural auditability, reproducibility and compliance with alliance-wide governance mandates are required.

In multinational contexts — such as Federated Mission Networking (FMN), Joint All-Domain Command and Control (JADC2) or other coalition operations — NAFv4 provides a common semantic foundation for interoperable system architectures. It allows NATO member states and partners to model, align and integrate their national

systems while maintaining sovereignty and ensuring mission-specific configurations. This is especially relevant in cross-domain systems (cyber, space, etc.), where modularity, information assurance and policy enforcement must span organisational boundaries [82].

The use of formal viewpoint decomposition further enhances the transparency of architectural decisions. Viewpoints such as NCV (strategic), NLV (functional), NSV (service) and NPV (physical) are not isolated models, but semantically interlinked representations that support iterative refinement and validation. In this thesis, this layering has been used to derive consistent, NATO-aligned architectural structures for the *OmniAware* platform, enabling traceability from stakeholder objectives to technical design choices.

While other frameworks such as TOGAF or UAF also support structured architectural modelling, their emphasis and target audiences differ. TOGAF is predominantly focused on enterprise IT governance in civilian domains, with extensive flexibility but limited defence-specific structuring. UAF is more comprehensive, but lacks the NATO-specific artefact taxonomy and procedural guidance required for cross-nation military interoperability. NAFv4, in contrast, is specifically tailored to the needs of NATO-aligned defence organisations and explicitly addresses compliance, multi-layer traceability and coalition interoperability [51], [53].

By integrating the ArchiMate language and the open-source Archi modelling tool in line with the official NATO Modelling Guide, this thesis ensures that architectural artefacts not only follow a methodologically correct structure, but also meet tooling and exportability requirements for collaborative defence development. The modelling process was executed in Archi 5.5.0 using ArchiMate 3.2 profiles and applied folder structures per viewpoint category, following guidance in the NATO ArchiMate Modelling Guide [74].

The adoption of NAFv4 throughout this thesis establishes it as a scientific backbone for the architectural methodology employed. Rather than improvising design steps, all modelling decisions — from capability mapping to Kubernetes-based deployment strategies — are grounded in formal NATO-compatible artefact structures. This ensures that the architecture is not only technically coherent and reproducible, but also aligns with strategic priorities of coalition-based system development, sovereignty requirements and NATO digital transformation goals.

2.3 CONFIDENTIAL COMPUTING AND DATA SOVEREIGNTY

In an era of pervasive digitalisation, the secure handling of sensitive data has become both a technological and strategic imperative — particularly for public sector, critical infrastructure and defence domains. As adversaries exploit the expanding attack surface of distributed systems, conventional perimeter-based and software-centric security paradigms fall short in protecting data through its entire lifecycle.

Confidential computing and **data sovereignty** are two converging concepts that address this fundamental challenge by redefining the security model for cloud-native, coalition-based and geopolitically sensitive infrastructures.

2.3.1 *Confidential Computing*

Confidential computing refers to the use of hardware-based technologies to isolate and protect data while it is being processed. It relies on TEEs that reside within the CPU, providing runtime encryption, logical isolation and remote attestation. These

mechanisms ensure that the code and data within an enclave remain protected—even from privileged components such as the operating system (OS), hypervisor or virtual machine monitor (VMM). Unlike traditional security measures that focus on data *at rest* or *in transit*, confidential computing fills the remaining gap: data *in use* [35], [42].

TEEs offer a set of capabilities that extend trust into untrusted execution environments:

- **Memory encryption** ensures that enclave-resident data is encrypted in random-access memory (RAM), preventing exposure through memory scraping or direct memory access (DMA) attacks.
- **Code integrity** guarantees that only cryptographically measured code is allowed to execute in the enclave.
- **Remote attestation** enables external verifiers to cryptographically validate the enclave's origin, codebase and configuration, forming the foundation of decentralised trust.

Leading implementations include Intel Software Guard Extensions (Intel SGX), Advanced Micro Devices, Inc. (AMD) Secure Encypted Virtualisation — Secure Nested Paging (AMD SEV-SNP), Intel Trust Domain Extensions (Intel TDX) and AWS Nitro Enclaves, each supporting different trust models and levels of programmability. While Intel Software Guard Extensions (Intel SGX) enables application-level enclaving with minimal Trusted Computing Base (TCB), SEV-SNP allows the isolation of entire VMs, facilitating lift-and-shift migrations of legacy defence systems into confidential environments. Nitro Enclaves provide a balance of compatibility and attestation support within cloud-native ecosystems [42].

Confidential computing underpins secure computing in coalition and adversarial contexts. In defence, enclaves enable secure AI inference on classified models, confidential digital twin simulations, secure federated learning across nations and attested Command and Control (C2) systems deployed across tactical edge nodes. These use cases are grounded in NATO's vision for FMN, which demands resilient, interoperable and cryptographically verifiable computing domains in multi-national operations. **C2** refers to the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of a mission. It includes the systems, processes and communication structures required for planning, decision-making and mission execution across all operational domains.

Crucially, the use of enclaves in such systems supports a shift from perimeter trust to cryptographic trust. This is particularly relevant for federated infrastructures where workloads cross national and organisational boundaries. Enclaves create **sovereign execution environments**: verifiable compute zones that enforce national data protection policies independently of infrastructure ownership [50].

2.3.2 *Data Sovereignty*

Data sovereignty, in turn, extends the principle of territorial sovereignty to digital information. It is the ability to define and enforce access, processing and storage policies over data throughout its lifecycle and regardless of its physical location. From a regulatory standpoint, it intersects with jurisdictional frameworks such as the GDPR, cloud assurance schemes such as the European Cybersecurity Certification Scheme (EUCS) and industrial sovereignty initiatives such as Gaia-X [50], [66].

Technical enforcement of data sovereignty is increasingly difficult in multi-cloud or coalition-operated environments where data can traverse multiple jurisdictions. Confidential computing resolves this by providing verifiable guarantees that data remains under cryptographic control — even during processing — regardless of where it physically resides or who manages the infrastructure. This is essential for the European defence sector, where NATO operations often rely on global communications and information systems platforms hosted across multiple jurisdictions [10], [43].

A sovereign cloud system leveraging TEEs can ensure that mission data is encrypted throughout its lifecycle, with remote attestation used to verify that only authorised enclaves with audited workloads may access or process it. This transforms traditional notions of cloud trust, replacing contractual and legal constructs with measurable, enforceable hardware-rooted guarantees [42], [66].

These capabilities are already reflected in evolving NATO policy. *AC322-D(2021)0032-REV1* outlines requirements for cloud-based handling of NATO-classified data. It stipulates that data must be processed in environments that provide strong isolation, support auditability and enable secure lifecycle management. Similarly, STANAG 4774 and 4778 establish standards for confidentiality metadata and digital bindings to ensure data remains linked to its classification and authorisation policies [5], [8]. TEEs and confidential computing fulfil these requirements both technically and operationally [43].

Moreover, emerging cloud-native confidential orchestration platforms such as Constellation or confidential containers enable enclave-based policies to be enforced at scale in Kubernetes clusters. Combined with policy engines, Public Key Infrastructure (PKI) anchors and decentralised key distribution, they facilitate attested service meshes where every microservice can be verified at runtime. This enables not only secure Continuous Integration and Continuous Delivery (CI/CD) pipelines, but also real-time mission workloads such as sensor fusion, video stream analytics or battlefield telemetry to be executed within fully sovereign boundaries [42].

Beyond tactical scenarios, confidential computing supports strategic autonomy in digital capability development. By ensuring that software artefacts, mission algorithms and data models remain protected — even when trained, compiled or evaluated on foreign infrastructure — nation-states can develop next-generation digital capabilities without sacrificing confidentiality or control. This is particularly relevant for AI model training, cyberdefence algorithms and predictive logistics systems.

In the long-term, confidential computing and data sovereignty are not just enablers of secure infrastructure — they are the cornerstones of digital deterrence. They enable trust to be built on measurable properties, not institutional assumptions. For future military clouds, they ensure that sovereignty is preserved even in compromised, contested or coalition-controlled environments. For NATO and the EU, they offer a technological path toward resilient, federated and sovereign digital power projection.

The convergence of confidential computing and data sovereignty redefines digital trust at the root of computing. Together, they enable a transformation from infrastructure-centric to policy-centric security. They allow defence organisations to process sensitive data securely on any infrastructure, build verifiable coalitions and maintain strategic control in a contested, multipolar cyber domain. As such, they are indispensable to any future-ready military cloud architecture and constitute key pillars of the *OmniAware* platform.

Several initiatives at the national and alliance level are already exploring or operationalising the capabilities described above. One example is the AWS European Sovereign Cloud (ESC), which introduces physically and logically separated cloud regions to support public sector workloads within the EU. These regions are independently

operated and staffed, ensuring that all data processing, support and operations remain under EU jurisdiction. Importantly, AWS Nitro Enclaves extend this offering by enabling confidential computing at the virtual machine boundary, allowing sensitive workloads to be executed in memory-encrypted, attested compute environments [65].

At the NATO level, the Federated Mission Networking (FMN) initiative serves as the alliance's reference architecture for interoperable command and control systems. FMN Spiral Specifications have increasingly included provisions for data tagging, cross-domain guard integration and secure multi-party information sharing. Confidential computing provides the technical foundation to operationalise such requirements by enabling verifiable enclave-based processing nodes that can operate even in untrusted coalition infrastructure [53].

In the German context, the Bundeswehr's cloud transformation strategy, as reflected in the BWI's multi-cloud roadmap, explicitly highlights the need for workload isolation, policy-enforceable encryption and cross-domain data sharing controls. Confidential computing directly supports these objectives. For example, deploying SEV-backed virtual machines in tactical private cloud infrastructure allows mobile units to execute mission logic securely, even in environments where physical security of the compute node cannot be guaranteed. Key command applications — such as secure digital maps, predictive logistics tools and encrypted mission reports — can all be hosted within attested enclaves, ensuring compliance with the Bundeswehr's own ZDV policies and NATO interoperability requirements.

The trust model of confidential computing depends heavily on the attestation infrastructure and key release protocols. A typical *remote attestation flow* involves the following steps:

1. The application requests an enclave instance (e.g. via Nitro Enclave, SGX or SEV-enabled hypervisor).

2. Upon launch, the enclave generates a *measurement*, which is a cryptographic hash of its code and configuration.

3. This measurement is signed by a hardware root of trust (e.g. Intel EPID or AMD's Platform Security Processor) and issued to a *verifier*, typically a key management service (KMS) or policy enforcement point.

4. The verifier evaluates the measurement against an allowlist (e.g. pre-approved mission applications) and policy metadata.

5. If the evaluation succeeds, the verifier instructs a KMS to release workload-specific decryption keys (e.g. for credentials, mission data or AI models) into the enclave's memory space.

This process ensures that only verified and authorised workloads receive the cryptographic materials necessary for their execution. The keys never leave the enclave or become accessible to the host system. This is particularly valuable in military deployment scenarios, where the compute infrastructure may be operated by third-party logistics providers, coalition nations or even located in adversarial zones [56].

Key release protocols can be further extended with policy constraints. For example, certain keys may only be released if the attestation originates from a platform within a specific NATO country, is geofenced to a particular base or is valid within a predefined mission time window. These policies can be enforced by integrating the attestation

infrastructure with identity and policy engines such as Secure Production Identity Framework For Everyone (SPIFFE)/SPIFFE Runtime Environment (SPIRE), `HashiCorp Vault` or confidential KMS instances [61].

A prominent example of confidential computing is AMD's Secure Encrypted Virtualisation Secure Nested Paging (SEV-SNP), a hardware-enforced TEE specifically designed to securely isolate entire VMs from potentially compromised hypervisors and cloud providers. SEV-SNP ensures that data and workloads remain confidential and integral by encrypting VM memory with individual AES-128 encryption keys that are generated and managed directly by the integrated AMD-SP [56].

The core root of trust for AMD SEV-SNP lies within this Secure Processor, a dedicated hardware security co-processor embedded in the AMD CPU. The Secure Processor (SP) handles critical operations including key generation, cryptographic measurement of VM components and the issuance of attestation reports. At the hardware level, each AMD CPU integrates a unique Chip Endorsement Key (CEK), permanently stored in chip fuses and never exposed externally. Derived from the CEK, the Versioned Chip Endorsement Key (VCEK) is used specifically to sign attestation reports, verifying both the integrity and authenticity of the VM environment, including the firmware, boot loader and operating system kernel components [61].

To maintain memory integrity, SEV-SNP implements the Reverse Map Table (RMP), a hardware-managed structure preventing unauthorised write operations to the memory regions of secured VMs. Additionally, SEV-SNP introduces Virtual Machine Privilege Levels (VMPLs), enabling fine-grained privilege separation within VMs. This mechanism is particularly beneficial for sensitive operations such as virtual TPM implementations.

The attestation mechanism in SEV-SNP is integral to its security model, allowing external entities to validate the trustworthiness of a VM environment. Initially, a Launch Measurement, consisting of cryptographic hashes of firmware, kernel images and kernel parameters, is generated during VM startup. This measurement is securely stored and verified against reference values each time the VM boots. During the attestation process, a VM requests an attestation report from the SP, which is then cryptographically signed using the VCEK. External parties validate this signed report against AMD's public certificate chain, thus ensuring the authenticity and integrity of the VM's execution environment [61].

Key distribution in SEV-SNP is securely facilitated through two primary workflows: dm-verity for integrity-only scenarios and dm-crypt for both integrity and confidentiality. In the integrity and confidentiality scenario, a Diffie-Hellman (DH) key exchange is integrated into the attestation process. Here, the VM generates an ephemeral DH key pair and includes the public key in the attestation report. Upon validation, the external key management service (KMS) securely transmits an encrypted disk encryption key, which the VM decrypts using the shared DH secret. Consequently, this method ensures secure key provisioning by exclusively granting access to attested VMs, thereby preventing unauthorised disclosure of sensitive cryptographic material [61].

Figure 2.4 effectively and visually illustrates the attestation and key management workflow to describe the cryptographic key provisioning process, highlighting interactions between the VM, AMD SP and external verifying entities.

Initially, the VM generates an ephemeral Diffie-Hellman (DH) key pair, embedding its public key into an attestation report requested from the AMD-SP. The SP measures critical boot components securely via the **Open Virtual Machine Firmware (OVMF)**, which is a UEFI firmware implementation designed for virtual machines, capable of securely storing and verifying cryptographic hashes (Kernel, Initramfs, Kernel parameters) and cryptographically signs these measurements using the Versioned Chip Endorsement

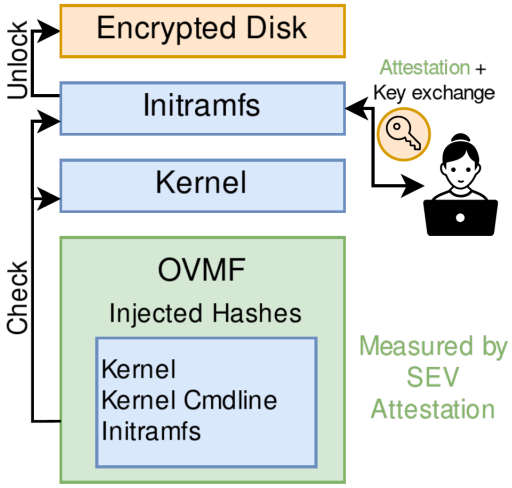


Figure 2.4: AMD SEV-SNP - Attestation and Key Management Workflow [61]

Key (VCEK). An external entity, typically the VM owner or a Key Management Service (KMS), verifies the attestation report using AMD's public key infrastructure. Upon successful validation, the KMS securely transmits the encrypted disk key, encrypted using the shared DH secret derived from the VM's public key, to the attested VM. The VM then uses the derived shared secret to decrypt the disk encryption key and unlock the encrypted disk, ensuring secure and authenticated boot processes [61].

Figure 2.5 provides a comprehensive visual explanation of memory protection mechanisms used in SEV-SNP.

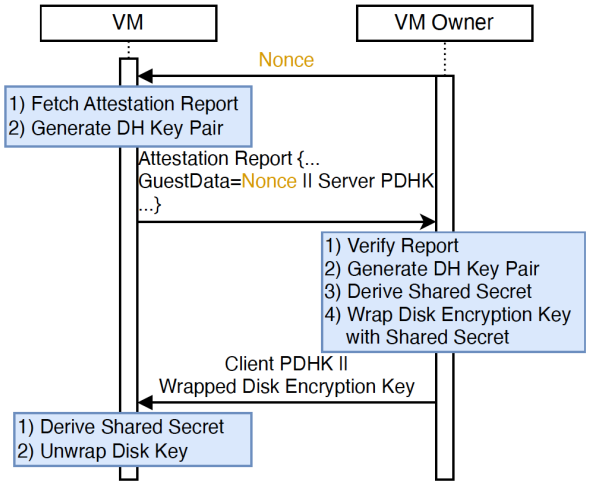


Figure 2.5: AMD SEV-SNP - Memory Protection and Key Provisioning Mechanisms [61]

Initially, the VM initiates a remote attestation request by generating a nonce which is a unique, randomly generated number used only once to prevent replay attacks, into an attestation report requested from the AMD-SP and an ephemeral Diffie-Hellman key pair, embedding the public component **Platform Diffie-Hellman Key** (PDHK) into the attestation report provided by the AMD-SP. The PDHK is an ephemeral,

platform-generated public key utilised in the DH key exchange protocol during the remote attestation process of AMD SEV-SNP environments. The VM Owner verifies the authenticity and integrity of the attestation report, subsequently generating their own ephemeral DH key pair to derive a shared secret. This secret is used by the VM Owner to securely encrypt (wrap) the disk encryption key. The VM receives this encrypted key, independently derives the shared secret from its PDHK, unwraps the encryption key and securely mounts the encrypted storage. This mechanism guarantees that disk encryption keys are securely provisioned exclusively to validated and trustworthy VM instances [61].

Having explored the foundational role of TEEs — particularly AMD SEV-SNP — in enabling remote attestation and enforcing cryptographic trust boundaries, it is now essential to broaden the focus toward **Edge Computing**. This shift is crucial, given that many mission-critical defence scenarios rely on geographically distributed, intermittently connected and physically exposed compute nodes. In this context, TEEs are not only a matter of secure computation but become enablers for trust in hostile or coalition-controlled edge environments. Before diving deeper, we refer to **Figure 2.6**, which compares TEE implementations — Intel SGX, AMD SEV and ARM TrustZone — based on capabilities, isolation levels and architectural characteristics relevant also for edge deployments [42].

	Intel SGX	**AMD SEV**	**ARM TrustZone**
Processor Architecture	x86-64	x86-64	ARM
Secure Storage	Yes	No	No
Remote Attestation	Yes	Yes	No
Memory Isolation	Yes	Yes	Yes
Memory Size Limit	Up to 128 MB EPC	Up to available RAM	3–5 MB
Trusted I/O	No	Yes	Yes
Operation Level	Ring 3	Ring 0	Ring -2
Compatibility	Windows	Linux-based VMs and hypervisors	Android, Linux
SDK	Provided	Not required	Provided
Memory Integrity Protection	Yes	No	No
Multithreading	Yes	Yes	No
Applications	Simple and security-sensitive	Complex and legacy	Lightweight

Figure 2.6: Comparison of TEE architectures [42]

It is important to note that especially the depiction of AMD SEV in the original figure is outdated, as it does not reflect the enhancements introduced with SEV-SNP, such as support for nested paging, the Reverse Map Table (RMP) and fine-grained privilege separation via virtual machine privilege levels VMPLs. These improvements significantly elevate its applicability in zero-trust edge scenarios.

For edge environments orchestrated via Kubernetes, such as in the deployment of microservice-based tactical applications, the most viable TEE choices are Intel SGX, Intel TDX and AMD SEV-SNP. ARM TrustZone, while ubiquitous in mobile contexts, lacks the granular attestation and isolation properties required for container-based workloads. Intel SGX supports application-level enclaving with a minimal TCB, making it suitable for lightweight and high-assurance workloads. However, its limited enclave memory and need for code refactoring constrain its usability [42].

In contrast, AMD SEV-SNP enables lift-and-shift of legacy services through VM-level isolation, aligning better with the operational realities of cloud-native edge clusters. This architectural fit is further underscored by recent advancements such as *Trusting the Cloud-Native Edge* [60], which proposes a secure enrolment architecture for edge worker nodes based on TPM attestation and RBAC-driven policy enforcement. However, a decisive limitation arises when shifting the focus from fog to true edge deployments: current market availability still lacks support for SEV-SNP-enabled edge hardware. While theoretically suitable, SEV-SNP remains confined to server-grade platforms, given the physical constraints, thermal design power and form factor requirements that edge-class devices cannot yet accommodate. Similarly, Intel TDX is bound to 4th Gen Xeon Sapphire Rapids processors, which are likewise ill-suited for decentralised, energy-efficient edge environments [42].

Consequently, attention is shifting toward alternative architectures that prioritise energy efficiency, embedded compatibility and extensibility. One notable example is the NVIDIA Jetson product family, which integrates a native TEE via OP-TEE. While OP-TEE does not natively support remote attestation, extensions and third-party frameworks exist to augment this capability. However, the security guarantees of such extensions remain a subject of ongoing research. Their reliability, verifiability and suitability under adversarial or disconnected conditions must be evaluated with particular care — especially in light of the stringent regulatory and mission-critical demands of defence-grade edge infrastructure, including secure key release, disconnected attestation workflows and compliance with cross-domain policies.

Determining the appropriate implementation layer for enclave support in defence-oriented edge deployments is a critical architectural decision. This choice depends on the required isolation granularity, the portability of workloads and the anticipated level of adversarial access. In practical terms, SEV-SNP offers a compelling compromise by enabling compatibility with containerised workloads while upholding a strong hardware-rooted trust anchor.

In multi-domain operations (MDO), data flows across services, domains and organisational boundaries — raising the challenge of establishing federated trust without compromising sovereignty. Confidential computing enables a distributed trust model in which participating nodes independently prove their security posture and eligibility to access certain classes of mission data. In this sense, attestation becomes a programmable form of **operational authorisation**, bridging the gap between cryptography and command policy.

For instance, a sensor platform from Nation A may provide video feeds into a joint coalition AI model hosted in a cloud enclave operated by Nation B. The enclave is attested and only if its measurement matches, a trusted configuration does Nation A allow its encrypted data stream to be decrypted and processed. Conversely, any attempt to process the data outside of this enclave configuration would result in key withholding — preventing data leakage or policy violations.

These mechanisms are also compatible with tactical edge deployments. Modern edge devices increasingly support embedded TEEs, allowing secure execution of in-

ference engines, mission analytics and encryption modules directly on unmanned platforms, forward-operating bases or mobile units. Paired with confidential boot and runtime attestation, such systems offer unprecedented levels of verifiability and policy control, even in disconnected or contested environments.

Confidential computing and data sovereignty are not just enablers of secure systems as they have significant operational implications and strategic relevance. They represent a strategic inflection point in how military and government organisations architect their digital platforms. They allow decision-makers to:

- Shift trust from infrastructure to verifiable computation.
- Operationalise security policy through cryptographically enforced runtime constraints.
- Distribute computing workloads across heterogeneous and coalition-owned infrastructure without losing sovereignty.
- Enable multinational collaboration without requiring full data sharing.
- Satisfy compliance with national and international regulations through measurable, auditable technical means.

Confidential computing and data sovereignty enable defence organisations to establish cryptographic perimeter controls that are resilient to jurisdictional ambiguity, provider-side compromise or insider threat. This transformation of the military cloud stack is not merely evolutionary — it is foundational to preserving decision superiority and secure coalition interoperability in the coming decades.

The subsequent chapter transitions from architectural modelling to implementation design, detailing the runtime and operational environment required to realise the previously defined services. It outlines how the abstract platform architecture is mapped to concrete cloud, edge and hybrid deployment topologies, enabling mission-compliant execution under real-world constraints.

2.4 CLOUD, FOG AND EDGE COMPUTING IN DEFENCE

The increasing digitisation and decentralisation of military infrastructures across all operational domains — land, air, sea, cyber and space — have made the role of distributed computing paradigms more critical than ever. As defence forces confront heterogeneous threat environments, intermittent connectivity and constrained physical infrastructure, the traditional reliance on centralised information systems is being replaced by hybrid and multi-layered approaches to data processing. These challenges necessitate architectural paradigms that can provide not only computational elasticity and resilience but also strong guarantees of mission assurance and information superiority across strategic, operational and tactical levels.

In this context, cloud computing has emerged as a dominant model for the orchestration of scalable, on-demand and geographically abstracted computing resources. Cloud platforms enable the efficient pooling of compute and storage capacity, as well as centralised orchestration and policy control. In defence scenarios, this allows for real-time operational planning, collaborative intelligence fusion and the delivery of command-and-control (C2) services across widely dispersed forces. Nonetheless, such infrastructures typically reside in remote hyperscale data centres, introducing operational fragilities —

especially in latency-sensitive, bandwidth-constrained or disconnected environments. These constraints become particularly relevant in scenarios involving mobile units, contested electromagnetic spectrums or adversarial conditions, where uninterrupted cloud uplinks cannot be assumed.

To mitigate these challenges, fog computing introduces a distributed intermediary layer that extends cloud-native services towards the periphery of the network. It acts as a processing continuum between the core and the edge, offering regional compute capacity with reduced round-trip delays. Fog nodes can be deployed on forward-operating bases, mobile platforms or maritime vessels, enabling low-latency execution of mission workflows, pre-processing of sensor data or even tactical orchestration of autonomous platforms. This architectural layer is especially useful in enabling near-real-time services that require situational responsiveness without full reliance on backhaul connections to centralised clouds.

Edge computing, in turn, represents the most decentralised paradigm, situating computational resources directly on sensors, platforms and actors operating at the tactical frontier. Unlike fog computing, which preserves a regional scope, edge computing executes data processing in close physical proximity to the data source — often within the same embedded system. This proximity drastically reduces latency, enhances responsiveness and ensures that critical analytics such as threat detection, local autonomy and dynamic reconfiguration — can be performed even in fully disconnected or denied environments. In modern defence architectures, edge computing enables AI-enabled battlefield analytics, encrypted local data fusion and the secure execution of containerised microservices within ruggedised devices and mobile platforms.

Each of these computing paradigms offers unique benefits, but only through their integration can the full spectrum of operational requirements be addressed. Defence-grade architectures increasingly follow a hierarchical model in which cloud, fog and edge form a distributed processing fabric. This fabric ensures that workloads are assigned to the optimal layer based on data sensitivity, mission urgency, computational intensity and network availability. The resulting architecture allows for asymmetric decision superiority, enabling military actors to act faster, with greater resilience and better-informed situational awareness than their adversaries.

Furthermore, these paradigms form the technological substrate upon which confidential computing, sovereign execution and coalition interoperability must be built. Secure workload migration, attested microservices and policy-based compute routing all depend on the existence of a distributed runtime layer capable of hosting such workloads across cloud, fog and edge tiers. Consequently, a thorough understanding of the design principles, technical constraints and operational implications of these paradigms is imperative for any future-ready military information system. This section elaborates on the architectural interplay between these domains and outlines the role each plays in enabling secure, scalable and interoperable defence platforms.

2.4.1 *Virtualisation*

At the technological core of cloud computing lie virtualisation and abstraction mechanisms that decouple software workloads from underlying physical infrastructure. Virtual machines (VMs) orchestrated by hypervisors such as Kernel-based Virtual Machine (KVM), Xen or VMware ESXi, enable the secure co-location of multiple operating systems on a single physical host. This isolation not only increases resource efficiency but also lays the foundation for scalable, secure multitenancy — a critical requirement for

defence-grade cloud infrastructure. Beyond virtual machines, container-based technologies such as Docker and orchestration systems like Kubernetes have further advanced cloud-native paradigms. Containers encapsulate application logic and dependencies into isolated runtime environments that can be deployed across heterogeneous platforms, thereby facilitating the modularisation of defence applications and microservice-based architectures [56], [57].

Virtualisation represents a pivotal technology for the abstraction and efficient utilisation of physical computing resources. By leveraging hypervisors, physical servers can be partitioned into multiple isolated VMs, each operating with individual configurations of CPU, RAM, storage and network resources. Two principal hypervisor types exist: Type-1 hypervisors are installed directly on the hardware (bare-metal), offering higher performance and better suitability for data centre operations, whereas Type-2 hypervisors run atop a host OS, typically for desktop or development environments [56].

To provision, configure and manage these resources reliably and at scale, cloud computing relies heavily on Infrastructure-as-Code (IaC) techniques. Declarative languages such as Terraform, AWS CloudFormation or Ansible enable version-controlled, repeatable and policy-driven deployments of infrastructure components, significantly reducing operational complexity and deployment risks. In defence environments, where consistency, auditability and automation are paramount, IaC becomes indispensable. Combined with DevSecOps pipelines and automated compliance checks, this foundational technology enables military cloud systems to remain resilient, auditable and agile under dynamic operational conditions [56], [57].

2.4.2 *Cloud Computing*

Cloud computing delivers computing services — including servers, storage, databases, networking, software and analytics — over the internet, offering faster innovation, flexible resources and economies of scale. Users typically pay only for the cloud services they use, helping to lower operating costs and run infrastructure more efficiently [2].

In the defence sector, cloud computing enables the centralisation of data and applications, providing authorised personnel with access to critical information from any location. This centralisation supports collaborative operations across different branches and allied forces, enhancing coordination and decision-making.

Cloud computing represents the foundational paradigm for elastic and on-demand provisioning of computational and storage resources in modern military infrastructures. Defined by the National Institute of Standards and Technology (NIST) as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources," cloud computing enables mission-critical applications to dynamically scale according to operational requirements [2]. Within the defence context, cloud computing delivers strategic advantages across multiple layers of command, control and intelligence operations — supporting data aggregation, secure mission planning, joint intelligence sharing and logistics optimisation.

Three service models — infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) — enable different levels of abstraction and control. In defence settings, IaaS is particularly relevant as it allows fine-grained control over workloads while leveraging cloud-native scalability and resilience. Deployment models such as private, public, hybrid or community clouds offer flexible options for balancing sovereignty, security and scalability [36].

The use of open-source platforms such as OpenStack further extends virtualisation into scalable and modular HPC infrastructures. OpenStack orchestrates compute, network and storage resources and supports advanced features like Non-unified Memory Access (NUMA)-aware scheduling, Single Root I/O Virtualisation (SR-IOV) for near-native I/O performance and containerised workloads in edge or cloud environments [38]. In confidential HPC scenarios, paravirtualisation and hardware-assisted memory management enable secure workload execution with minimal performance overhead, even in virtualised environments [22].

The interplay between virtualisation and cloud computing enables defence organisations to transition from static, siloed systems toward agile, software-defined and policy-enforceable infrastructures. This evolution is essential to support emerging requirements in confidentiality, interoperability and operational autonomy across coalition and sovereign environments.

In practical terms, cloud environments abstract physical hardware into software-defined pools of resources, managed via orchestration platforms that support IaC and DevSecOps workflows. For military cloud deployments, this means rapid instantiation of mission-specific environments, policy-based deployment of confidential workloads and real-time replication of strategic datasets across redundant availability zones. Cloud-native technologies such as container orchestration (e.g. Kubernetes), service meshes and CI/CD pipelines facilitate modularity and rapid adaptability — core requirements in dynamic battlefield conditions.

Despite these benefits, the centralised nature of hyperscale clouds also introduces operational limitations. In high-threat environments with unreliable or denied connectivity, dependence on distant data centres for mission execution creates unacceptable latency and availability risks. Moreover, defence missions require sovereign control over workload execution and data lifecycle management — constraints that traditional public cloud deployments do not natively satisfy. As a result, the emergence of sovereign cloud initiatives — such as the AWS ESC — seeks to reconcile the benefits of cloud elasticity with strict jurisdictional and policy-based control over digital assets [65].

The AWS ESC is designed to meet the specific needs of public sector customers in the EU, ensuring that data remains within the jurisdiction of the EU and is subject to its legal frameworks. This initiative highlights the growing importance of data sovereignty in cloud computing, particularly for defence applications where sensitive information must be protected from external threats and regulatory uncertainties [65].

From a capability standpoint, military cloud environments must enforce compliance with mission assurance standards, such as the NATO FMN Spiral Specifications and national cyber resilience frameworks. This necessitates the integration of secure enclave technologies (e.g. Nitro Enclaves or SEV-SNP-based VMs), policy-driven key management infrastructures and attestation-enabled service meshes that allow only verified workloads to access classified resources. Thus, while cloud computing offers the highest degree of elasticity, its defence-grade applicability depends entirely on the deployment of confidentiality, sovereignty and zero-trust enhancements that align with coalition-level operational and regulatory expectations [43], [53], [82].

2.4.3 *Fog Computing*

Fog computing extends cloud computing capabilities to the edge of the network, enabling data processing closer to the data source. This paradigm reduces latency and

bandwidth usage by processing data locally rather than transmitting it to centralised data centres [84].

For military operations, fog computing is particularly beneficial in environments with limited connectivity or where real-time data processing is crucial. By processing data at or near the source, such as on a battlefield or in remote locations, fog computing supports timely decision-making and reduces the reliance on constant connectivity to central servers.

Fog computing has emerged as a critical paradigm in modern distributed computing architectures, particularly in defence environments where latency, bandwidth and data sovereignty are paramount. It acts as an intermediary layer between the cloud and edge devices, addressing the inherent limitations of both edge and cloud computing by providing decentralised processing capabilities closer to data sources [58].

Unlike conventional cloud-centric models, where data is transmitted to distant data centres, fog computing leverages geographically distributed nodes — such as routers, base stations and gateways — to process, analyse and store data locally or regionally. This proximity to end devices dramatically reduces transmission delays, enhances bandwidth efficiency and facilitates real-time or near-real-time decision-making, which is crucial for mission-critical military applications.

In a defence context, the relevance of fog computing is particularly evident in scenarios requiring stringent requirements for low-latency communication, high availability and secure operations across potentially intermittent and constrained network environments. Applications include tactical command and control systems, autonomous vehicle coordination and sensor networks for battlefield surveillance, where instantaneous data processing can be life-saving [67].

Nevertheless, fog computing environments introduce complex challenges, especially regarding resource management, security and orchestration. Given the heterogeneity of fog nodes — ranging from lightweight devices with limited resources to powerful edge servers — task offloading and resource allocation require sophisticated optimisation strategies. Traditional cloud-oriented algorithms are insufficient due to their assumptions of abundant and homogeneous resources.

Recent research proposes heuristic and metaheuristic approaches, such as the Multi-Objective Firefly Algorithm (MFA), to address these challenges by optimising trade-offs between conflicting Quality of Service (QoS) parameters, notably energy consumption and transmission delay [58]. The MFA dynamically assigns computational tasks based on real-time assessments of device availability, workload characteristics and network conditions, achieving significant improvements in resource utilisation and system responsiveness.

Security remains a foundational concern within fog computing. The decentralised nature of fog architectures expands the attack surface, making nodes vulnerable to unauthorised access, data breaches and Denial of Service (DoS) attacks. Unlike traditional perimeter-based defences, securing fog environments requires context-aware, distributed security mechanisms capable of autonomous detection, mitigation and recovery. Recent advancements advocate for the integration of AI-driven security orchestration frameworks to automate threat response across heterogeneous fog infrastructures, especially within 5G and Beyond 5G (B5G) ecosystems [67].

Furthermore, maintaining data sovereignty and operational security within multinational defence coalitions imposes strict requirements on how and where data is processed within fog nodes. This necessitates the deployment of Trusted Execution Environments (TEEs) and hardware-based confidential computing techniques to safeguard sensitive information during computation, even in potentially untrusted environments.

The strategic integration of fog computing into connected defence platforms enhances operational agility, resilience and security. It enables distributed intelligence across the tactical edge while adhering to the fundamental principles of data sovereignty, security-by-design and interoperability — cornerstones for next-generation multinational military operations.

2.4.4 *Edge Computing*

Edge computing involves processing data at the periphery of the network, nearer than fog computing to the data source, rather than relying on a centralised data-processing warehouse. This approach minimises latency and bandwidth use, which is critical for applications requiring real-time processing and response [15].

In defence applications, edge computing enables devices such as drones, sensors and autonomous vehicles to process data locally, allowing for immediate analysis and action. This capability is vital in scenarios where rapid response times are essential and connectivity to central data centres may be intermittent or unavailable.

Building upon the decentralised processing capabilities enabled by fog computing, edge computing constitutes the most granular manifestation of distributed computing architectures. While fog computing aggregates and processes data within intermediate nodes such as gateways and regional servers, edge computing further advances this paradigm by embedding computational resources directly at or near the data source [15].

Edge computing represents a strategic enabler for time-sensitive and autonomy-driven military operations by minimising the distance between data generation, processing and actionable decision-making. Tactical platforms such as unmanned aerial vehicles, battlefield sensor networks and autonomous ground vehicles benefit significantly from edge architectures, where even minimal latencies in data processing can determine mission success or failure [69].

Fundamentally, edge computing decentralises not only computation but also storage and control, creating a distributed intelligence landscape that enhances operational resilience and reduces reliance on vulnerable communication backhauls. By performing data preprocessing, anomaly detection and initial analytics locally, edge devices enable the selective transmission of high-value, prefiltered information to higher-echelon fog or cloud infrastructures [59]. This selective data propagation mitigates bandwidth saturation, improves energy efficiency and enhances the robustness of situational awareness frameworks under adversarial conditions.

The architectural design of edge computing environments in defence contexts must account for resource constraints, intermittent connectivity and heightened security requirements. Lightweight machine learning models, often optimised for low-power inference through pruning, quantisation and hardware acceleration, are increasingly deployed to perform sophisticated analytics at the tactical edge [73]. For instance, field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) have been demonstrated to significantly accelerate neural network inference on edge platforms, achieving multi-fold improvements in latency and energy consumption relative to general-purpose processors [72].

From a security perspective, edge nodes represent critical vulnerabilities within the operational architecture due to their exposure to physical tampering and cyber threats. Advanced security mechanisms such as hardware-based Trusted Platform Modules (TPM), secure boot protocols, remote attestation and confidential computing via Trusted Execution Environments (TEEs) are indispensable for ensuring data integrity, confiden-

tiality and trustworthiness of computations performed at the edge [60]. These measures provide cryptographic guarantees that uphold mission assurance even in contested electromagnetic environments.

Moreover, the emergence of 5G and B5G networks acts as a catalyst for extending edge capabilities by delivering ultra-reliable, low-latency communication (URLLC) and massive machine-type communications (mMTC). The tight integration of edge computing with advanced wireless infrastructures enables distributed orchestration of computational tasks, seamless mobility support and resilient mission continuity across dynamic and geographically dispersed operational theatres [67].

In synthesising computation, storage, analytics and security at the data origin, edge computing transforms tactical platforms into intelligent, autonomous and sovereign digital entities. It plays an indispensable role in next-generation defence concepts such as Multi-Domain Operations (MDO), Joint All-Domain Command and Control (JADC2) and the Federated Mission Networking (FMN) vision of NATO. By complementing cloud and fog layers, edge computing forms the foundation for resilient, interoperable and secure connected defence infrastructures capable of operating effectively even under conditions of degraded communications or adversarial disruption.

The integration of virtualisation, cloud, fog and edge computing creates a robust and flexible computing environment tailored to the unique demands of defence operations. Virtualisation provides the foundational layer for resource abstraction and isolation. Cloud computing offers scalable and centralised resources for data storage and application deployment. Fog computing bridges the gap between the cloud and the edge, enabling localised processing and decision-making. Edge computing empowers devices at the network's periphery to operate autonomously and efficiently.

Together, these technologies support a comprehensive computing infrastructure that enhances operational effectiveness, resilience and adaptability in diverse and challenging military environments.

Major cloud service providers, such as AWS, have increasingly recognised the critical requirements of defence and public sector organisations operating across cloud, fog and edge environments. AWS has developed specialised services and deployment models that align with the operational constraints and sovereignty requirements inherent in defence infrastructures.

At the cloud layer, AWS GovCloud (US) and the forthcoming AWS ESC exemplify dedicated regions designed to meet stringent regulatory, compliance and operational control standards [65]. These regions ensure that mission-critical data remains within controlled jurisdictions, providing foundational capabilities for classified and coalition-based workloads.

For fog computing scenarios, AWS Outposts extends the AWS cloud model into on-premises and forward-operating locations, enabling the deployment of native AWS services with reduced latency and local data processing capabilities. By deploying Outposts at regional bases or naval vessels, defence organisations can achieve operational resilience and situational responsiveness without relinquishing control over sensitive data [65].

At the tactical edge, AWS Snowball Edge and AWS Snowcone devices provide ruggedised, portable computing and storage resources that support disconnected operations in contested or bandwidth-limited environments. These edge devices enable local execution of mission applications, AI inferencing, sensor data aggregation and encrypted data persistence, all while maintaining a cryptographically verifiable security posture through features such as secure enclave-based computation [65].

Furthermore, the AWS Wavelength service, designed to extend 5G networks with

ultra-low latency cloud services at the network edge, offers promising capabilities for future integration into tactical communication infrastructures. This service facilitates distributed intelligence by enabling mobile platforms to access cloud-native services with minimal delay, supporting mission-critical analytics and decision-making at the point of need.

AWS's approach to cloud, fog and edge computing reflects a comprehensive strategy to support sovereign, resilient and scalable defence architectures. The seamless integration of these layers empowers defence organisations to dynamically shift workloads between central, regional and tactical nodes based on mission requirements, connectivity conditions and security policies, thus aligning with the overarching objectives of connected defence platforms. The concept of confidential computing has emerged as a critical advancement in response to growing concerns regarding data sovereignty, privacy and security in cloud computing environments. Traditional cloud architectures primarily focus on securing data at rest and in transit. However, they offer limited protection for data during processing, where vulnerabilities to insider threats, compromised hypervisors or untrusted administrators may exist.

confidential computing addresses this security gap by integrating hardware-based Trusted Execution Environments (TEEs) and confidential computing technologies into the cloud infrastructure itself. These mechanisms enable the execution of sensitive workloads within isolated, attested and cryptographically protected environments, ensuring that data remains secure even during computation. Through remote attestation protocols, organisations can verify the integrity and trustworthiness of the underlying infrastructure before sensitive information is processed.

In the context of defence, government and critical industries, Confidential Cloud provides a technological foundation for enforcing strict data residency, sovereignty and operational control requirements. By embedding cryptographic trust at the infrastructure level, confidential computing architectures facilitate secure MDOs, coalition-based information sharing and compliance with stringent regulatory frameworks such as the General Data Protection Regulation (GDPR) and emerging sovereign cloud standards.

Major cloud service providers have begun to incorporate confidential computing capabilities into their service offerings, recognising the imperative for verifiable trust in sensitive and mission-critical applications. These advancements not only enhance the security posture of cloud environments but also enable new operational models where sensitive workloads can be securely deployed across public, hybrid and edge cloud infrastructures without compromising confidentiality, integrity or sovereignty.

Cloud providers such as AWS have introduced specialised confidential computing offerings to address the stringent security and sovereignty requirements of defence and public sector organisations. AWS Nitro Enclaves, a key component of this strategy, enable the isolation of sensitive workloads within dedicated, cryptographically attested execution environments, without requiring changes to existing application architectures [65].

Nitro Enclaves operate by partitioning off portions of an EC2 instance's memory and CPU resources to create isolated enclaves that are inaccessible even to the instance owner, hypervisor or operating system. These enclaves leverage hardware-based attestation mechanisms to establish verifiable trust in the enclave's configuration and integrity before sensitive data or cryptographic materials are provisioned. This architecture aligns with Confidential Computing principles, ensuring that data remains protected not only at rest and in transit but also during processing [65].

In parallel, the AWS ESC initiative aims to extend these capabilities by offering physically and logically separated cloud regions within the European Union (EU),

operated by EU personnel under EU jurisdiction. This framework integrates confidential computing technologies such as Nitro Enclaves into sovereign cloud environments, providing verifiable guarantees for data residency, operational control and security compliance. The result is a comprehensive confidential computing model that meets both technical and geopolitical requirements for sensitive workloads.

Moreover, AWS Key Management Service (KMS) integrates with Nitro Enclaves to enable enclave-restricted cryptographic key operations. This ensures that keys used for encryption, decryption and signing can only be accessed by verified and attested enclaves, further enhancing the confidentiality and sovereignty of mission-critical data [65].

The AWS approach to confidential computing thus enables defence organisations to operationalise zero-trust security principles, enforce strict data residency policies and deploy sensitive workloads in cryptographically isolated environments. These capabilities are essential for supporting coalition operations, sovereign mission mandates and multinational interoperability initiatives such as NATO Federated Mission Networking (FMN).

By embedding confidential computing mechanisms into its sovereign cloud architecture, AWS facilitates the creation of secure, policy-enforced digital environments that uphold both technological and strategic imperatives for modern defence infrastructures.

2.5 HIGH-PERFORMANCE COMPUTING FOR DEFENCE APPLICATIONS

High Performance Computing (HPC) refers to the use of aggregated computing power to solve complex computational problems far beyond the capabilities of standard servers or desktop systems. Characterised by massively parallel processing architectures, tightly coupled clusters and high-throughput interconnects, HPC systems enable the efficient handling of vast datasets and computationally intensive simulations across scientific, industrial and military domains [39], [57].

Fundamentally, HPC systems are designed to maximise computational throughput, minimise latency and deliver near-RT processing capabilities. They consist of hundreds to hundreds of thousands of interconnected compute nodes, often equipped with multi-core CPUs, high-bandwidth memory architectures and accelerators such as Graphics Processing Units (GPUs) [39]. Each node contributes to solving a fraction of the problem space, orchestrated via distributed scheduling systems that manage task allocation, resource synchronisation and data communication.

Traditionally, HPC has been associated with on-premises supercomputers housed in dedicated facilities, exemplified by systems such as El Capitan, currently the world's fastest supercomputer [83]. These facilities feature highly specialised infrastructure optimised for cooling, power efficiency and high-speed interconnection, often employing Remote Direct Memory Access (RDMA) fabrics such as InfiniBand to minimise latency and maximise data throughput.

In practice, HPC applications encompass a wide range of domains including weather prediction, nuclear simulations, computational fluid dynamics, genomic sequencing and increasingly, artificial intelligence model training. The distinguishing characteristic of HPC workloads lies in their requirement for extensive parallelism, where computational tasks are decomposed into concurrent operations across numerous processing elements.

A critical distinction must be made between general-purpose computing and HPC: whereas conventional systems can handle parallel workloads to a limited extent via multi-threaded processing, HPC systems are purpose-built for highly parallel execution at massive scale, supporting both tightly coupled and massively parallel workloads [39].

From an architectural perspective, HPC systems typically adopt one of two paradigms:

- **Tightly coupled clusters** where nodes share memory spaces or low-latency interconnects, enabling high degrees of inter-process communication.
- **Loosely coupled grids** where jobs are independently executed across distributed resources with minimal interdependence.

The boundary between high-performance enterprise clusters and true HPC is defined by scale, interconnect efficiency and workload characteristics. HPC begins where the orchestration of thousands of cores and high-speed synchronisation mechanisms becomes essential to meet workload demands — typically when problems cannot be resolved within the memory, compute or I/O limits of a single system or modest cluster [57].

With the advent of cloud computing, HPC is no longer confined to traditional on-premises facilities. Cloud-based HPC (HPCaaS) services democratise access to scalable, high-performance resources, allowing organisations to elastically provision compute clusters without upfront capital expenditure [39]. Nevertheless, challenges related to network latency, workload orchestration and security — especially for sensitive defence-related applications — persist and necessitate careful architectural considerations.

The integration of confidential computing into HPC environments addresses critical trust concerns by ensuring that data remains protected during computation, a concept pivotal for military and coalition-based scenarios. Trusted Execution Environments (TEEs) such as AMD Secure Encrypted Virtualisation (SEV-SNP) and Intel SGX enable the secure processing of classified workloads on both dedicated and public cloud HPC infrastructures [39].

In the defence context, HPC forms an indispensable pillar for advanced simulation capabilities, operational planning, predictive analytics and the RT fusion of multi-domain operational data. Its role becomes even more critical when operating across coalition networks, necessitating architectures that combine raw computational power with sovereignty-preserving and cryptographically verifiable execution environments.

The integration of HPC capabilities into defence infrastructures represents a pivotal development in addressing the escalating computational demands of modern military operations. HPC enables the simulation of complex physical systems, the rapid analysis of vast datasets and the development of predictive models essential for mission-critical decision-making [16]. Historically, the Department of Defense (DoD) has recognised the strategic value of HPC through initiatives such as the High Performance Computing Modernisation Program (HPCMP), which consolidated disparate computational resources across service branches into a unified, scalable and resilient ecosystem [16].

HPC infrastructures in defence settings serve a multitude of operational needs, ranging from computational fluid dynamics for aircraft design to RT battlefield simulations and the optimisation of logistics and supply chain operations. Through parallel processing and distributed architectures, HPC platforms dramatically reduce the time-to-solution for simulations that would otherwise be computationally prohibitive [34]. Moreover, the emergence of High Performance Data Analytics (HPDA) has expanded the traditional HPC role from purely physics-based simulations towards encompassing machine learning, artificial intelligence (AI) and autonomous system training workloads [16].

A defining characteristic of defence-oriented HPC is the necessity for continuous modernisation. Given the rapid obsolescence of hardware and the evolving threat landscape, defence HPC environments undergo systematic technology refresh cycles,

typically investing significant resources annually to maintain competitiveness [16]. Modern acquisition strategies increasingly rely on application-centric benchmarking to ensure that new systems reflect the actual computational workloads of defence research and operational communities, rather than generic benchmarks such as LINPACK or HPCG [16].

Furthermore, confidentiality, integrity and availability requirements in defence HPC infrastructures demand robust security postures. Secure enclaves, Trusted Platform Modules (TPMs) and attested hardware environments are increasingly integrated to ensure that sensitive simulations and mission planning data remain protected throughout the computational lifecycle [61].

The role of HPC extends beyond pure computational acceleration; it acts as an enabler of digital engineering, allowing for the early virtual integration and testing of complex systems, significantly reducing the cost, time and risk associated with physical prototyping. Initiatives such as the Computational Research and Engineering Acquisition Tools and Environments (CREATE) programme exemplify how HPC supports the entire lifecycle of military platforms, from initial concept development to sustainment engineering [16].

Notably, advancements in hybrid architectures, combining traditional HPC nodes with specialised accelerators such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), are transforming the computational landscape. These heterogeneous environments enable the efficient execution of both numerical simulations and AI-driven workflows, supporting evolving operational paradigms that blend physics-based and data-driven approaches [72], [73].

The trend towards decentralised high performance edge computing also highlights the necessity of bridging centralised HPC facilities with edge nodes capable of pre-processing, filtering and selectively forwarding data to core data centres for deeper analysis [49]. This integration is critical for time-sensitive missions where low-latency, high-throughput and operational resilience are paramount.

HPC constitutes a cornerstone of next-generation defence IT architectures, underpinning RT operational capabilities, advanced simulations and sovereign digital ecosystems. The strategic alignment of HPC resources with mission objectives not only enhances operational readiness but also ensures technological superiority in a digitally contested battlespace.

Beyond traditional on-premises deployments, major cloud providers have expanded their service offerings to support HPC workloads through scalable, cloud-native platforms. AWS has emerged as a impactful player in this field by delivering specialised HPC services that align with the demands of scientific, industrial and increasingly military applications.

AWS offers a suite of HPC-optimised services, including EC2 instances with high core counts, enhanced memory bandwidth and low-latency networking features. Compute-optimised instance families, such as C6gn and Hpc6id, are specifically designed for tightly coupled HPC workloads, delivering high performance per core while maintaining cost efficiency. Moreover, AWS Elastic Fabric Adapter (EFA) enables applications requiring high levels of inter-node communication, such as computational fluid dynamics or large-scale simulations, to achieve near-native Message Passing Interface (MPI) performance by providing low-latency, high-bandwidth networking capabilities [27].

To orchestrate complex HPC environments, AWS ParallelCluster offers an open-source cluster management tool that automates the deployment and scaling of HPC clusters on AWS infrastructure. It supports traditional schedulers such as SLURM, Torque and AWS Batch, enabling seamless migration of existing HPC workflows to the

cloud while providing elasticity, automation and cost visibility. For highly sensitive or sovereign workloads, AWS integrates ParallelCluster deployments with Nitro-based EC2 instances, ensuring hardware-enforced isolation and offering the possibility to operate within confidential computing enclaves.

AWS also supports hybrid HPC architectures through services such as AWS Outposts and AWS Snowball Edge, which allow organisations to deploy HPC clusters at the edge or in disconnected environments. These capabilities are particularly relevant for defence operations, where tactical field deployments require both substantial compute power and data sovereignty. In addition, AWS Batch simplifies the orchestration of containerised HPC workloads, offering flexible resource management, job dependency tracking and fault-tolerant execution across multi-instance compute environments.

To accelerate data-intensive HPC workflows, AWS offers high-performance storage solutions such as Amazon FSx for Lustre, a fully managed file system optimised for fast processing of workloads like machine learning model training, video rendering and large-scale simulations. FSx for Lustre can be directly linked to Amazon S3 storage, providing a seamless data pipeline between durable object storage and high-throughput file systems.

Security remains a primary concern in cloud-based HPC deployments. AWS implements confidential computing principles through its Nitro Hypervisor and Nitro Enclaves, ensuring that sensitive HPC workloads can be processed securely within attested, isolated execution environments. Furthermore, AWS KMS and integration with external key management systems enable fine-grained control over cryptographic operations, meeting stringent requirements for classified and mission-critical data protection [65].

Through the combination of scalable HPC infrastructure, secure execution environments and sovereign operational models, AWS enables defence organisations to conduct large-scale simulations, predictive modelling, RT sensor data fusion and mission analytics without compromising security, confidentiality or compliance. The integration of confidential HPC capabilities into sovereign cloud architectures, such as the AWS ESC, represents a decisive enabler for next-generation connected defence operations.

2.6 SENSOR FUSION AND SITUATIONAL AWARENESS

As military operations become increasingly reliant on digitised and interconnected environments, the need for comprehensive situational awareness (SA) has intensified across all operational domains. Situational awareness refers to the capability to perceive, comprehend and project critical elements of the battlespace, enabling timely and informed decision-making under dynamic conditions [48]. In this context, the Internet of Military Things (IoMT) emerges as a transformative paradigm, leveraging a vast ecosystem of interconnected sensors, platforms and devices to support enhanced operational awareness and mission effectiveness [54].

The IoMT ecosystem builds upon the core principles of the broader Internet of Things (IoT) but introduces additional constraints regarding mobility, security, energy efficiency and operational resilience under adversarial conditions. Through dense networks of interconnected edge devices — including unmanned vehicles, surveillance sensors and wearable systems — the IoMT facilitates real-time data collection, fusion and dissemination across distributed command structures [48], [71].

A fundamental enabler of this vision is multi-sensor data fusion, integrating heterogeneous data sources to generate coherent and actionable battlefield intelligence.

Recent advances in machine learning-based fusion techniques, such as convolutional neural networks (CNNs) and reinforcement learning (RL) frameworks, significantly enhance object detection, threat identification and environmental understanding [48], [71]. By combining visual, infrared and radar sensor modalities, fused imagery provides improved depth perception, better robustness under low-visibility conditions and more accurate target classification compared to single-sensor systems.

In parallel, the integration of cloud-edge collaboration architectures enables scalable, resilient processing across IoMT infrastructures. Cloud platforms provide centralised, large-scale analytics capabilities for strategic intelligence, while edge nodes ensure tactical responsiveness through localised inference and decision-making [54]. Techniques such as grey relational analysis (GRA) combined with backpropagation neural networks (BPNN) have been successfully applied to risk situational awareness in power distribution scenarios under cloud-edge architectures, offering promising parallels for military applications [54].

However, the distributed nature of IoMT also introduces substantial challenges. Secure communication, data integrity and trust management are critical, particularly in coalition environments where devices may operate across different jurisdictions. Moreover, network resource constraints and the need for RT responsiveness demand optimised communication and computation strategies, including federated learning models and adaptive task offloading mechanisms [48], [71].

Within autonomous systems, Vehicle-to-Infrastructure Multi-Sensor Fusion (V2I-MSF) frameworks further demonstrate the operational value of cooperative perception architectures. By integrating data from vehicle-mounted and roadside infrastructure sensors, V2I-MSF enhances environmental visualisation, improves motion estimation and strengthens obstacle avoidance capabilities, particularly in complex urban or contested settings [71].

Consequently, the convergence of situational awareness, IoMT, cloud-edge collaboration and advanced data fusion techniques defines a critical capability layer for future military operations. These developments not only enhance decision superiority across tactical, operational and strategic echelons but also align with broader initiatives for secure, sovereign and resilient military digital infrastructures.

In the evolving landscape of connected defence operations, cloud-based architectures play an increasingly critical role in enabling RT SA and advanced sensor fusion. AWS has strategically expanded its service portfolio to address the stringent requirements of military and public sector organisations, offering scalable, secure and resilient capabilities that can be directly leveraged to enhance IoMT infrastructures and operational intelligence.

At the data ingestion layer, AWS IoT Core facilitates the secure, low-latency collection of telemetry data from a broad array of distributed military sensors, including unmanned aerial systems, ground vehicles and environmental monitoring devices. Its native support for secure device authentication, fine-grained access control and end-to-end encryption ensures the integrity and confidentiality of operational data from edge to cloud. Furthermore, AWS IoT Greengrass extends cloud capabilities to edge nodes by enabling local data pre-processing, event-driven analytics and disconnected operations — critical features for deployed assets operating in contested or bandwidth-constrained environments [28].

For the orchestration and integration of heterogeneous sensor data, AWS offers a range of serverless services such as AWS Lambda and AWS Step Functions. These services enable RT, event-driven workflows that perform initial data normalisation, validation and fusion, reducing data redundancy and enhancing the quality of downstream

analytics. Complex sensor fusion pipelines can be constructed using AWS Kinesis for RT data streaming and aggregation, allowing multiple sensor modalities, including electro-optical, infrared and radar inputs, to be synchronised and correlated at scale.

In the domain of machine learning-enabled situational awareness, Amazon SageMaker provides a robust platform for developing, training and deploying deep learning models optimised for sensor data fusion tasks. Models for target detection, classification and threat prioritisation can be rapidly prototyped and deployed, with support for edge inference through SageMaker Edge Manager. This enables near-RT threat assessment and decision support capabilities to be deployed directly onto edge nodes, reducing reliance on cloud uplinks and improving operational agility [29].

To visualise and disseminate the fused situational picture, AWS Amplify and Amazon Location Service offer mechanisms to build secure, RT geospatial applications that render integrated sensor data overlays. These tools support the dynamic updating of the Common Operational Picture (COP), enabling commanders and analysts to maintain a comprehensive, coherent and actionable understanding of the battlespace across multiple domains.

Security and compliance underpin the entire AWS service offering. Sensitive situational awareness data flows can be protected through AWS Nitro Enclaves, which provide isolated, attested execution environments for critical data processing tasks. Additionally, the deployment of such services within AWS GovCloud (US) or the AWS ESC ensures adherence to defence-grade regulatory requirements, including data residency, controlled access and operational sovereignty [65].

The strategic integration of AWS cloud, edge and machine learning services offers a robust foundation for achieving advanced situational awareness and resilient sensor fusion in modern military operations. These capabilities align with the overarching principles of confidentiality, integrity and availability, while supporting coalition interoperability, operational flexibility and sovereign control over mission-critical data infrastructures [28].

2.7 C4ISR: COMMAND, CONTROL, COMMUNICATIONS, COMPUTERS, INTELLIGENCE, SURVEILLANCE AND RECONNAISSANCE

The concept of C4ISR — *Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance* — represents a comprehensive operational and technological framework essential for orchestrating military operations in increasingly complex and contested environments. C4ISR systems aim to integrate multi-domain data acquisition, secure communications, computational processing and decision-support mechanisms into a unified, dynamic operational picture that facilitates situational awareness, mission command and synchronised execution [25].

At its foundation, C4ISR seeks to enable the rapid and reliable dissemination of accurate, actionable information across tactical, operational and strategic levels of command. This requires architectures capable of assimilating large volumes of heterogeneous data in real time, synthesising critical intelligence and securely distributing insights even under conditions of degraded communication or electromagnetic spectrum denial [17].

Modern C4ISR systems increasingly rely on decentralised, service-oriented architectures (SOAs), leveraging cloud, fog and edge computing paradigms to ensure resilience, scalability and coalition interoperability. Research demonstrates that probabilistic frameworks, such as Bayesian Networks (BN) and Dynamic Bayesian Networks (DBN), enhance the resilience assessment of C4ISR infrastructures, enabling proactive

risk mitigation and adaptive resource reallocation in contested operational environments [12].

Service deployment in C4ISR must accommodate limited bandwidth, dynamic topologies and adversarial threats. Innovative approaches, such as service orchestration through Quantum Evolutionary Algorithms (QEAs) and Multi-Objective Genetic Simulated Annealing Algorithms (MOGSAAs), optimise the placement and replication of critical services, improving system robustness and communication efficiency under constrained conditions [13], [18].

Given these evolving operational demands, cloud service providers such as AWS offer a portfolio of technologies that align with the architectural principles of next-generation C4ISR systems. At the data ingestion layer, AWS IoT Core enables the secure connection and management of thousands of fielded sensors, platforms and autonomous systems. It ensures data integrity through device attestation, end-to-end encryption and fine-grained identity and access management policies [28].

At the tactical edge, AWS IoT Greengrass extends cloud-native capabilities to local environments, supporting sensor fusion, event-driven processing and autonomous decision-making even in intermittently connected or denied areas. This capability directly addresses the need for resilience and operational continuity in dynamic theatres of operation.

RT data aggregation and initial fusion are supported by AWS Kinesis, which provides scalable and low-latency streaming infrastructure. Tactical data from unmanned systems, ground forces and maritime assets can be dynamically aggregated, filtered and correlated, enabling the construction of a near-RT COP.

For higher-level analytics, AWS SageMaker offers a fully managed environment to develop, train and deploy machine learning models for automated target recognition, threat classification and predictive mission analytics. Integrated with AWS Ground Truth, annotated multi-modal datasets from electro-optical, infrared and radar sensors can be efficiently generated and used to train sensor fusion models [29].

Operational sovereignty and mission assurance are underpinned by AWS Nitro Enclaves, providing isolated, attested execution environments for sensitive C4ISR workloads. Additionally, the use of AWS GovCloud and the emerging AWS ESC ensures that mission data remains compliant with strict national sovereignty, security and data residency requirements [65].

The integration of decentralised, sovereign and resilient cloud services into C4ISR architectures enables a new level of operational agility, robustness and interoperability. By leveraging distributed sensor integration, real-time fusion, secure processing and sovereign cloud governance, connected defence platforms can maintain decision superiority and mission effectiveness even in highly contested multi-domain environments.

2.8 RELATED WORK

The evolution of large-scale data platforms has been significantly influenced by developments in big data processing, open data management and distributed system design. Several initiatives have addressed the challenges associated with data storage, management, governance and analytical pipelines, offering valuable insights and architectural paradigms. Nevertheless, when compared to the next-generation objectives pursued by the *OmniAware* project — namely, the design of a sovereign, secure and interoperable military data platform — certain gaps and limitations in existing platforms become evident.

Smart Open Data As a Service (SODAS) [21] proposes an advanced open data platform that enhances legacy systems such as Comprehensive Knowledge Archive Network (CKAN) by introducing an extensible data model based on Data Catalog Vocabulary Version 2 (DCATv2), RT data harvesting and flexible metadata management. While SODAS addresses interoperability and data quality issues in civilian open data ecosystems, it does not inherently incorporate sovereign execution, secure enclave processing or resilience against adversarial conditions, which are central to *OmniAware*'s military-grade requirements.

The Sparkling Water platform [9] demonstrates effective integration of data mining libraries in distributed data processing environments. By developing a complete software layer for executing various data mining steps, it highlights the performance benefits achievable through distributed architectures. However, its focus remains on horizontal scalability and machine learning performance rather than on secure data sovereignty, trust federation or mission assurance — key pillars of *OmniAware*'s architectural blueprint.

Efforts to unify heterogeneous storage systems have been explored by Nguyen and Won [4] through the introduction of a Data Storage Adapter for big data platforms. Their system simplifies access to diverse data storage engines and supports multiple data processing frameworks. While this approach improves technical interoperability, it lacks support for cryptographic trust anchors, attested data flows or sovereign control mechanisms, all of which are integral to *OmniAware*'s secure multi-domain design.

In the domain of campus data governance, Zhao [62] proposes a big-data-driven governance platform to address issues of data quality, standardisation and lifecycle management within smart campus environments. The focus on multi-layered architectures comprising data governance, platform and service layers resonates conceptually with *OmniAware*'s layered control plane. However, the scope of campus platforms is limited to internal organisational needs and does not extend to coalition-based, contested or sovereign operational scenarios envisioned for connected defence environments.

Recent research by Nagarkar et al. showcases the development of a multi-cloud data pipeline for recommendation systems, integrating services across Google Cloud and MongoDB Atlas. Their work underlines the feasibility of synchronising data and machine learning workflows across heterogeneous cloud environments, which aligns in part with *OmniAware*'s ambition to orchestrate mission-critical workloads across sovereign cloud regions and tactical edge nodes. Nevertheless, their approach remains oriented towards civilian content platforms without the embedded security, compliance and sovereignty guarantees required for defence operations [55].

In summary, while significant advances have been made in data management, distributed processing and interoperability across various domains, none of the surveyed platforms fully address the stringent requirements for sovereignty, security, multi-domain operational resilience and coalition interoperability. *OmniAware* distinguishes itself by combining sovereign-controlled data orchestration, attested confidential processing and dynamic multi-domain federation into an integrated architecture tailored specifically for the demands of future coalition defence operations.

2.9 METHODOLOGICAL APPROACH AND STRUCTURAL OVERVIEW

To address the stated research questions, this thesis adopts a *design science research* methodology [1], combining conceptual modelling, architectural design and prototype-based validation. The central research artefact — a secure, sovereign and interoperable data platform for defence use cases — is developed iteratively across all layers of the NAFv4, with a strong focus on confidentiality, compliance and deployment realism.

The structure of the thesis reflects this layered approach:

- **Chapter 1** introduces the research context, objectives and guiding questions.
- **Chapter 2** lays the conceptual and methodological groundwork, covering relevant standards, technologies and architectural principles.
- **Chapter 3** presents the system architecture and design of the *OmniAware* Defence Platform based on NAFv4 artefacts.
- **Chapter 4** transitions to the implementation layer, including secure deployment and integration aspects.
- **Chapter 5** evaluates selected implementation facets under realistic constraints.
- **Chapter 6** concludes the thesis and outlines avenues for further research and operationalisation.

By structuring the work along the modelling-implementation-validation triad, this thesis demonstrates the feasibility of a NATO-aligned, confidential computing-enabled defence data platform, grounded in both academic rigour and operational applicability.

3 ARCHITECTURE AND DESIGN

Summary: Following the conceptual, regulatory and technological foundations established in Chapter 2, this chapter transitions from architectural theory to system design. It introduces the formal architecture of the *OmniAware* Core Platform and elaborates on its realisation through NATO-compliant artefacts, attested infrastructure components and security-validated deployment topologies.

The architecture is developed in alignment with the methodological pillars of the NAFv4 and is implemented using the modelling methodology (cf. Section 2.2). The design follows a viewpoint-based decomposition logic that ensures traceability from strategic capability definitions to deployable service blueprints. The focus lies on the practical instantiation of the sovereign, federated and secure platform defined in Chapter 1, emphasising composability, attestation integrity and operational separation of concerns.

To demonstrate applicability, the architecture is structured around two distinct operational scenarios derived from mission requirements: *Platform Health Monitoring (PHM)* and the *Contextual Image Verification System (CIVS)*. Each use case guides the capability mapping and view generation across NAFv4-compliant artefact layers. These include, but are not limited to, the NSV-1 (System Deployment), NPV-1/2 (Architecture Roadmap and Lines of Development) and NAV-1 (Standards and Reference Architecture) viewpoints.

The PHM scenario serves as the primary design driver and is presented in full detail across architecture and implementation layers. The CIVS scenario acts as a derivative reference to demonstrate architectural extensibility and cross-domain applicability. This modelling approach allows for the representation of both shared platform capabilities and use-case-specific components while maintaining alignment with defence-grade assurance frameworks.

The remainder of this chapter is organised as follows. Section 3.1 presents the high-level capability and service model; Section 3.2 outlines the deployment architecture through NAFv4 artefacts, including blackbox and system topology views; Section 3.3 addresses the confidential computing security layer and policy-enforced trust chain; Section 3.4 defines critical interface designs. Where applicable, reference is made to actual implementation artefacts, such as CloudFormation templates, CI/CD pipelines and enclave attestation flows.

By linking NATO-aligned architecture modelling with verifiable security primitives, this chapter lays the foundation for the system implementation outlined in Chapter 4 and the validation strategy discussed in Chapter 5.

3.1 OVERVIEW AND VIEWPOINTS

Summary: This section establishes the architectural groundwork of the *OmniAware* platform by structuring its logical model through a deliberately selected subset of NAFv4 views. The chosen viewpoints—NCV-2, NCV-3, NSOV-3, NSOV-6 and NSV-1—enable conceptual traceability between strategic capability evolution and realisable technical deployment.

In particular, emphasis was placed on modelling service functions with fine granularity to ensure traceable mapping to mission-driven capabilities. This modelling depth supports composability, policy enforcement and infrastructure separation. Orchestration logic (NSOV-6) and capability-to-function allocation (NSOV-3) were explicitly integrated to uphold operational autonomy and deployment modularity.

The resulting architecture blueprint serves not only as a structural anchor for subsequent cloud-native system design but also as an abstraction layer for reasoning about security- and compliance-aligned deployment across sovereign execution domains.

3.1.1 *Architectural Methodology and Modelling Approach*

The architectural development presented in this chapter follows a structured, model-driven methodology grounded in the NAFv4. The goal is to translate operational mission requirements into technically sound, secure and interoperable cloud-native system architectures. The modelling approach strictly adheres to a viewpoint-based decomposition aligned with NAFv4, created in accordance with the modelling stack outlined in Section 2.2 to construct semantically valid, traceable and reusable artefacts [53], [74].

The design process begins with the definition of concrete operational use cases — in this thesis, PHM and CIVS — which are then mapped to capability structures in the NCV-2 view. These capabilities are refined through logical activities (NLV), service function mappings (NSV) and physical deployment artefacts (NPV), ensuring traceability across all architectural layers. Special attention is given to integrating security and compliance requirements into the modelling process from the outset, particularly with regard to confidential computing, remote attestation and sovereign deployment strategies.

By adopting a formal and iterative architecture development methodology, this thesis ensures that each modelled component is both contextually meaningful and technically sound. The resulting architecture is designed not only to demonstrate the feasibility of secure defence cloud deployments, but also to serve as a reusable blueprint for future coalition-based mission platforms.

3.1.2 *Contribution and Project-Specific Realisation*

This section presents a major contribution of this thesis, reflecting the author's original implementation work, architectural synthesis and methodological decision-making within the development of the *OmniAware* Connected Defence Platform. Building upon the previously introduced conceptual foundations, this part transitions from theory to practice, tracing how NAFv4-compliant architecture was instantiated and operationalised through practical artefacts and modelled system views.

The contribution is structured around two main pillars:

Methodological Reflection. While certain modelling elements were derived from established NATO and AWS best practices, this thesis involved deliberate architectural decisions tailored to the PHM use case. For instance, the inclusion of NSV-4a and NSOV-6 was prioritised over more abstract orchestration chains (NSV-5) to increase implementation realism. Similarly, NAV-1 was introduced manually, despite being optional in the official viewpoint catalogue, to strengthen compliance traceability across lifecycle stages.

1. **Architectural Realisation:** This includes the implementation of the NAFv4-compliant models (e.g. NSV-4a, NSV-6, NPV-3), the development of a PHM-focused reference architecture and the detailed representation of capability-to-technology mappings. Each architecture artefact was manually modelled and validated using the modelling methodology (cf. Section 2.2), ensuring traceability and semantic rigour.

2. **Interface and Infrastructure Prototyping:** As a practical realisation of the PHM use case, this thesis provides a containerised microservice blueprint for attested workloads, including enclave-enabled components. Core infrastructure templates, secrets handling mechanisms and `Vault`/Key Management System (KMS) integrations are highlighted as part of the implemented MVP.

By combining methodical modelling discipline with secure, infrastructure-as-code deployments, this work demonstrates how defence-relevant confidentiality, compliance and interoperability requirements can be translated into a working system foundation. The presented implementation artefacts — architectural views, cloud infrastructure modules and interface blueprints — constitute the tangible result of this research and provide a reusable and extendable blueprint for future defence cloud developments.

3.1.3 *Viewpoint Selection and Model Justification*

The *OmniAware* platform was modelled in accordance with the NAFv4 viewpoint taxonomy realised using the tooling and validation principles defined earlier (cf. Section 2.2). While the NAFv4 defines over 50 architectural viewpoints across its full specification, this thesis adopts a deliberately reduced and semantically focused subset. This decision reflects both the constraints of a time-bounded PoC and the intention to maintain traceability and model clarity without excessive redundancy. NSV-5 was omitted due to redundancy with orchestration logic already embedded in NSOV-6.

NAFv4 itself encourages tailoring of the viewpoint catalogue based on relevance and the following three selection principles were applied:

1. **Relevance to mission modelling:** Views were prioritised that directly support capability decomposition, operational scenario realisation and service deployment.

2. **Traceability across architectural layers:** The chosen views enable semantic continuity from capability definition to physical deployment (e.g. NCV-2 → NSOV-3 → NSV-1 → NPV-2).

3. **Modelling economy:** Redundancies across views were consciously avoided by using semantically expressive artefacts (e.g. NSOV-6 for both service grouping and orchestration).

Architectural Decision Rationale. The selection and structuring of views were not based solely on tool availability or template conformance, but reflect applied modelling decisions. Instead of pursuing comprehensive viewpoint coverage, the thesis concentrated on views with high architectural expressiveness and implementation relevance. These choices were made based on a structured analysis of stakeholder priorities, expected functional coverage and practical constraints encountered during the modelling phase. The result is a NATO-compliant architectural baseline that supports model reuse, auditability and deployment-level realisability.

Viewpoint Selection Rationale. The final subset includes: NCV-2 (Capability Dependencies), NCV-3 (Capability Roadmap), NSOV-3 (Service Functions), NSOV-6 (Service Orchestration Logic), NSV-1 (System Deployment), NPV-2 (Resource Mapping) and NAV-1 (Compliance Traceability). Several other candidate views — such as NSV-5 (Orchestration Chains), NLV-4 (Information Exchanges) and NAV-2 (Standards Coverage) — were considered but intentionally excluded. NSV-5 was omitted due to redundancy with orchestration logic already embedded in NSOV-6. NAV-2 and NLV-4 were deprioritised due to limited marginal benefit for a first iteration model and insufficient tooling support within Archi. These exclusions reflect a pragmatic modelling economy and deliberate simplification to ensure semantic clarity and model completeness within the constraints of the PoC scope.

Having justified the methodological selection and structuring of architectural viewpoints, the following section introduces their concrete application to the operational design of the *OmniAware* platform. In line with the capability-driven modelling paradigm of NAFv4, this begins with the NCV-2 view — the conceptual anchor that maps mission-specific capability clusters and their dependencies.

These capability definitions form the starting point for traceable refinement across all subsequent viewpoints, enabling semantic alignment between strategic intent, service orchestration and technical deployment. The capability view not only reflects stakeholder priorities and use-case logic, but also structures the architectural backbone on which service, orchestration and trust enforcement models are built.

3.1.4 *Strategy*

The following view illustrates the capability dependencies underlying the core architecture of the *OmniAware* platform. It highlights the incremental development and integration of critical defence capabilities, ranging from foundational components such as *Secure Federated Data Exchange* to advanced mission enablers such as *Confidential AI-based Mission Planning*.

As depicted in Figure 3.1, the realisation of sovereign and mission-resilient capabilities is built upon foundational layers such as secure infrastructure orchestration and sovereign identity management. These core capabilities serve as enablers for higher-level services such as confidential analytics and federated mission command and control (C2). Dependencies between capabilities are represented using *serving relationships*, indicating sequential or conditional development requirements.

- **General:** Cross-cutting platform-level enablers such as `C1_Cloud Computing Platform`, `C2_Sensor Data Ingestion` and `C4_Confidential Computing/Data Sovereignty`, providing the core infrastructure and security primitives.

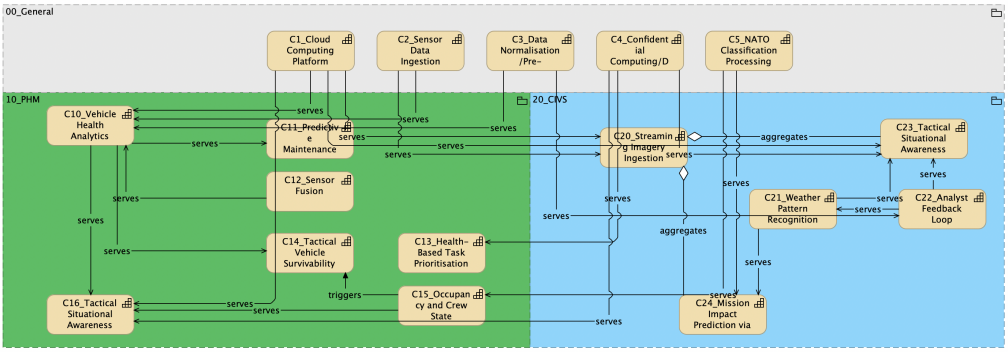


Figure 3.1: NCV-2: *OmniAware* Capability Dependencies

- **PHM:** Prognostic and Health Management functions, including telemetry-based analytics, component-level health assessment and survivability estimation (e.g. `C10-C16`).
- **CIVS:** Visual Intelligence and Mission Simulation functions for real-time awareness, pattern recognition and environmental impact analysis (e.g. `C20-C24`).

Each connection indicates a logical and traceable dependency that must be fulfilled to enable downstream capabilities. For instance, `C10_Vehicle Health Analytics` relies on foundational services for ingestion, normalisation and cloud orchestration. Likewise, `C24_Mission Impact Prediction` requires sovereign data handling (`C5`) and pre-processed inputs from visual and weather sources.

Capability Dependency Justifications. A detailed justification of all modelled capability relationships is included in the appendix (cf. Appendix 1).

Design Considerations and Capability Prioritisation. The decomposition of the implementation roadmap (cf. Figure 3.2) follows a capability-centric prioritisation that reflects mission relevance, architectural dependency and system-level security impact across the *OmniAware* platform. This prioritisation does not emerge from agile feature breakdowns, but is instead anchored in a structured analysis of operational workflows and stakeholder needs.

As outlined in Section 3.1.4, the capability structures and interdependencies were derived from detailed BPMN models and strategic product documentation in the form of PR/FAQ artefacts [75], [77], [79]. This modelling approach ensures that the platform's capability roadmap is not the result of arbitrary breakdowns, but is grounded in the mission-centric logic of the PHM and CIVS use cases.

Capabilities deployed in the early phase (Q1) include foundational enablers such as secure cloud computing, data ingestion and confidential computing. These are essential for enabling trusted enclave execution, identity-based access control and policy enforcement, forming the technological backbone for secure data processing. The subsequent phase (Q2) builds on this base and integrates telemetry-specific analytics such as vehicle health monitoring, sensor fusion and early classification. These processing functions are tightly coupled with upstream data ingestion and serve as prerequisites for domain-specific logic.

Phases Q3 and Q4 progressively integrate more complex analytical and decision-support capabilities. Q3 capabilities focus on applied health analytics and situational

prediction — such as predictive maintenance and crew state estimation — which require reliable, pre-processed data from earlier stages. Q4 introduces high-level situational awareness features including AI-based image analysis, weather pattern recognition and mission impact prediction. These capabilities synthesise and contextualise upstream outputs to support operational planning and cross-domain decision-making.

By prioritising capability clusters based on functional dependencies, strategic mission alignment and feasibility considerations, the roadmap follows a pragmatic implementation sequence. This sequence mirrors the dependency logic visualised in the NCV-2 view (cf. Figure 3.1) and ensures traceability and coherence in line with NAFv4 architectural guidance.

This capability-centric decomposition ensures a secure, modular and scalable implementation path in alignment with NAFv4 standards and the strategic objectives of *OmniAware*.

Building upon the capability dependencies outlined in **NCV-2**, the following roadmap illustrates the temporal evolution and sequencing of key capabilities across the *OmniAware* platform. It highlights the staged development from foundational enablers towards mission-critical analytics, simulation and decision-support functionalities.

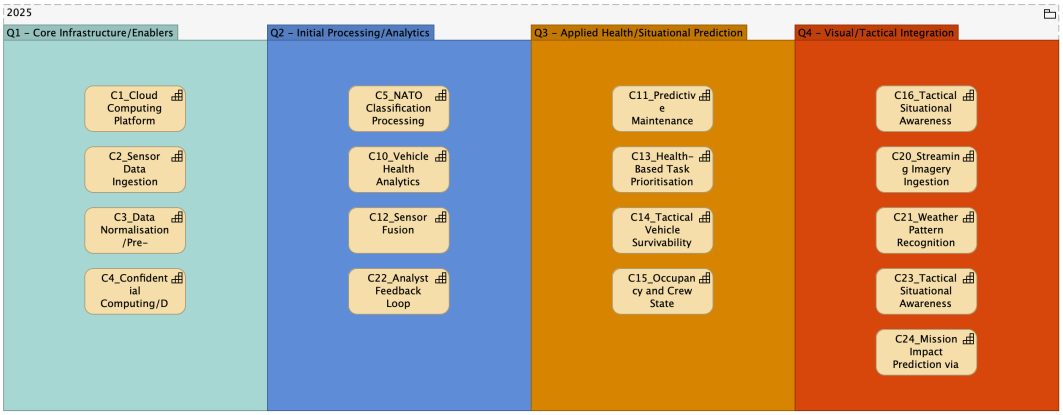


Figure 3.2: NCV-3: *OmniAware* Capability Roadmap

As illustrated in Figure 3.2, the roadmap decomposes the implementation timeline into four sequential capability phases, aligned with quarterly milestones:

- **Q1 - Core Infrastructure/Enablers:** Deployment of foundational services, including `C1_Cloud Computing Platform`, `C2_Sensor Data Ingestion`, `C3_Data Normalisation/Pre-Processing` and `C4_Confidential Computing/Data Sovereignty`, which provide the essential backbone for secure execution and data integrity.

- **Q2 - Initial Processing/Analytics:** Implementation of intermediate functions for data transformation and classification, notably `C5_NATO Classification Processing`, `C10_Vehicle Health Analytics`, `C12_Sensor Fusion` and feedback handling via `C22_Analyst Feedback Loop`.

- **Q3 - Applied Health/Situational Prediction:** Integration of advanced mission health features such as `C11_Predictive Maintenance`, `C13_Health-Based Task Prioritisation`, `C14_Tactical Vehicle Survivability` and `C15_Occupancy and Crew State`, enabling data-driven mission readiness insights.

- **Q4 - Visual/Tactical Integration:** Final integration of high-level situational awareness and simulation layers, including `C16_Tactical Situational Awareness`, `C20_Streaming Imagery Ingestion`, `C21_Weather Pattern Recognition`, `C23_Tactical Situational Awareness` and `C24_Mission Impact Prediction via Simulation`.

The roadmap reflects implementation feasibility and architectural dependency constraints. Each phase builds incrementally upon the prior, reducing integration risk and ensuring capability coherence across domains.

The phased implementation approach is consistent with NAFv4 methodology. It supports traceability between concept-level capability development and technical instantiation in Logical and Service Specification Viewpoints, thereby facilitating stakeholder alignment and milestone validation.

3.1.5 *Application*

Building on the previously defined capability roadmap (cf. **NCV-3**), the following diagram illustrates the phased evolution of foundational service functions in alignment with capability clusters and implementation phases of the *OmniAware* platform.

Design Translation and Architectural Consolidation. The application view builds on the previously defined *capability roadmap* (cf. `NCV-3`) and systematically transfers each capability into a corresponding set of service functions and orchestration patterns. These services are explicitly modelled in `NSOV-3` and `NSOV-6`, reflecting not only the implementation maturity but also the modularisation logic of the *OmniAware* platform.

Rather than emerging from conventional top-down decomposition, all mapped service functions originate from structured artefacts—specifically from BPMN process models and PR/FAQ documentation aligned with the PHM and CIVS mission domains [75], [77], [79]. Each function thus represents a concrete instantiation of previously defined capabilities, ensuring operational fidelity and traceability to stakeholder needs and tactical objectives.

This transition from abstract capabilities to executable service logic follows a layered modelling approach: foundational general services (e.g. `SF1-SF8`) form the platform core for security and data integrity; PHM services (`SF10-SF19`) reflect health-state analytics and mission readiness logic; CIVS services (`SF20-SF27`) support sensor fusion, visual pattern recognition and simulation-based planning. Their temporal allocation across capability stages Q1-Q4 (cf. `NCV-3`) is preserved and reflected in their grouping and orchestration semantics in `NSOV-6`.

This design step consolidates architectural decisions by mapping high-level capability models to deployable and modular service layers, compliant with NAFv4 viewpoints and ready for downstream orchestration and policy control implementation.

Service Function Decomposition and Service Structuring. The functional design of the *OmniAware* platform is grounded in a structured decomposition of *services* and their respective *service functions* that realise the capability demands defined in Section 3.1.4. Rather than aiming for exhaustive microservice design, this decomposition aims to provide a logically coherent breakdown of service functions grouped by capability cluster and mission-specific requirements.

The modelling approach follows NAFv4-compliant views, notably NSOV-6, where service functions are linked to applications via realisation relationships. Although these relationships are formalised using realises connectors, they serve primarily as analytical artefacts to express design intent rather than enforce concrete implementation blueprints.

This abstraction allows early-stage modelling without locking down the full deployment stack. In particular, service functions were derived by analysing the interaction and information flows embedded in the BPMN models and cross-validated against capability requirements from NCV-2. For example, functions such as Sovereign Policy Enforcement or Secure Storage and Access Layer support the implementation of C4_Confidential Computing/Data Sovereignty, while others such as Confidential Data Ingestion or NATO Classification Processing contribute to C5 and C10 in the PHM context.

Function groupings were additionally guided by early cloud architecture constraints: the PoC deliberately excluded orchestration platforms like Kubernetes in favour of simplified stack deployments to reduce system complexity in Q1/Q2. This constraint had a direct impact on the grouping of services and design of functional boundaries. Consequently, the number of deployed functions was limited and collocated in EC2-based trust zones (e.g. SEV-SNP or Nitro Enclaves), mapped closely to infrastructure-as-code templates and security policies.

Through this structuring, the design retains the ability to reason across abstraction levels ——from BPMN interaction flows to capability fulfilment and service-level realisation — without prematurely locking into operational dependencies or deployment frameworks.

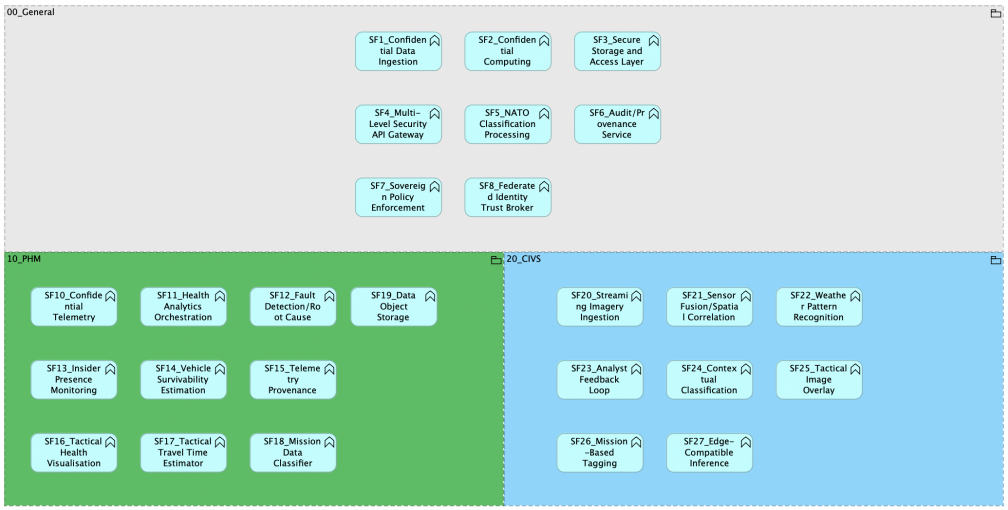


Figure 3.3: NSOV-3: *OmniAware* Service Functions

As shown in Figure 3.3, the NSOV-3 models all *Service Functions* of the ***OmniAware* Core System**, grouped according to their functional domain — General, PHM and CIVS. Each service function represents a discrete, independently deployable microservice or execution component that delivers mission-critical functionality aligned with NAFv4 service viewpoints.

- **General Functions (SF1-SF8):** These include foundational services such as SF1_Confidential Data Ingestion, SF2_Confidential Computing, SF3_Secure Storage and Access Layer, SF4_Multi-Level Security API Gateway, SF5_NATO

`Classification Processing`, `SF6_Audit/Governance Service`, `SF7_Sovereign Policy Enforcement` and `SF8_Federated Identity Trust Broker`. They form the baseline for secure data handling, compliance, identity federation and sovereign execution.

- **PHM Functions (SF10-SF19):** These functions realise the platform's predictive health monitoring and analytics capabilities. They include `SF10_Confidential Telemetry`, `SF11_Health Analytics Orchestration`, `SF12_Fault Detection/ Root Cause`, `SF13_Insider Presence Monitoring`, `SF14_Vehicle Survivability Estimation`, `SF15_Telemetry Provenance`, `SF16_Tactical Health Visualisation`, `SF17_Tactical Travel Time Estimator`, `SF18_Mission Data Classifier` and `SF19_Data Object Storage Governance`.
- **CIVS Functions (SF20-SF27):** These functions enable situational awareness and mission analytics. The services include `SF20_Streaming Imagery Ingestion`, `SF21_Sensor Fusion/Spatial Correlation`, `SF22_Weather Pattern Recognition`, `SF23_Analyst Feedback Loop`, `SF24_Contextual Classification`, `SF25_Tactical Image Overlay`, `SF26_Mission-Based Tagging` and `SF27_Edge-Compatible Inference`.

Each service function is directly realisable through containerised workloads and is mapped to one or more capabilities as described in the NSOV-4 and NCV-3 Views. Service Functions are intentionally modelled at a granular level to allow fine-grained lifecycle management, scalable deployment and functional reuse across application domains.

For implementation-level design, this implies that an additional application layer would typically encapsulate service logic and interface definitions. In this model, however, the service functions are considered abstracted enough to represent these semantics directly. This simplification does not compromise the analytical value of the view but should be kept in mind when interpreting architectural compliance or extending the model in future work.

Building on the functional decomposition outlined in NSOV-3, the NSOV-6 view focuses on the structural composition and internal dependencies of services within the *OmniAware* platform. It visualises orchestrated relationships, service reuse patterns and logical groupings relevant for deployment, modularity and policy enforcement across mission domains.

As shown in Figure 3.4, the services are logically grouped and composed into three orchestration clusters:

- **Secure Data Entry and Trust Enforcement (S1-S3, S5-S7)**: This grouping governs the secure ingestion, metadata management and enforcement of security policies such as encryption, RBAC and attestation. These services are reused across mission workflows and form the backbone of data lifecycle protection.
- **Orchestrated Backend Services (S4, S8-S9)**: This cluster enables auditability, federated identity mediation and compliance tagging. Services here are typically `used by` or `depended on` by policy enforcement and mission analytics logic.
- **Frontend-Orchestrated Workflows (S10-S13, S20-S23)**: These services realise higher-level application flows, such as notification, dashboard rendering or fusion pipelines. They `compose` their logic from several backend capabilities and often initiate secure data interactions.

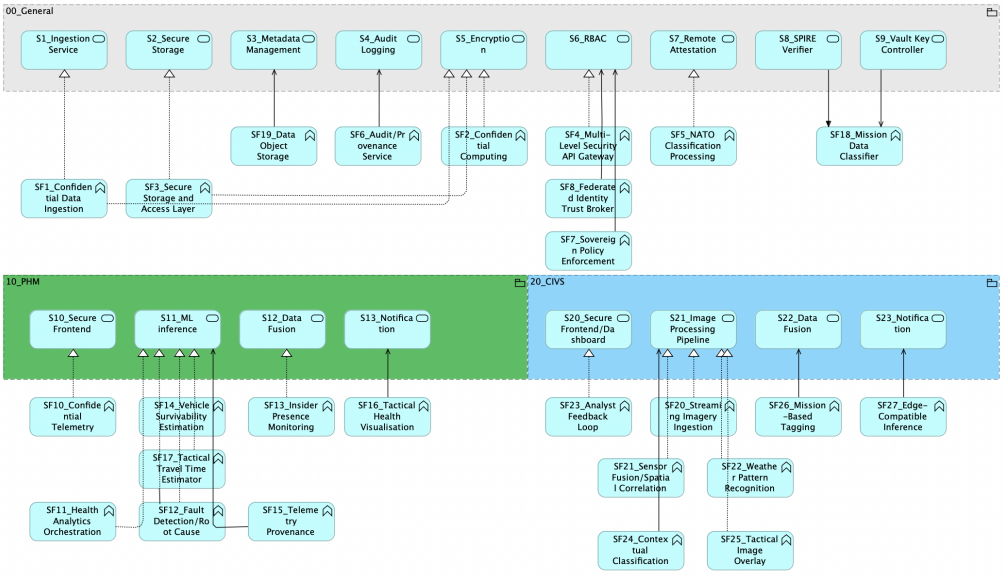


Figure 3.4: NSOV-6: *OmniAware* Service Structure

This view enables architects to reason about orchestration paths, service chaining and security integration in a modular and reusable way. It forms the basis for fault-tolerant, policy-bound and scalable mission deployments.

Although the modelling process broadly follows the NAFv4 viewpoint methodology, the creation of `NLV-4` (Information Exchange Requirements) models deliberately were omitted. This decision is motivated by the focus on security-critical deployment and interface views, which were prioritised over operational information flow descriptions due to project constraints and the absence of stable consumer-side communication semantics at this stage.

3.1.6 *Technology*

As illustrated in Figure 3.5, the NSV-1 maps *OmniAware*'s modular services to their concrete execution environments, grouped by trust domains and hardware-backed runtime protections. The deployment model differentiates between general-purpose backend services hosted in secure cloud/fog infrastructures and mission-specific services operating at the tactical edge.

General Services — such as `S1_Ingestion Service`, `S2_Secure Storage`, `S3_Metadata Management`, `S4_Audit Logging` and `S6_RBAC` — are hosted exclusively within confidential computing infrastructure using AMD SEV-SNP-backed virtual machines. These VMs each instantiate a *Kata Confidential Runtime*, enforcing memory encryption and runtime isolation via attested container boundaries. Deployed Kubernetes pods encapsulate each service and are chained through a policy-controlled key release mechanism operated by the `Policy Proxy` (`30-2_Policy Proxy`).

PHM and CIVS Services — namely `S10-S13` (PHM) and `S20-S23` (CIVS) — are realised at the tactical edge on embedded devices (`Jetson AGX Orin`) equipped with OP-TEE. Due to the architectural limitations of TrustZone, this only provides lightweight enclave protection without full runtime confidentiality or remote attestation. The services are grouped into mission-specific execution cores (*Trusted Execution Core (OP-TEE)*) and

executed in isolated Edge Pods. Examples include health telemetry analytics, secure UI rendering, image pipelines and notification services. Each node is tightly coupled to the mission domain it supports, either PHM or CIVS and deployed on mobile edge nodes (e.g. vehicle-based compute).

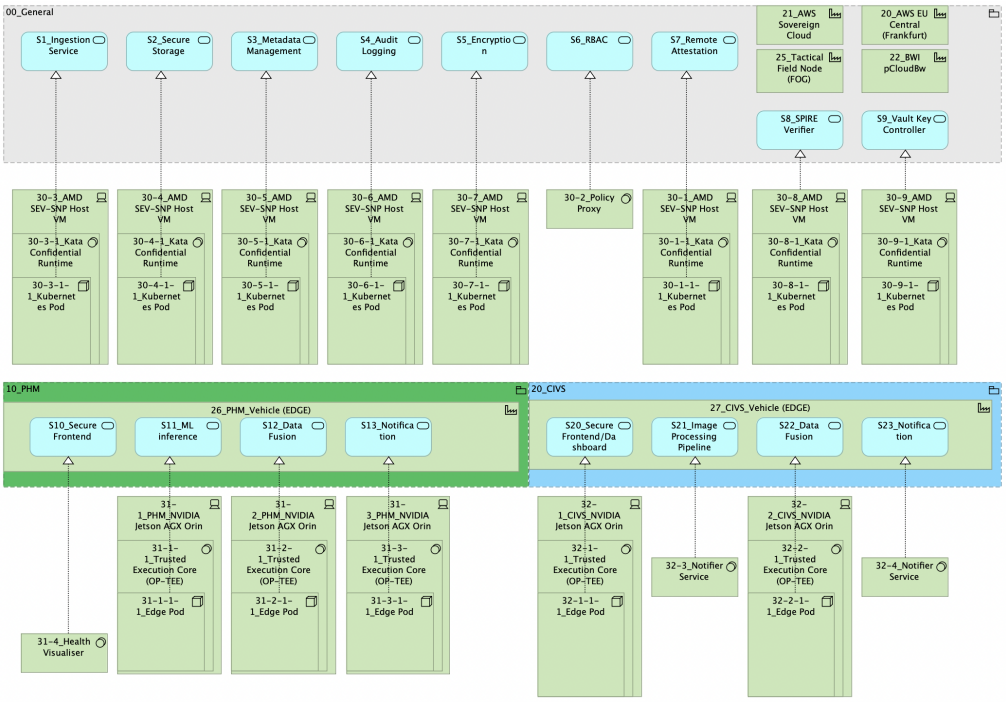


Figure 3.5: NSV-1: *OmniAware* System Deployment

The deployment model presented in `NSV-1` avoids consolidating multiple services into shared compute units. Instead, each service is mapped to an isolated execution environment (e.g. dedicated AWS Lambda function or EC2 instance), ensuring strict separation of concerns, reduced blast radius and alignment with zero-trust security principles.

This architecture reflects the principle of mission-informed deployment: while general-purpose backend workloads are centralised in sovereign cloud environments to benefit from cryptographic attestation and trusted orchestration, latency-critical domain logic is pushed towards the edge, sacrificing some assurance levels for responsiveness. The deployment ensures:

- **Confidentiality:** All sensitive control plane services run in memory-encrypted SEV-SNP VMs with attested Kata runtimes.
- **Modularity:** Each functional unit is encapsulated as a service in an isolated pod, aligned with its hardware trust level.
- **Trust Differentiation:** Mission-critical logic executes in OP-TEE-secured Jetson nodes, modelled as Trusted Execution Cores under degraded but locally trusted conditions.
- **Purpose:** Demonstrates deployment alignment of *OmniAware* services across multi-tier execution zones with differentiated trust.

- **Scope:** Shows the runtime mapping of service components to specific enclave types (SEV-SNP, OP-TEE) and container runtimes (Kata/Edge).
- **Value:** Enables traceable reasoning about cloud-to-edge service placement, runtime assurance and operational resilience under sovereign and coalition constraints.

Edge Runtime Architectures. Several alternative edge execution models—including Intel SGX enclaves, `gVisor` or `Firecracker` microVMs—were considered during the architectural review. However, these approaches were excluded due to limited attestation flexibility, lack of interoperability across trust domains or the absence of integrated remote attestation workflows. In contrast, the use of OP-TEE-enabled `Jetson AGX Orin` devices offered lightweight, deterministic runtime environments with support for mission-specific embedded inference pipelines under degraded trust assumptions.

Kata Confidential Runtime. The deployment of backend control-plane services on SEV-SNP-enabled virtual machines leverages the *Kata Confidential Runtime* to combine hardware-backed isolation with cloud-native container orchestration. Unlike static VMs or lightweight sandboxing solutions (e.g. `gVisor`, Firecracker), Kata provides a confidential micro-VM abstraction layer fully integrated with the Kubernetes Container Runtime Interface (CRI). This allows each service pod to run in an individually attested enclave environment while maintaining compatibility with standard Open Container Initiative (OCI)-compliant tooling.

The selection of Kata was informed by both architectural pragmatism and mission-grade feasibility: it enabled seamless integration into existing `Vault`-based encryption workflows, supported policy-bound key release and provided a reproducible deployment path within the PoC scope. While alternatives such as Kubernetes-native confidential container runtimes (e.g. the *Confidential Containers* project of the CNCF) offer promising long-term capabilities, they remain at *Incubating* maturity and lack full support for attestation, key provisioning and edge portability [47].

Accordingly, Kata was selected not due to formal endorsement by NAFv4 or NIST, but based on its practical integration maturity across heterogeneous sovereign environments. The resulting architecture balances strong isolation guarantees with the need for DevSecOps-compatible container lifecycles and deterministic attestation control, aligning with the operational and compliance objectives defined for confidential backend services.

Kubernetes-native Confidential Containers. While Kubernetes-native confidential computing runtimes —- such as confidential computing-enabled deployments using SEV-SNP —- offer promising long-term potential, their integration was deliberately excluded from this PoC. This decision reflects current maturity and tooling limitations, especially regarding remote attestation support, control-plane compatibility with `Vault` and the orchestration overhead associated with Kubernetes-managed runtimes in constrained environments. The use of Kata instead prioritised reproducibility, simplified attestation integration and full compatibility with existing OCI- and Kubernetes-based workflows.

Technology Decisions and Trade-Offs. The technology layer was structured around mission-informed trust domains. While the use of SEV-SNP-enabled virtual machines with *Kata Confidential Runtime* provided strong isolation and attestation guarantees for cloud-hosted backend workloads, such capabilities were not available on edge-class hardware. As a result, OP-TEE-enabled devices such as *Jetson AGX Orin* were employed

at the tactical edge, accepting trade-offs in terms of runtime confidentiality and remote attestation. Furthermore, Kubernetes-native service meshes — while considered — were consciously excluded in favour of static, file-based service bindings to minimise system complexity and ensure deterministic service orchestration under constrained network conditions. These decisions reflect a deliberate balance between trust anchoring, architectural expressiveness and deployment feasibility in defence-grade environments.

The architectural overview in this section has deliberately focused on a selected and traceable subset of NAFv4-compliant views: `NCV-2` (Capability Dependencies), `NCV-3` (Capability Roadmap), `NSOV-3` (Service Functions), `NSOV-6` (Service Orchestration Logic) and `NSV-1` (System Deployment). These views constitute the core architectural elements required to demonstrate semantic continuity from strategic objectives through service realisation to physical deployment environments.

Rather than exhaustively modelling all NAFv4 artefacts, the selected views were prioritised based on their architectural expressiveness, methodological necessity and feasibility within the scope of a time-bounded PoC. The resulting model enables robust traceability between high-level mission needs and operational capabilities, while supporting early design validation of interoperable, sovereign and secure defence systems.

It must be noted that the presented architectural artefacts reflect a first iteration and are subject to refinement. As the platform matures and stakeholder inputs evolve, the capability catalogue, service structure and interface logic will likely require further alignment and expansion. Nevertheless, the current model offers a structured and methodologically sound foundation for advancing to the logical design of platform services.

The following section (3.2) builds on this foundation by elaborating the architectural implications of cloud-native service deployment across sovereign cloud, fog and edge domains — highlighting how the architectural principles established here are practically operationalised in hybrid execution environments.

In summary, this chapter has laid the architectural foundation of the *OmniAware* platform by methodically selecting and modelling a strategically relevant subset of the NAFv4 viewpoint catalogue. Rather than pursuing exhaustive coverage, the modelling scope was deliberately constrained to enable traceable capability decomposition, service orchestration and sovereign system deployment within the boundaries of a time-constrained PoC.

The resulting architecture offers a semantically coherent and operationally viable model that links high-level strategic objectives to deployable infrastructure artefacts. These models provide the structural basis for implementation and compliance validation across diverse execution environments.

The next chapter shifts perspective from architectural abstraction to runtime environments. It examines how distributed computing paradigms — spanning sovereign cloud, fog and edge — bring the defined services to life, ensuring secure, scalable and mission-resilient operation in coalition-based deployments.

3.2 CLOUD, EDGE AND HIGH-PERFORMANCE COMPUTING

Summary: Chapter 3.2 operationalised the architectural abstractions from Chapter 3.1 by deploying them across distinct cloud, edge and high-performance computing environments. The respective AWS-based reference architectures illustrated how core capabilities—including PHM and CIVS — are realised in practice. A serverless-first approach was adopted to reduce system complexity, increase elasticity and ensure secure telemetry processing. Moreover, deployment-specific trade-offs such as audit account isolation, minimal latency at the tactical edge and sovereign cloud seperation were elaborated to accommodate the varying mission demands across coalition-based and national operations.

3.2.1 *Deployment Methodology and Realisation Approach*

Building on the capability- and service-centric foundations established in Section 3.1, this section advances into the deployment and infrastructure perspective of the *OmniAware* architecture. While the previous chapter addressed what the platform does, this section focuses on how and where it is executed.

To capture this shift, a refined subset of NAFv4 viewpoints was selected:

- **NSV-1 (System Deployment)** — depicts the runtime environment of services, distributed across cloud, fog and edge tiers.
- **NPV-1 (Architecture Roadmap)** and **NPV-2 (Lines of Development)** — structure the deployment logic across sovereign trust zones, physical cluster types and workload evolution paths.
- **NAV-1 (Standards and Reference Architecture)** — documents the architectural principles, deployment primitives and system classifications used in the PoC.

In line with the design scope of this thesis, the NSV-5 viewpoint was intentionally omitted. While NSV-5 typically addresses system interactions and orchestration sequences, the platform's event-driven, loosely coupled design renders a full NSV-5 modelling unnecessary at this stage. Interactions are instead reflected in the architectural layering and flow logic of the deployment model.

The following sections integrate a detailed breakdown of the system deployment logic, guided by two visual reference architectures: one for PHM and one for CIVS. These artefacts were collaboratively developed with a Lead Cloud Architect, who was primarily responsible for the structural and compositional design of the overall cloud deployment. Selected architectural decisions were contributed and security-related components, including encryption, key management and attestation flows were integrated. The reference architectures were used to validate the technical feasibility and component interplay across AWS Fargate, EC2-based enclaves and edge-deployed Lambda services. They represent the operationalisation of the previously defined service and capability landscape in a federated, secure and mission-resilient execution context.

As part of the Strategic Collaboration Agreement (SCA) between Capgemini and Amazon Web Services (AWS), all architectural designs and implementation artefacts are required to adhere to the principles of the **AWS Well-Architected Framework (WAF)**. This framework serves as a best-practice guideline across five key pillars: operational excellence, security, reliability, performance efficiency and cost optimisation [44].

In accordance with the milestone plan defined for the MVP development phase, a formal **Well-Architected Review** was conducted to validate the alignment of the *OmniAware* system architecture with AWS expectations. As a result, the detailed PHM and CIVS reference architectures were not only modelled for internal validation and communication purposes, but also to fulfil compliance and auditability requirements within the WAF-based project governance structure.

To support this process, the architectural diagrams were implemented using **Draw.io** templates derived from the AWS Architecture Icons and tailored to reflect the services, boundaries and responsibilities relevant to defence-grade deployments. Each architecture is organised to illustrate the logical separation of service domains (e.g. ingestion, analytics, datalake, consumer, audit) while embedding core WAF-compliant practices such as encryption at rest and in transit, Identity and Access Management (IAM) scoping, audit log retention and service-level decoupling.

Service Selection Transparency. Each AWS service included in the architecture fulfils a distinct mission-related role. CloudTrail and CloudWatch Logs provide event-level observability, while Macie supports anomaly detection on telemetry metadata. SageMaker enables enclave-compatible model inference and Quicksight supports mission-state visualisation. All services were selected based on their ability to embed auditability, scalability and attested processing into the operational workflow of the PHM and CIVS scenarios.

This approach ensured that both use-case specific implementations (PHM and CIVS) were not only operationally sound but also **WAF-ready** — allowing them to pass the initial milestone review criteria and qualify for further enablement support through the Field Ready Kit and Foundational Technical Review (FTR) preparation.

To complement the formal architecture views, the deployment model is supported by two reference architectures that were collaboratively developed during the project. These diagrams represent the practical translation of capability and service design into technical artefacts, including security primitives, enclave deployment topologies and federated service chains.

3.2.2 *Contribution and Project-Specific Realisation*

This section presents the deployment-centric complement to the modelling artefacts discussed in Chapter 3.1. It reflects the implementation-oriented work, including the derivation of service deployment blueprints, operational trust zones and architecture-driven infrastructure abstractions for the *OmniAware* platform. The technical realisation is grounded in previously modelled capabilities and service flows, bridging architectural design with runtime instantiation.

The contribution is structured around two main pillars:

Infrastructure Realisation. This includes the development and black box abstraction of AWS-based reference architectures for the PHM and CIVS scenarios, structured around NAFv4-compliant views such as NSV-1, NSV-4a and NPV-3. The reference architectures reflect trust-segmented infrastructure domains, identity-enforced runtime boundaries and confidential computing patterns. Each artefact was manually derived and visualised using draw.io templates aligned with the author's modelling decisions from 3.1, ensuring traceability from logical capabilities to operational infrastructure.

Security Blueprinting and Author Contribution. In addition to structural deployment views, this thesis contributes reference implementations for secure service orchestration and workload protection. These include enclave-enabled runtime clusters (based on SEV-SNP and OP-TEE), policy-based secret distribution (via `Vault` and KMS), as well as WAF-aligned design patterns for IAM scoping, audit integration and network zoning. These elements were implemented and documented by the author and serve as deployable validation artefacts within the PoC, supporting auditability, security evaluation and compliance demonstration.

3.2.3 *AWS Well-Architected Framework - Reference Architecture*

The realisation architectures presented in this section are grounded in the capability decomposition modelled in `NSV-1` and `NPV-2`, ensuring architectural consistency between abstract capability definitions and concrete infrastructure deployments. Each depicted service in the PHM and CIVS reference views corresponds to a mission-specific capability cluster —– such as `C10_Vehicle Analytics` or `C4_Confidential Computing` —– and reflects the validated implementation path through dedicated AWS-native components. The mapping was derived to preserve service modularity, data flow integrity and zero-trust execution boundaries as formalised in the respective viewpoint models. This section introduces the reference architecture developed in accordance with the AWS Well-Architected Framework (WAF) and aligned with security, compliance and performance expectations defined in the Strategic Collaboration Agreement (SCA) between Capgemini and AWS. It serves as the conceptual backbone for the PHM and CIVS deployment views and contextualises their mapping to security-aligned infrastructure patterns.

The reference architecture reflects the implementation of WAF principles across the five pillars — operational excellence, security, reliability, performance efficiency and cost optimisation. As such, it represents a validated design scaffold that underpinned both the initial WAF Review and the development of deployable Landing Zone artefacts for the PoC.

High-Level Architecture Modelling. The architecture provides an abstracted black box representation of *OmniAware*'s core service domains. It outlines the separation of concerns across ingestion, analytics, API exposure, audit and management, while encapsulating key functional building blocks. Each architectural block corresponds to a logical service grouping defined in earlier NSOV-3 and NSOV-6 models.

Account Separation and Trust Segmentation. The reference model includes dedicated AWS accounts for ingestion, datalake, audit and security, aligned with organisational and runtime trust boundaries. This structure reflects NPV-3 requirements for enforced trust zones and maps to deployable blueprints supporting role isolation, encrypted data transfer and minimal exposure of high-sensitivity artefacts.

Security-by-Design Integration. The model embeds core WAF-aligned security primitives, including IAM scoping, secure key handling via KMS and audit tagging. Sensitive paths are constrained through attested components such as enclave-enabled compute nodes and policy-controlled boundary enforcement. These principles directly influence the Landing Zone definitions described in Sections 3.2.4 and 3.2.5.
The high-level reference architecture provides an abstracted view of core *OmniAware* deployment principles. It emphasises the separation of concerns across ingestion, storage,

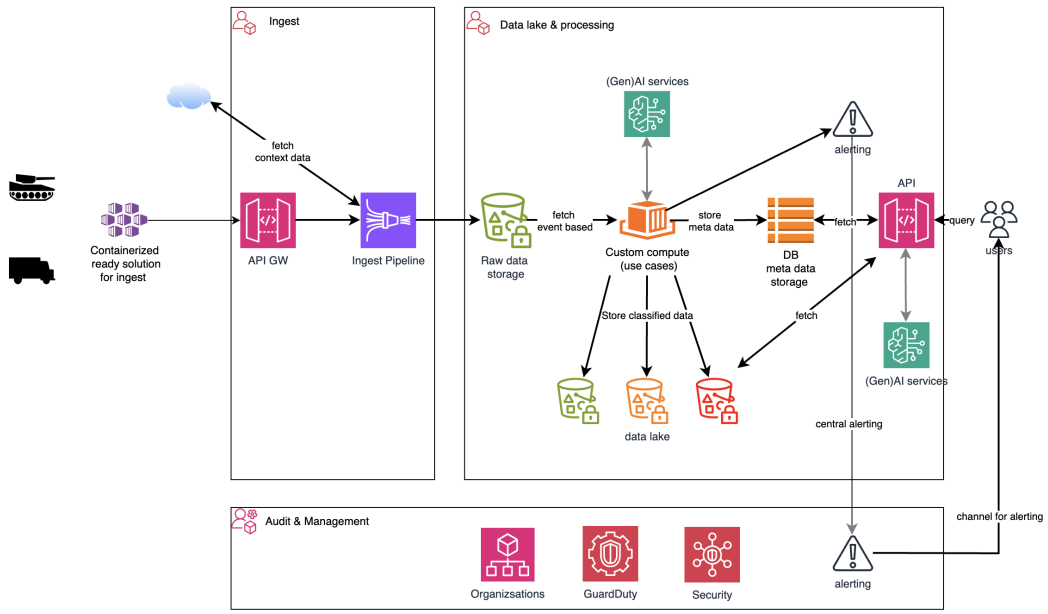


Figure 3.6: *OmniAware* Deployment Principles - High-Level Reference Architecture [80]

processing, API exposure and alerting layers. Additionally, it highlights security-by-design elements such as dedicated accounts for audit and key management services. This abstraction serves as a conceptual introduction to the PHM and CIVS reference architectures and contextualises their cloud-native service orchestration patterns.

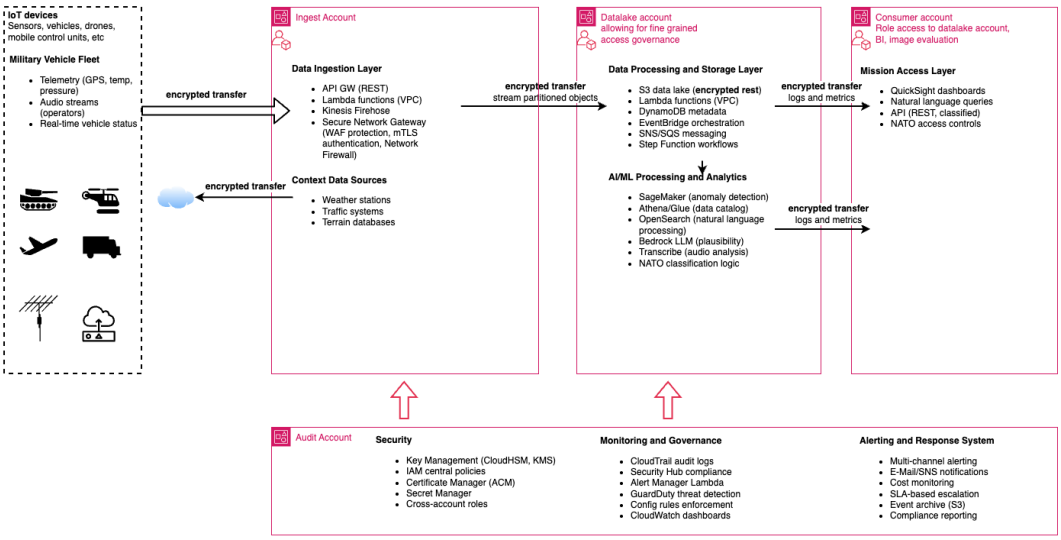


Figure 3.7: PHM - High-Level Overview of the Reference Architecture

The *PHM reference architecture* models a telemetry-based edge-cloud pipeline that enables RT health assessment of tactical platforms. It integrates attested compute nodes, enclave-secured workloads and secure message ingestion. The architecture includes enclave runtimes using SEV-SNP and OP-TEE technologies, supporting trusted container execution and confidential telemetry analytics.

The *CIVS reference architecture* illustrates a secure image ingestion and validation

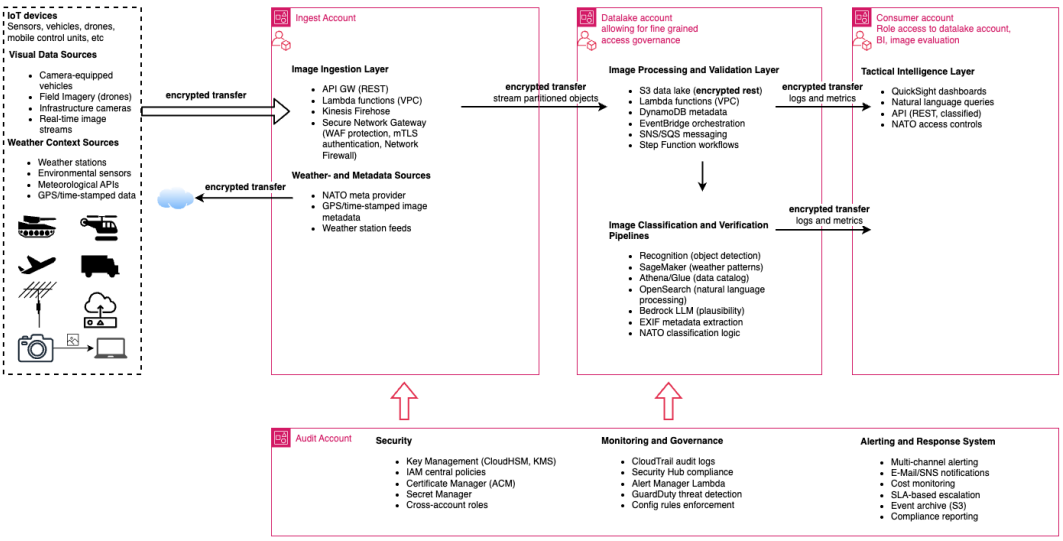


Figure 3.8: CIVS - High-Level Overview of the Reference Architecture

flow that includes confidential classification, pattern recognition and cross-validation against external authoritative sources. This pipeline enables verifiable mission context inference under degraded trust conditions.

Given its broader scope and technical maturity, this section begins with the Platform Health Monitoring (PHM) use case. Compared to other scenarios, PHM encompasses a more complete set of architectural components — from ingestion and storage to analytics and auditability — and therefore serves as a suitable example for illustrating the reference architecture developed as part of the *OmniAware* platform. The selected architectural approach reflects the principles of account separation, secure service orchestration and scalable data management within a defence cloud context. The PHM scenario thus forms the primary basis for the detailed architectural walkthrough. In contrast, the Contextual Image Verification System (CIVS) is introduced subsequently in abbreviated form to outline its specific adjustments and differences relative to the same architectural framework.

The PHM system is structured into five principal AWS accounts, each mapped to a functional architectural layer:

- **PHM Ingest Account:** Responsible for the secure collection and preprocessing of telemetry data streams from operational platforms. It employs edge-integrated data acquisition agents and secure ingestion endpoints, supporting both batch and streaming paradigms.

- **PHM Data Lake Account:** Provides scalable, schema-flexible storage for structured and unstructured data, leveraging services such as Amazon S3, Glue Catalogues and Lake Formation. It supports versioning, tagging and data lineage tracking to ensure data governance and auditability.

- **PHM Consumer Account:** Hosts mission analytics pipelines, anomaly detection services and dashboarding interfaces. It integrates with AWS analytics services such as Athena, SageMaker and QuickSight and enables policy-driven access via IAM and RBAC.

- **PHM Audit Account:** Implements independent security monitoring and operational logging via services such as CloudTrail, AWS Config and Security Hub. It facilitates continuous compliance validation and decoupled audit operations across accounts.
- **Shared Services Account:** Provides core infrastructure services including VPC peering, DNS resolution, central logging sinks and shared data models. It also acts as a control plane for cross-account orchestration and backup strategies.

Each account is configured with dedicated `Service Control Policies (SCPs)` and tightly scoped IAM roles, ensuring that only explicitly defined cross-account interactions are possible. This design principle enables zero-trust isolation while preserving operational collaboration across the PHM platform lifecycle. The architectural model aligns with the principles of minimal blast radius, scalable domain separation and policy-driven service interconnection.

Differentiation from CIVS: The reference architecture for the Contextual Image Verification System (CIVS) reuses the general pattern of account separation but adapts the specific services and data flows to meet image processing requirements. Unlike PHM, which focuses on structured sensor telemetry and time-series analytics, CIVS handles media-rich, often classified data with higher sensitivity. It introduces hardened ingestion pipelines for tactical imagery, image validation workflows and cross-referencing capabilities with external authoritative data providers. Additional compliance controls are applied, such as automated image classification and redaction pipelines, to satisfy mission-specific security constraints.

The **Ingest Account** serves as the primary entry point for telemetry and contextual data streams into the *OmniAware* platform. It hosts the secure intake of operational sensor data from vehicles, UAVs and edge platforms, while enabling data enrichment via publicly available contextual sources (e.g. weather services). Its architecture ensures that data is securely transmitted, filtered and pre-processed before being forwarded to the central processing layers.

As illustrated in Figure 3.9, the ingestion pipeline is initiated through an `API Gateway` configured for secure HTTPS access, backed by the `Amazon Certificate Manager` issuing custom TLS certificates. This guarantees transport layer encryption and identity-bound access to telemetry sources. In parallel, external contextual information — such as weather patterns and terrain data — is fetched via scheduled `Lambda` functions. These background enrichment jobs are protected by `Network Firewalls` and routed through `VPC Endpoints` to isolate them from the public internet.

The core of the ingestion logic is implemented via `AWS Lambda` and `Amazon Kinesis Firehose`. Lambda functions classify incoming messages by type (e.g. image, telemetry, audio), perform basic schema validation and apply initial metadata tagging. This ensures downstream services receive semantically structured payloads. Firehose transforms and buffers the data (e.g. to Apache Parquet or Iceberg formats), enabling retries and fault isolation. Partitioning schemes are applied based on mission ID and source class, with schema evolution tracked in the `Glue Schema Registry`.

Security controls embedded into the account include KMS-managed encryption of payloads at rest, protection against Distributed Denial-of-Service (DDoS) via AWS Shield and WAF-based request inspection for API access. Monitoring and alerting are implemented using CloudWatch and EventBridge, which track ingestion throughput, transformation failures and operational anomalies. Critical audit logs are forwarded to a decoupled Audit Account for long-term retention and compliance.

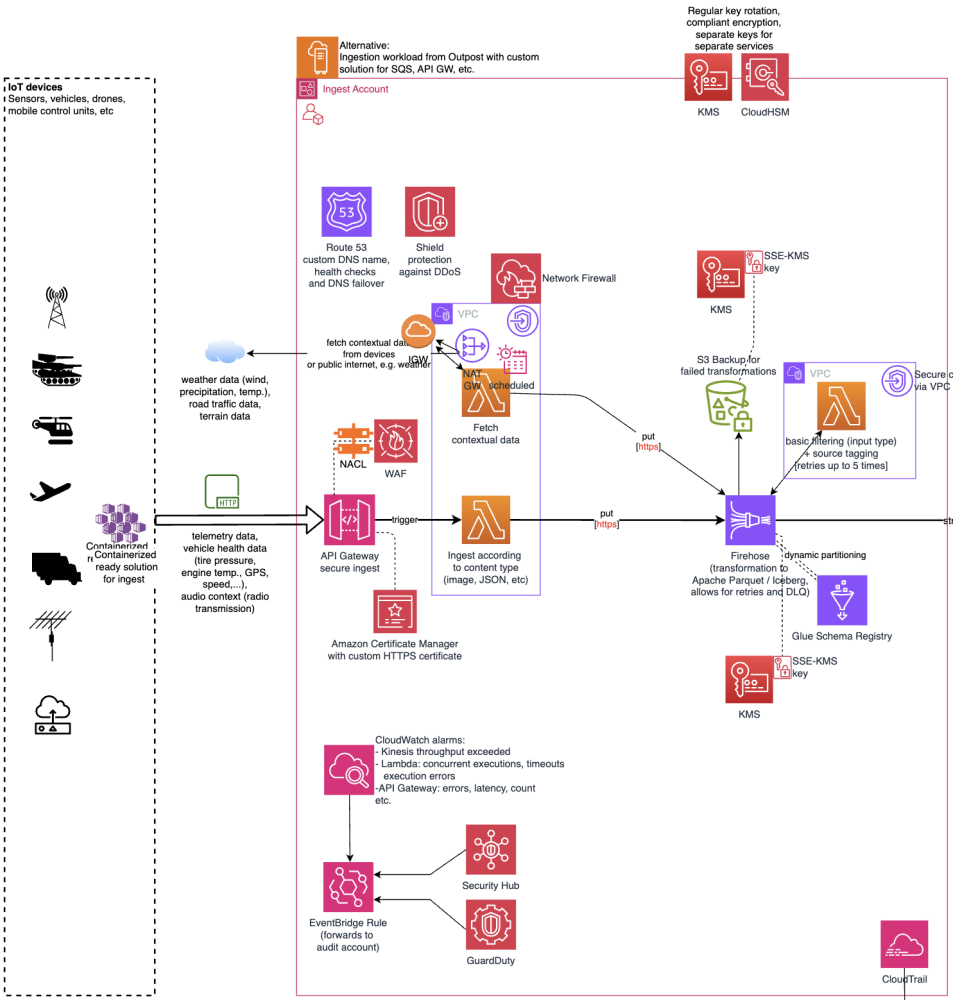


Figure 3.9: Ingest Account (PHM) - Secure telemetry intake and contextual enrichment pipeline [80]

The **Data Lake Account** forms the analytical and storage-centric core of the PHM architecture. It is responsible for securely ingesting, persisting and processing telemetry and contextual data within a scalable, policy-enforced cloud environment. While primarily serving as the platform's long-term data repository, this account also hosts mission-specific compute logic to enable near-real-time processing and anomaly detection workflows.

Event Pipeline Decoupling in PHM. The telemetry pipeline is intentionally decomposed into discrete `NSOV-3` functions —— each realised as a serverless Lambda function —— that operate in an event-driven chain. This architectural pattern enables asynchronous scaling, functional traceability and modular error handling while supporting auditability across pipeline stages.

As shown in Figure 3.10, incoming data is first written to a raw storage tier configured in `compliance mode`, ensuring immutability and traceable auditability. All data objects are encrypted with service-specific `SSE-KMS` keys to maintain confidentiality throughout the ingestion lifecycle. Event-driven triggers initiate downstream compute modules, typically realised as container-based workloads or serverless functions, which perform classification, contextual enrichment or plausibility evaluation.

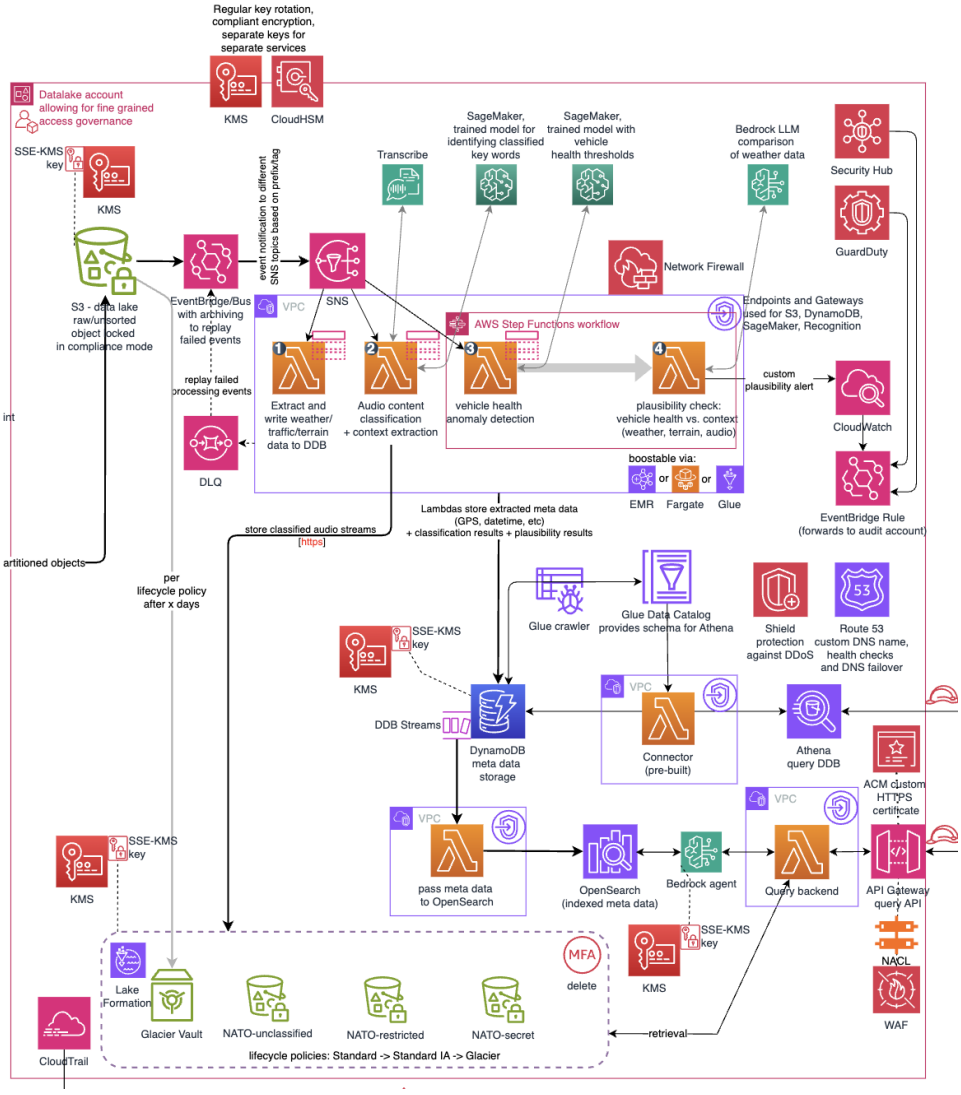


Figure 3.10: Data Lake Account (PHM) - Mission analytics, metadata storage and classified data lifecycle enforcement [80]

The **Consumer Account** provides the user-facing access layer of the PHM architecture. As shown in Figure 3.11, this account exposes selected outputs from downstream analytics pipelines to authorised users via controlled, queryable interfaces. It allows mission personnel and analysts to interact with PHM results without compromising the integrity or classification boundaries of upstream data processing.

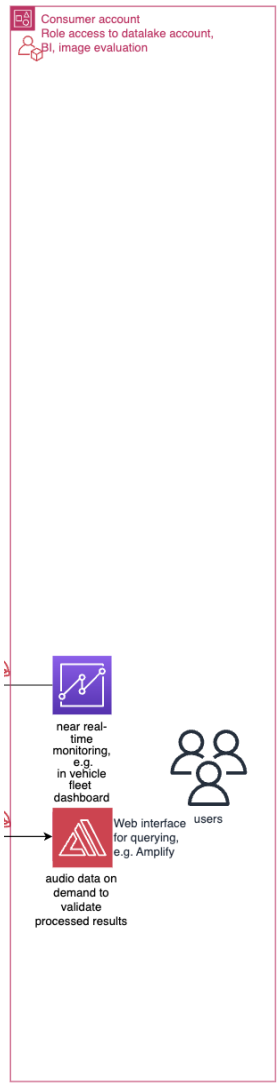


Figure 3.11: Consumer Account PHM - Presentation and Interaction Layer [80]

The core component is a secured `API Gateway`, which mediates all external requests and enforces authentication and authorisation policies via scoped `IAM` roles. This design ensures that only explicitly permitted user groups (e.g. operational command, logistics or vehicle fleet management) can access telemetry-based findings. Role-based controls are enforced to align with mission confidentiality and coalition interoperability requirements.

Presentation logic is handled via web applications (e.g. `Amplify`), offering dashboards and interactive interfaces that support mission-state visualisation, anomaly review and contextual alerts. Near real-time updates (e.g. via WebSockets or event-driven notification services) can be triggered for critical fault indicators, ensuring that end users are informed promptly and securely.

Optionally, advanced business intelligence (BI) services (e.g. `Quicksight`) may be integrated to visualise statistical trends, usage anomalies or mission readiness states. These tools enable operational decision-makers to derive strategic insights while ensuring that no raw telemetry or classified metadata is directly exposed.

By decoupling analytics exposure from the main processing accounts, the Consumer Account upholds the principle of least privilege and strengthens horizontal separation between processing and presentation. Its minimal trust surface and role-scoped access model contribute significantly to platform-wide confidentiality and mission assurance.

Mission-specific processing logic is encapsulated within `custom compute modules`, which may invoke pre-trained `GenAI models` hosted on managed services such as `SageMaker`. Extracted metadata — such as timestamps, GPS coordinates or anomaly scores — is stored in a dedicated `DynamoDB` metadata repository to support rapid querying and operational alerting via API.

The Data Lake itself is logically partitioned according to NATO classification levels (e.g. Unclassified, Restricted, Secret) and enforces strict encryption key separation. Lifecycle policies ensure automatic tiering across storage classes (e.g. Standard → Infrequent Access → Glacier), optimising both cost efficiency and compliance. A `Glue Data Catalog` maintains schema consistency across datasets and enables serverless analytical access through `Athena` or `OpenSearch` indexing.

Compliance and observability are ensured through integrated logging via `CloudTrail`, vulnerability scanning by `Amazon Inspector` and security findings aggregation in `Security Hub`. Failed processing attempts are redirected into a `Dead Letter Queue (DLQ)`, enabling traceability and reprocessing logic. This account thereby encapsulates both persistent storage and mission-aligned compute logic under sovereign control.

The **Audit Account** is dedicated to the collection, verification and long-term preservation of security-relevant events across the PHM system landscape. As illustrated in Figure 3.12, it enables centralised auditability, forensics and compliance enforcement by integrating logging, threat detection and compliance automation capabilities.

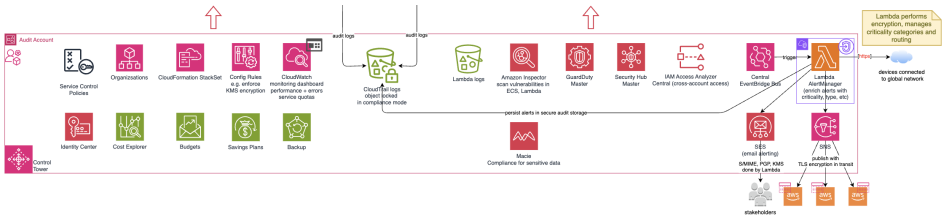


Figure 3.12: Audit Account PHM - Cross-Cutting Security and Compliance Capabilities [80]

All services and resources deployed in the Ingest, Data Lake and Consumer accounts stream their security events, access logs and telemetry into centralised, immutable stores hosted in the Audit Account. Services such as `CloudTrail`, `CloudWatch Logs` and `VPC Flow Logs` ensure fine-grained visibility into control plane, network and application-level actions. These logs are then processed and archived using long-term, write-once-read-many (WORM) storage (e.g. `S3` with Object Lock), satisfying auditability and evidence preservation requirements for sensitive military workloads.

A key architectural feature is the segregation of audit operations from operational workloads. This ensures that no user or workload can tamper with logs or monitoring policies from within the mission systems. Instead, the Audit Account maintains read-only cross-account roles with access to compliance-relevant telemetry and is governed

by a separate administrative domain with minimal access surface.

Advanced threat detection is enabled via services such as `GuardDuty`, which continuously analyses log data for anomalies and potential breaches. Findings are programmatically forwarded to an SIEM or central SOC environment, supporting near real-time incident response. Additionally, `AWS Config` and `Security Hub` provide configuration drift detection and compliance reporting aligned with military security frameworks such as NIST 800–53, ISO/IEC 27001 or NATO D/32.

Due to its foundational role in maintaining security assurance, a separate **Security Account** is introduced to offload critical monitoring services (e.g. `Inspector`, `Macie`) from the Audit domain and to enforce stricter role isolation. The Audit Account therefore functions as a forensic archive and compliance anchor, while the Security Account executes active scanning, classification and vulnerability management.

This two-tiered model reduces blast radius, increases tamper resistance and aligns with Zero Trust principles. It ensures that audit and compliance functions operate under strict role isolation and independent control paths, as expected in defence-grade architectures.

While the CIVS architecture builds upon the same core account segmentation and orchestration logic as the PHM use case, several service components merit closer examination due to their heightened security implications.

One notable difference lies in the hardened ingestion pipeline for classified imagery data. This includes additional pre-processing layers for pattern-based filtering, classification tagging and plausibility validation, all executed within enclave-attested `Lambda` functions. These steps are explicitly designed to mitigate the risk of tampered or manipulated visual content being introduced into the system.

Moreover, the use of `Amazon Rekognition Custom Labels`, tailored to CIVS-specific mission parameters, introduces a potential attack surface requiring runtime attestation and policy-constrained key access. In combination with `SageMaker`-based anomaly detection and `Bedrock`-backed LLM correlation checks, these services necessitate fine-grained auditability and cross-account alert routing, which are realised through a reinforced EventBridge and Simple Notification Service (SNS) security path.

Finally, given the higher classification level of image data processed within CIVS, stricter KMS segmentation and compliance tagging are enforced across the `Data Lake Account`, following distinct NATO-aligned lifecycle policies for storage, deletion and retrieval.

This architectural divergence further emphasises the need for isolated security controls, which are addressed in the following section through the introduction of the dedicated *Security Account*.

In accordance with the AWS Well-Architected Framework, the architectural blueprint separates high-privilege security services from general-purpose audit and monitoring accounts. The **Security Account** is dedicated to managing encryption keys (KMS), vulnerability scanning (Inspector), web application firewall rules (WAF) and custom HTTPS certificates. This segregation reinforces the principle of least privilege and facilitates compliance with defence-specific security baselines, especially when integrating sovereign cloud controls and cross-domain processing mechanisms.

While the technical lead for the overall architectural design was assumed by the project's cloud lead architect, the security controls and confidential computing components within both reference architectures were specifically developed under my responsibility. This includes the definition of enclave enforcement logic, remote attestation flows and policy-driven key release mechanisms across all deployment layers.

These artefacts serve not only as architectural visualisations but also as deployment

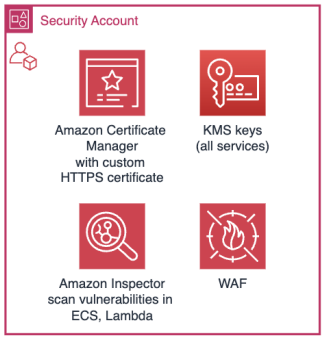


Figure 3.13: *OmniAware* Security Principles - Security Account for Cross-Service Security Functions [80]

planning blueprints that were reused for the validation of service flows in Chapter 3.1 and security assurance models in Chapter 3.2.

Having detailed the AWS Well-Architected Framework and the corresponding reference architectures for the *OmniAware* platform, the subsequent section explores the integration and application of HPC within this architectural context. High-Performance Computing capabilities, while initially considered complementary, become increasingly critical as operational scenarios evolve towards advanced predictive simulations and intensive analytical workloads. The following section thus discusses how HPC elements align with and extend the established architectural principles, focusing particularly on secure and confidential execution in sovereign environments.

The reference architecture presented in this section establishes a validated deployment blueprint grounded in the principles of the AWS Well-Architected Framework and tailored to defence-specific requirements through the SCA collaboration. By structuring the platform around dedicated accounts, isolated trust zones and WAF-compliant design patterns, it enables modular, auditable and secure service orchestration. The following sections apply this validated architecture to concrete use case scenarios — beginning with the PHM deployment model — to illustrate how the defined architectural principles translate into executable cloud-native infrastructure within a mission-centric context.

Service-Level Design Rationale for Capability Realisation. The selection of AWS-native services in the PHM and CIVS reference architectures was driven by the objective to map each mission-relevant capability to a well-defined, scalable and governable service implementation.

For instance, in the PHM scenario, telemetry ingestion is realised using `Kinesis Firehose`, chosen for its ability to buffer high-throughput sensor data with minimal operational complexity. The signal classification logic is deployed as an isolated AWS Lambda function, supporting immutable code artifacts and deterministic invocation patterns. All telemetry classification results are persisted in `DynamoDB` to ensure low-latency access and serverless scalability, while an `Audit Account` handles forwarding to WORM-enabled S3 buckets via cross-account policies, fulfilling forensic compliance requirements.

Similarly, in the CIVS architecture, image ingestion is orchestrated through EventBridge rules that route image references to multiple Lambda-based service functions. These include pre-processing, `AWS Rekognition` for object detection and a redaction module which leverages `OpenCV via Lambda Layers` to ensure extensibility. The separation into discrete services allows traceable compliance with classified content handling

policies and facilitates future upgrades, such as the integration of confidential inference via Nitro Enclaves.

Each architectural decision thus serves to align functional decomposition with deployment modularity, account-level governance and attestation compatibility. This service-to-capability mapping forms the core of the architecture's compliance-aware design philosophy.

Capability-Driven Deployment Mapping. The mapping between platform capabilities and cloud-native services is guided by mission-specific requirements. For example, `C1_Cloud Computing Platform` leverages AWS Lambda and EC2 instances to ensure modular orchestration and managed backend execution. `C4_Confidential Computing` integrates `Kata`-based runtimes on SEV-SNP-enabled VMs to enable attested microservices for secure data-in-use processing. `C10_Vehicle Analytics` builds on GPU-accelerated `Jetson AGX Orin` edge devices, reflecting latency-aware trust zone separation and policy-based inference orchestration.

Deployment Rationale. To maximise operational agility and minimise infrastructure management overhead, the architecture intentionally prioritises AWS-managed services over self-managed EC2 deployments. Serverless technologies such as Lambda, Fargate and Kinesis Firehose enable rapid prototyping, embedded compliance (e.g. IAM scoping, KMS encryption, CloudTrail auditability) and streamlined cost control in line with WAF best practices. EC2-based enclave deployments were selectively used for confidential workloads requiring SEV-SNP attestation in sovereign environments where Nitro Enclaves are not supported.

While confidential computing integration and the inclusion of TEE runtimes such as Kata played a pivotal role in securing data-in-use, several additional architectural decisions shaped the system model.

First, the deliberate use of NAFv4 views — specifically `NSOV-3`, `NSOV-6` and `NSV-1` — provided a structural anchor to align capability decomposition with deployable service components.

The adoption of the *serverless-first* paradigm reflected a strategic design choice to reduce operational overhead, streamline resource management and avoid the complexity of self-managed container orchestration. This decision is operationalised through managed services such as AWS Lambda, EventBridge, DynamoDB and Step Functions, which provide scalable, attested and fully decoupled runtime environments.

Further, the PHM and CIVS reference architectures (cf. Figures 3.7 and 3.8) implement strict account segmentation and trust zone isolation, which serve as runtime boundaries in accordance with the NSV-1 execution model.

From a composability perspective, each *service function*—as modelled in `NSOV-3`—was implemented as a logically separable unit within a dedicated cloud service account. This enabled policy-based decoupling of security primitives, lifecycle management and infrastructure concerns.

Finally, the explicit exclusion of Kubernetes-native confidential containers and service meshes reflects a pragmatic trade-off. While promising, their current limitations in attestation support, integration maturity and sovereign trust enforcement were deemed insufficient for deployment in a constrained PoC setting.

In sum, the architectural decisions embody a hybrid principle: capability-driven abstraction rooted in NAFv4 traceability, combined with practical cloud-native realisations that balance composability, manageability and defence-aligned compliance boundaries.

Model-Realisation Divergence and Kubernetes Deferral in Q1-Q2. The realised architecture in Q1 and Q2 reflects a pragmatic instantiation of the modelled capabilities using AWS-native serverless primitives, such as Lambda, Eventbridge and Firehose, to minimise operational complexity and accelerate time-to-value in a tightly scoped environment. Rather than deploying full Kubernetes orchestration, the system employs lightweight, modular services to validate core functional paths under real-world constraints, laying the groundwork for future orchestration extension.

This deviation reflects a conscious prioritisation of minimal viable product (MVP) delivery over full container orchestration. Particularly in mission-grade environments with strict compliance requirements and auditability constraints, the use of serverless services ensures better traceability, lower overhead and immediate integration with AWS's WAFR-aligned controls.

Accordingly, the current reference architectures (Figures 3.7 and 3.8) capture a streamlined, account-isolated structure that omits Kubernetes control planes, container orchestration layers or service mesh logic. These elements may be introduced in future capability expansions (cf. Q3-Q4 roadmap) once platform maturity and compliance state permit. The modelled design thus intentionally diverges from the actual PoC deployment topology to maintain forward compatibility without overengineering early phases.

3.2.4 *High-Performance Computing*

Although not the primary focus of the current implementation, selected architectural design elements were developed with High-Performance Computing (HPC) use cases in mind. These include defence-relevant workloads such as digital twin simulations, predictive maintenance, federated model training and encrypted batch processing across secure sovereign clouds.

Within the scope of the *OmniAware* platform, HPC integration was considered primarily at the cloud tier (e.g. AWS EU Central 1), where advanced compute resources including GPU/TPU-based instances, high-throughput networking (e.g. ENA or SR-IOV) and NUMA-aware scheduling are available. In combination with SLURM or container-native batch systems (e.g. AWS Batch), this enables parallel execution of simulation-heavy tasks and compute-intensive AI training.

Security and compliance requirements were a key design driver. In this context, HPC workloads can be executed inside SEV-SNP protected virtual machines, ensuring cryptographic isolation and policy-enforced key provisioning for mission data. Although a performance trade-off exists due to hardware-based memory encryption and attestation flows, these measures are justified by the required data-in-use confidentiality guarantees [61].

Initial explorations also considered lightweight, edge-deployable HPC variants (e.g. GPU-enabled Jetson Orin clusters in the CIVS context), which could serve as decentralised inference backends or simulation nodes. However, limitations in energy efficiency, attestation compatibility and orchestration tooling currently constrain their practical deployment in defence-grade scenarios.

This section outlines only preliminary considerations. A more comprehensive HPC integration — including confidential containerised scheduling, real-time workload offloading and sovereign AI training pipelines — is foreseen for future capability expansions.

While the reference architectures presented above serve to visualise the concrete deployment structure, including the services allocated across the various cloud accounts (e.g. Ingest, Datalake, Audit), their primary purpose lies in contextualising the platform design within a distributed computing paradigm. These architectural visualisations enable the mapping of deployed services to the underlying layers of cloud, edge and HPC infrastructure, thereby bridging operational capabilities with deployment-specific constraints.

To complement the architectural discussion above, the following NAFv4-compliant models (`NSV-1`, `NPV-1`, `NPV-2`, `NAV-1`) visualise selected aspects of the *OmniAware* system's deployment and standard alignment. These views formalise the platform's layered deployment logic, service allocation and classification structures in accordance with the NATO Architecture Framework methodology.

3.2.5 *Deployment, Compliance and Structural Governance*

To complement the formal introduction of the NSV-1 view in Section 3.1, the following discussion situates the system deployment logic within the broader context of cloud, edge and high-performance computing paradigms as defined in this section. While Figure 3.5 has already illustrated the runtime alignment of services across trust zones, this section now elaborates how the deployment model supports federated execution across distributed tiers.

Building upon the system deployment view previously introduced in Figure 3.5, this section recontextualises the NSV-1 artefact with respect to runtime tiering across cloud, edge and high-performance computing environments.

The NSV-1 model in Figure 3.5 visualises the runtime alignment of core services in the *OmniAware* architecture across differentiated execution tiers, governed by trust boundaries and mission scope. In the context of **cloud computing**, general-purpose backend functions — such as `S1_Ingestion Service`, `S2_Secure Storage` and `S3_Metadata Management` — are deployed in memory-encrypted virtual machines (VMs) backed by SEV-SNP, offering full-stack confidentiality through enclave-supported `Kata Confidential Runtime` and policy-bound key release.

Edge deployments are characterised by highly mission-specific execution logic — including PHM- and CIVS-bound components such as `S10-S13` and `S20-S23` — executed in `OP-TEE`-enabled embedded compute units (e.g. `Jetson AGX Orin`). These edge workloads represent trusted execution cores operating under degraded connectivity and resource constraints, while maintaining local assurance guarantees through embedded runtime integrity.

While not explicitly modelled in the current artefact, **high-performance computing** (HPC) contexts are implicitly enabled via enclave-compatible cloud compute nodes that support enclave-aware containerisation, for example using attested `Kata`-based orchestration in GPU-accelerated EC2 or `Fargate` workloads. These deployments are architected to offload model-intensive analytics (e.g. fault prediction, cross-validation or simulation) from latency-sensitive edge tiers into confidential backend services.

In summary, the NSV-1 view reflects a tiered deployment strategy, aligning services with the optimal execution environment based on latency, confidentiality and mission relevance — thereby ensuring secure, scalable and mission-resilient system operations across federated cloud-edge infrastructures.

The NPV-1 view, illustrated in Figure 3.14, models the planned evolution of key architectural capabilities over time, following the NATO Architecture Framework's

temporal logic. It aligns technical deliverables with defined quarterly milestones and maps capability chaining to ensure architectural and operational consistency.

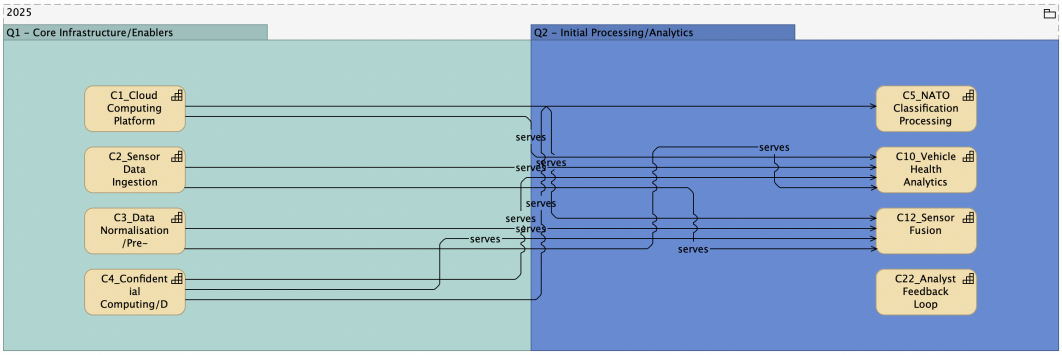


Figure 3.14: NPV-1: *OmniAware* Architecture Roadmap (Q1-Q2)

As shown in the figure, `Q1` capabilities — including `C1_Cloud Computing Platform`, `C2_Sensor Data Ingestion`, `C3_Data Normalisation/Pre-Processing` and `C4_Confidential Computing/Data Sovereignty` — establish foundational infrastructure for secure execution and compliant data handling. These services enable the realisation of `Q2` capabilities in *Initial Processing/Analytics*, such as vehicle telemetry analytics (`C10`), NATO classification pipelines (`C5`) and sensor fusion logic (`C12`), all of which depend on pre-ingestion processing and trusted compute environments.

The directional `serves` relationships model the dependency flow between capability layers, ensuring that mission logic is grounded in validated infrastructure. Capabilities like `C12_Sensor Fusion` and `C22_Analyst Feedback Loop` operate as key integrators, linking upstream infrastructure with downstream decision-making workflows.

Notably, the current model is scoped to the capabilities of **Q1 and Q2**. While the overall roadmap spans four quarters, modelling for **Q3 and Q4** has been deliberately omitted. This decision reflects the project's present maturity level and the evolving nature of stakeholder priorities within the SCA framework. As the strategic direction and technical priorities for these later quarters are still undergoing refinement, omitting them avoids premature specification and overcommitment to unstable milestones.

In summary, the NPV-1 deployment roadmap provides a visual and semantically structured representation of early-stage architectural progression. It captures the realised and planned capabilities under the PoC scope and ensures coherence with the validated scenario logic underpinning the *OmniAware* platform.

The NPV-2 elaborates the physical instantiation of capabilities over time by structuring them into **Lines of Development** (LoDs). Each LoD represents a coherent sequence of technical activities, infrastructure evolutions and mission-focused deliverables aligned with defined operational contexts. It provides a temporal and structural logic for how capabilities progress from planning to realisation.

Building on the layered semantics of NAFv4, the NPV-2 model formalises how abstract capability elements defined in NCV-2 are instantiated as technology-specific deployment artefacts — including software containers, VMs, enclave-enabled runtimes and edge compute platforms. This mapping leverages `realises` relationships to trace capabilities to their physical execution layers, ensuring consistent alignment with mission needs, delivery environments and security postures.

In the case of *OmniAware*, two principal Lines of Development have been identified. The first is centred around the PHM use case, comprising capabilities related to telemetry ingestion, secure processing and edge analytics. The second addresses

the CIVS scenario, where secure image handling, contextual verification and federated analysis are prioritised. These LoDs serve as structural backbones for engineering and operational rollouts, ensuring continuity across development phases and cross-account security boundaries.

Deployment Alternatives. While EC2-based deployments offer flexibility and enclave integration (e.g. SEV-SNP), they require manual orchestration, patching and lifecycle control — which were deemed impractical for a time-boxed PoC with limited operational scope. By contrast, AWS-managed services abstract infrastructure overhead while embedding WAF-aligned security primitives. Alternative technologies such as Fargate for confidential workloads or Intel TDX-backed clusters were evaluated but ultimately excluded due to edge portability constraints or insufficient ecosystem maturity.

Figure 3.15 illustrates the currently implemented NPV-2 model for the first two quarters, reflecting only those capabilities that are scheduled for realisation during the core duration of this thesis project. This scope limitation aligns with the fiscal segmentation imposed by the Strategic Collaboration Agreement (SCA), whereby `Q1` ends in April and `Q2` in July 2025. As the roadmap beyond this period remains subject to change, capabilities planned for `Q3` and `Q4` have deliberately been excluded to avoid speculative modelling.

Scope Delimitation. This modelling decision reflects a conscious trade-off between architectural completeness and implementation fidelity. Rather than extending the deployment roadmap beyond the maturity level achievable within the thesis timeline, the scope was deliberately confined to Q1-Q2 to ensure validation depth, model accuracy and methodological soundness. Additional quarters were withheld to avoid speculative overreach and to preserve focus on realisable artefacts.

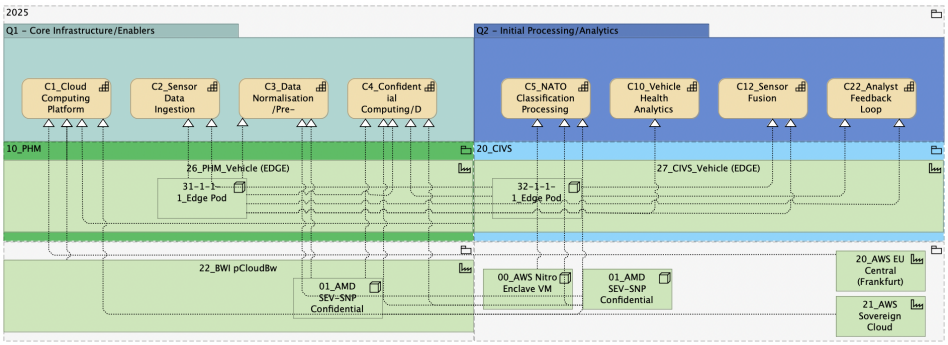


Figure 3.15: NPV-2: *OmniAware* Lines of Development and Deployment Topology (Q1-Q2)

The diagram highlights how each capability is anchored to a specific platform, ranging from general-purpose VM infrastructure to enclave-secured Kubernetes pods on AWS Nitro and Jetson-based edge hardware.

Orchestration Alternatives. Alternative orchestration models — such as enclave-enabled AWS Fargate or Intel TDX-backed container clusters — were reviewed but intentionally omitted. While Fargate lacks edge portability and sovereign trust integration, Intel TDX — despite its promising architecture — remains an emerging technology with limited ecosystem maturity, restricted attestation interoperability and insufficient validation in defence-grade deployments. As of mid-2025, robust integration with

sovereign or air-gapped environments remains technically constrained, rendering Intel TDX unsuitable for operational modelling in this context.

In Q1, foundational infrastructure such as the sovereign cloud platform (C1), sensor ingestion points (C2) and pre-processing units (C3) are instantiated across central cloud nodes and edge proxies. Confidential workloads (C4) are bound to trusted execution environments using either AMD SEV or Nitro Enclaves, depending on the deployment domain.

Capabilities introduced in Q2, including advanced analytics (C10, C12) and mission-aware human feedback loops (C22), are deployed in dedicated pods, segmented by execution constraints and mission roles. Notably, some nodes — such as the CIVS Consumer Pod — span multiple LoDs, revealing shared infrastructure dependencies. This enforces the architectural principle of composability, ensuring capabilities can be developed and tested in isolation yet deployed in integration.

To maintain clarity, the figure clusters capabilities within each LoD and links them to the underlying deployment environments. Each platform is annotated to reflect the type of nodes it supports (e.g. EC2 AMD-SEV, Kata, Jetson) and which capability instances it hosts. An important architectural distinction within the NPV-2 view lies in the separation of cloud platform environments based on their supported confidential computing capabilities. Specifically, two sets of sovereign cloud locations are modelled: the AWS Sovereign Cloud (Germany-BMI) and the AWS EU Central (Frankfurt) region on the one hand and the BWI pCloudBw on the other.

While the former are assumed to support both AMD SEV-SNP-backed virtual machines and AWS Nitro Enclaves, the latter is currently modelled with AMD SEV-SNP as the sole supported confidential computing primitive. This reflects the current deployment constraints and hardware assumptions associated with the BWI-operated pCloudBw infrastructure, which does not natively expose AWS Nitro Enclave functionality at the time of this writing.

Sovereign Trust Anchor: Choosing SEV-SNP over Nitro and TDX. The use of SEV-SNP-based virtual machines as the default confidential computing primitive reflects a deliberate choice for sovereign, interoperable trust enforcement. In contrast to Nitro Enclaves, which depend on proprietary attestation and tight AWS integration, SEV-SNP enables externally verifiable attestation chains and policy-bound confidentiality across providers. TDX-based container stacks, while promising, were excluded due to insufficient integration maturity and the absence of stable confidential orchestration tooling in sovereign deployment scenarios.

Technology Selection. The choice to model all confidential computing workloads with SEV-SNP reflects a deliberate architectural decision in favour of a more open and interoperable trust anchor. Unlike AWS Nitro Enclaves, which are tightly integrated into the proprietary AWS attestation and key release infrastructure, SEV-SNP enables verifiable attestation flows that are not confined to a single cloud provider. This universality supports the modelling of sovereign trust domains across heterogeneous platforms — including national and coalition cloud environments. Moreover, SEV-SNP's support for full virtual machine (VM) isolation with externally verifiable integrity measurements makes it a preferred choice for federated deployments in contexts where cloud-native services such as Nitro Enclaves are either unavailable or incompatible with national governance constraints. The resulting architecture aligns with the decentralised attestation logic discussed in Section 3.2, reinforcing policy-bound trust enforcement and platform

neutrality within sovereign mission contexts.

Accordingly, capabilities mapped to the BWI environment are exclusively realised through SEV-SNP backed deployments, enforcing container or VM-level attestation via AMD's Secure Processor. This differentiation ensures realistic modelling of confidential workloads and underlines the technical divergence in sovereign cloud environments across national providers. The realisation links have been manually verified and reflect the authoritative decomposition of mission functions into executable artefacts.

By consolidating mission-specific instantiations, the `NPV-2` view supports hardware-aware rollout planning, identifies shared execution targets and reveals implicit dependencies across LoDs. It complements the service-level mapping established in `NSV-1` and forms the physical backbone for further operational planning within sovereign defence deployments. To ensure semantic consistency and traceability across modelling artefacts, the following view introduces the `NAV-1` model. It documents the architectural principles, deployment primitives and system classifications embedded within the PoC and situates the realised mission functions within a structured reference architecture aligned with the principles of the NAFv4 methodology.

The Architecture Foundation Viewpoint **NAV-1** provides a meta-level representation of the architecture as a managed, traceable and semantically coherent artefact set. Within the NAFv4 modelling structure, NAV-1 plays a central role by documenting the architectural scope, artefact status, stakeholder responsibilities and versioning history of the system model. Its objective is to ensure that the architectural process adheres to a formally governed methodology and remains auditable across its entire lifecycle [74].

The NAV-1 view developed for the *OmniAware* platform reflects three essential governance artefacts defined in the official NATO Modelling Guide: the **A2: Architecture Products** catalogue, the **A5: Architecture Status** tracker and the **A6: Architecture Versioning** scheme. Together, they form the foundational registry for all architectural viewpoints and provide a single source of truth for methodological consistency and stakeholder alignment.

The A2 artefact register includes all formally modelled views produced throughout the architectural process — including, but not limited to, capability mappings (e.g. `NCV-2`), logical service interactions (`NSV-5`), physical resource deployments (`NPV-2`) and confidential computing integration models (`NSV-1`). Each view is assigned a unique identifier, associated viewpoint class, responsible domain (e.g. technical, operational, security) and publication status. These attributes are mapped directly within the modelling environment using structured folders and tagged metadata.

To support lifecycle traceability, the **A5: Architecture Status** mechanism captures the maturity level and validation state of each artefact. Views developed in iterative cycles are annotated as *Draft*, *Validated* or *In Review*, depending on their degree of internal approval and external feedback (e.g. from internal Capgemini team, governance or AWS solution architects). This enables stakeholders to distinguish experimental modelling artefacts from production-grade architecture inputs used in the AWS FTR or the FRK.

Version control is formalised using the **A6: Architecture Versioning** construct, whereby each artefact is assigned a semantic version identifier (e.g. `v0.9-draft`, `v1.0-final`), along with associated change logs and governance notes. This structure ensures full transparency in the evolution of architectural models, enabling reproducibility, rollbacks and documentation of rationale. Version tracking is managed using model annotations and embedded documentation, aligned with the principles of model-driven governance outlined by NAFv4.

The NAV-1 viewpoint therefore constitutes the architectural backbone for traceability, reproducibility and methodological governance. It ensures that all architectural

products are consistently catalogued, assigned clear responsibilities and subject to continuous refinement based on stakeholder input. As such, it not only provides an internal quality assurance mechanism, but also aligns the architecture with NATO's formal requirements for multi-stakeholder and multi-domain interoperability.

Figure 3.16 illustrates the NAV-1 view for the *OmniAware* platform. The model serves as a structured consolidation of all architectural artefacts, standards and validation procedures referenced throughout this thesis. It formalises their integration into a traceable compliance baseline and connects each element to its underlying methodology, regulatory context or review evidence.

The view is divided into five logical sections (**A-E**), aligned with NATO Architecture Framework Version 4 (NAFv4) best practices and ArchiMate modelling conventions. Each section contributes to the validation and traceability of the architectural baseline.

Section A (Artefacts) enumerates all core deliverables developed during the project's first two quarters. These include model views such as NCV-2, NSV-1 and NPV-2, but also narrative artefacts like the PR/FAQ, the Field Ready Kit (FRK) and the consolidated remediation plan with included high-risk items. Notably, the section differentiates between implementation artefacts (e.g. IaC), project planning outputs and assessments such as the FTR and WAFR results. Importantly, artefacts such as Versioning and Status anchor deliverables within the SCA-aligned milestone structure.

Section B (Standards and Frameworks) provides an overview of all regulatory and methodological sources that constrain or guide the system's design. These include both strategic standards (NAFv4, UAF-DMM, NATO ArchiMate Guide) and technical compliance requirements such as ISO/IEC 27001, BSI C5, AWS TSE-SE and STANAG 4774/4778. All elements are represented as Constraints or Principles, depending on whether they mandate specific properties or provide architectural guidance. By clustering them semantically, the model allows for cross-referencing artefacts in Section A with their compliance roots.

Section C (Methodology) contains the methodological blueprint underpinning the system model. Key references include the ArchiMate Modelling Guide for NAFv4, Capability-driven Design and the adoption of Model-driven Engineering. The modelling rationale for traceability (NCV $\rightarrow$ NSV), lifecycle alignment with the SCA and viewpoint decomposition logic is anchored here. Each element is structured as a Principle.

Section D (Architecture Compliance) documents the validation state of the current architecture. Key assessments include the general Compliance Status, an explicit Validation Record: AWS FTR and the most recent Review Date. Version control is formalised through a concrete Model Version v1.0, while residual issues such as Open Risk: Data Sovereignty are documented as Constraints. This section can be extended over time to reflect future model iterations, including Q2 or Q3 validation events.

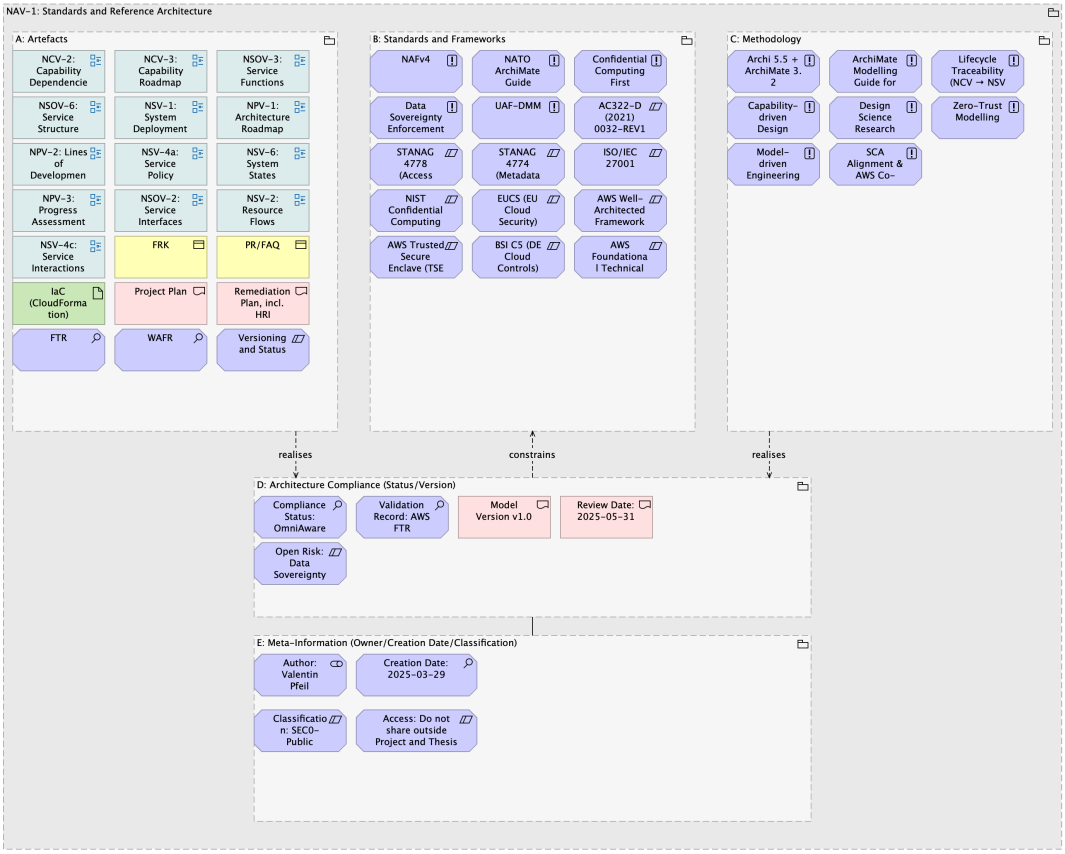


Figure 3.16: NAV-1: *OmniAware* Standards and Reference Architecture

Section E (Meta-Information) provides metadata for traceability. This includes the document owner, creation date and classification level. The access notice restricts external sharing to the thesis context and designated project stakeholders.

Overall, this view ensures that all design decisions, artefacts and validations are grounded in traceable and structured architectural evidence. It complements previous capability- and deployment-centric models by offering a compliance-centric lens through which the architecture's rigour, maturity and alignment with formal standards can be assessed.

The `NAV-1` and its placement within this chapter — which focuses on the deployment and interaction of cloud, fog and edge computing layers — reflects the necessity of aligning runtime system representations with a traceable architectural foundation. Cloud-native defence systems require not only performance and interoperability but also verifiable alignment with standards, secure lifecycle governance and traceable modelling methods. These aspects are encoded within the `NAV-1` structure.

Tool Selection. The choice of using `Archi` and `draw.io` over formal toolchains such as `Sparx EA` or `OpenSCAP`-based compliance modelling was based on accessibility, reproducibility and architectural transparency. While these platforms offer extended automation or policy-mapping features, they often require proprietary licensing or assume specific runtime infrastructures. The selected tools allow for lightweight, traceable and version-controlled modelling aligned with project-specific governance constraints.

Specifically, the `NAV-1` view aggregates and contextualises core design principles (e.g. `Capability-driven Design`, `Zero-Trust Modelling`), modelling artefacts (e.g. `NPV-2`:

Lines of Development) and validation mechanisms (e.g. AWS FTR, WAFR) into a structured compliance framework. It thereby documents the foundational rules under which the physical deployment layers in NPV and NSV views are defined and validated.

Furthermore, the inclusion of artefacts such as the PR/FAQ, IaC CloudFormation and the structured application of AWS's TSE assessment in NAV-1 supports a unified audit trail between strategic principles and executable deployments. It bridges the gap between model-level governance and system-level instantiations in a federated deployment environment.

While the views presented so far — NSV-1, NPV-1/2 and NAV-1 — primarily address deployment, compliance and structural governance of the architecture, the subsequent section introduces a shift in focus towards the security-critical execution logic underpinning the *OmniAware* platform.

Granularity and Visual Abstraction. To maintain methodological rigour and prevent overmodelling, all reference diagrams — particularly within NSV-1 and NPV-2 — follow a black-box abstraction principle. While more granular internal states (e.g. kernel versions, enclave hash trees, node-level firewall rules) could have been modelled, these were intentionally abstracted to preserve generalisability and ensure focus on mission-aligned architecture components.

The architectural structures introduced in this chapter do not aim at exhaustiveness, but at rigorous, standards-aligned abstraction. By combining NAFv4-compliant viewpoints with validated secure-by-design principles, the architecture forms a robust methodological base for secure, mission-centric system realisation. The following chapter evaluates how this architecture manifests in implementation.

3.3 CONFIDENTIAL COMPUTING

Summary: This section focused on the integration of confidential computing to enforce data confidentiality across heterogeneous environments. The chapter elaborated on architectural strategies involving TEEs, the application of runtime attestation mechanisms and the deployment of enclave-based service execution using technologies such as AMD SEV-SNP and AWS Nitro Enclaves. Furthermore, the decision to adopt *Kata Confidential Runtime* was substantiated through comparative analysis against alternatives, highlighting its alignment with the Vault-based policy enforcement framework. The resulting blueprint ensures cryptographic assurance and compliance without compromising operational agility.

The increasing reliance on distributed and federated computing environments across military and intelligence domains has intensified the need to protect sensitive data not only at rest and in transit, but also during active processing. *confidential computing* addresses this requirement by enabling secure processing of *data-in-use* through the application of hardware-enforced TEE technologies.

Unlike conventional protection mechanisms such as encryption at rest or in transit, which merely safeguard data stored or transported across systems, confidential computing provides runtime protection against threats originating from compromised system software, privileged insiders or co-tenants in multi-tenant environments [33]. This is particularly relevant in defence-grade environments where coalition operations, sovereign control and cross-domain execution increase the attack surface.

According to the Confidential Computing Consortium (CCC) [70], confidential computing is defined as:

> *The protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment (TEE).*

This thesis adopts this definition and applies it specifically to defence cloud and edge architectures, where mission-critical data must remain protected across the full data lifecycle — including real-time processing, in-situ analytics and dynamic decision support. While confidentiality of *data-at-rest* and *data-in-transit* is assumed to be enforced via conventional encryption and network-level safeguards (e.g. TLS, VPN, KMS), the unique contribution of confidential computing lies in its ability to protect data during active computation on potentially untrusted hosts.

3.3.1 *Security Architecture and Attestation Workflow*

The *OmniAware* platform places a particular emphasis on *data-in-use* confidentiality due to the high sensitivity of telemetry, sensor fusion and imagery data. These data types not only reflect operational states and mission intentions, but also inform tactical decisions in near-real-time. Consequently, any unauthorised inference or compromise during runtime could lead to mission disruption or strategic disadvantage.

Figure 3.17 visualises the typical data lifecycle in military systems and situates confidential computing within the broader security context.

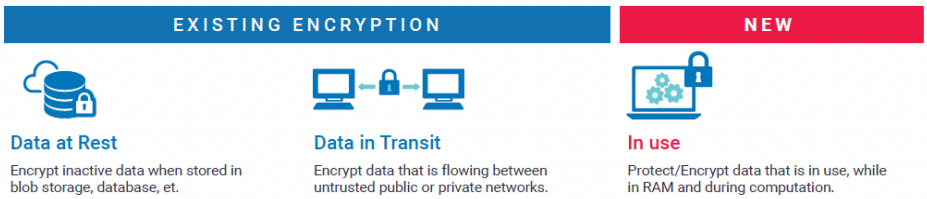


Figure 3.17: Data Lifecycle and Protection Domains: Confidential computing secures data during active processing (*in use*) and complements existing mechanisms for *at rest* and *in transit* protection [35].

In summary, confidential computing forms a foundational building block of secure mission computing by enabling verifiable isolation, cryptographic attestation and policy-enforced key release — even under degraded trust assumptions. Its adoption in the *OmniAware* architecture supports the strategic objectives of information sovereignty, zero-trust execution and federated collaboration across defence ecosystems.

Beyond its technical capabilities, confidential computing also aligns with key security and compliance frameworks applicable to defence-grade infrastructures. These include the NATO Cloud Security Instruction *AC/322-D(2021)0032-REV1-U* [20], which mandates enclave-based protection, attestation and workload isolation for mission-sensitive data. National standards, such as the German BSI C5 catalogue [11], provide control baselines for regulated cloud operations and map directly to ISO/IEC 27001:2022 through structured compliance tables [24]. Complementing these, the AWS Trusted Secure Enclaves Sensitive Edition (TSE-SE) guidance [81] defines provider-side implementation best practices for confidential workloads. Together, these frameworks reinforce the strategic relevance of confidential computing in sovereign and coalition-based defence environments.

This foundational understanding of confidential computing sets the stage for a more detailed discussion of its architectural implications, particularly with respect to

security requirements, standardisation frameworks and Trusted Execution Environment integration strategies.

Confidential computing plays a critical role within secure system architectures by addressing foundational security requirements in defence-grade cloud and edge environments. These requirements include *data sovereignty*, *isolation*, *compliance assurance*, *attestation* and secure *key management*. Each of these dimensions is essential to enable mission-critical data processing under adversarial or coalition-based conditions.

To address these challenges, confidential computing integrates with established security frameworks, such as those outlined by NIST, ISO/IEC 27001 and the NATO Architecture Framework [53], [74]. These standards provide baseline controls for cryptographic operations, trust establishment, policy enforcement and lifecycle management of sensitive workloads.

At the architectural level, several principles have emerged as indispensable for modern military IT systems. These include:

- **Zero Trust**: assumes no implicit trust — neither in networks nor in identities — and requires continuous verification at all access boundaries;
- **Security-by-Design**: integrates security as a core component from the earliest stages of system development;
- **IaC** (Infrastructure as Code): ensures deterministic, reproducible and auditable deployments by codifying infrastructure artefacts;
- **Automated Compliance Enforcement**: enables continuous validation of system states against pre-defined policies using automated controls and tooling.

Confidential computing aligns strongly with these principles. By enforcing *hardware-based runtime isolation*, it enables the practical realisation of Zero Trust concepts at the compute layer. Furthermore, the use of *remote attestation* mechanisms strengthens compliance auditing and supports policy-based workload orchestration, even across untrusted or federated domains.

TEEs form the technological anchor of this paradigm. They establish an isolated, attested enclave within the processor, allowing sensitive computations to occur without exposure to the host operating system or hypervisor. This guarantees that mission-critical processes remain verifiable and confidential — a prerequisite for coalition operations, cross-domain data exchange and secure multi-tenancy in sovereign defence cloud platforms.

In addition to its architectural contributions, confidential computing supports compliance with strategic security frameworks and cloud assurance catalogues relevant to defence operations. One such directive is the NATO Security Instruction *AC/322-D(2021)0032-REV1-U*, informally referred to as "NATO D32", which mandates enclave-based isolation, remote attestation and workload separation for handling classified mission data in coalition environments [20]. This instruction outlines both technical and procedural requirements for mission data processing, including constraints on identity federation, cryptographic boundaries and audit traceability.

At the national level, the German Federal Office for Information Security (BSI) provides the *Cloud Computing Compliance Criteria Catalogue (C5)*, which defines a control baseline for the operation of cloud workloads under regulatory oversight [11]. The 2022 extension of this standard further aligns with ISO/IEC 27001 by providing a reference mapping of C5 controls to information security management requirements [24].

Confidential computing aligns with C5 across key areas such as access control, data isolation, cryptographic operations and audit logging, especially when operated within Trusted Execution Environments.

Cloud-native frameworks such as the *AWS Trusted Secure Enclaves - Sensitive Edition (TSE-SE)* specification [81] complement these efforts by defining provider-specific security profiles for enclave-based workloads. The TSE-SE model emphasises runtime attestation, dynamic key release and enforcement of immutable infrastructure definitions, thereby ensuring that confidential workloads remain verifiably isolated throughout their lifecycle.

These external frameworks reinforce the applicability of confidential computing to sovereign and coalition-aligned defence deployments. Their shared emphasis on attestation, cryptographic trust roots and lifecycle assurance directly map to the capabilities enabled through confidential computing and validate its architectural integration within the *OmniAware* platform.

Trusted Execution Environments (TEEs) form the technological foundation of confidential computing by establishing secure enclaves within a processor or platform that guarantee the confidentiality and integrity of data-in-use. They offer isolated execution contexts that are resistant to attacks from privileged system software, hypervisors and co-located tenants [70]. In the defence context, TEEs are particularly valuable due to the increasing reliance on distributed computing infrastructures and the requirement to enforce policy-compliant data protection — even under coalition or contested trust conditions.

This thesis focuses on three representative TEE implementations, each designed for a distinct deployment domain:

- **AMD SEV-SNP:** Designed for virtualised cloud infrastructures, SEV-SNP encrypts memory pages and ensures runtime state integrity through Secure Nested Paging and the integration of a Memory Integrity Tree. It extends AMD's Secure Encrypted Virtualisation by adding support for attestation, restricted hypervisor access and secure VM state handling [32].

- **AWS Nitro Enclaves:** A cloud-native TEE model embedded within Amazon EC2 instances, Nitro Enclaves provide hardware-isolated execution by creating a secure enclave without persistent storage or network access. Data exchange occurs via vsock interfaces and workload attestation is supported via AWS KMS integration and a Nitro Attestation Document [81].

- **Jetson OP-TEE:** Based on the ARM TrustZone architecture, OP-TEE is tailored for embedded and edge deployments. It supports secure boot, cryptographic services and isolated execution of Trusted Applications (TAs) in the Secure World, separated from the Rich OS. Its lightweight footprint makes it suitable for tactical devices in constrained environments such as CIVS or PHM edge nodes.

From a design perspective, the technical differentiation among these TEEs can be summarised along six dimensions: deployment scope, isolation method, attestation capability, trusted root, code and data confidentiality and integration overhead. Table 3.1 provides a comparative overview:

The suitability of each TEE type for military deployments depends on several factors: sensitivity of data, required level of decentralisation and compliance requirements. SEV-SNP excels in cloud or fog infrastructures with virtualised workloads that require isolation from the hypervisor. Nitro Enclaves, being tightly integrated with AWS services,

Table 3.1: Comparison of Selected Trusted Execution Environments [42], [61], [81]

Feature	AMD SEV-SNP	AWS Nitro Enclaves	Jetson OP-TEE
Deployment Scope	Cloud Virtualisation, Cloud-native	Embedded	Tactical Edge
Isolation Method	Encrypted VM Memory, Integrity Tree	Dedicated vCPU, vsock-only IPC	ARM TrustZone World Partitioning
Remote Attestation	Supported (SEV certificate chain)	Nitro Attestation Document via KMS	Custom TA-based Attestation
Trusted Root	AMD Platform Security Processor	Nitro Hypervisor	TPM, ARM SoC fuses / BootROM
Code + Data Confidentiality	Strong	Strong	Medium-High
Integration Overhead	Medium (Hypervisor-dependent)	Low (Managed EC2 APIs)	Low-Medium (requires TA development)

offer a managed and scalable enclave option for mission-critical IaaS platforms with strong attestation and zero external access. OP-TEE, in contrast, enables lightweight confidentiality at the tactical edge and supports sovereign hardware deployments in environments where cloud connectivity is intermittent or restricted.

For the *OmniAware* platform, all three TEEs are architecturally relevant. SEV-SNP is leveraged in sovereign cloud workloads where hypervisor trust cannot be assumed. AWS Nitro Enclaves provide a flexible runtime for telemetry processing and redaction pipelines with managed attestation capabilities. Jetson OP-TEE secures field-level inference tasks on NVIDIA-based edge devices used in PHM and CIVS contexts.

Design decisions regarding the integration of TEEs into the *OmniAware* architecture were guided by three principal requirements:

- **Layered Trust Enforcement:** Using `NSV-6` state transitions, the architecture implements remote attestation sequences during workload initialisation (e.g. via SPIFFE/SPIRE). These ensure that only verified workloads receive cryptographic material (e.g. via `Vault` or KMS).
- **Policy Mapping:** In `NSV-4a`, service policies define enclave placement rules based on classification, domain ownership and encryption strength. For instance, C5-classified workloads must run on attested hardware with external visibility disabled.
- **Assurance Monitoring:** In `NPV-3`, enclave instantiation events, attestation responses and cryptographic operations are logged to a provenance engine that validates compliance with NATO D32 fallback assumptions [20].

As highlighted in NATO D32 (*AC/322-D(2021)0032-REV1-U*), Security Enforcing Services (SES) such as TEEs must provide robustness, fallback behaviour and verifiable enforcement of constraints such as enclave sealing, boot integrity and identity scope isolation. For example, the SES design rules SC-AIS-05-04-NR to -07-NR define expectations around enclave degradation resistance, rollback recovery and environmental assumptions [20].

The *OmniAware* architecture accommodates these SES-level constraints by enforcing key lifecycle boundaries and fallback protocols across all supported TEEs. Should an enclave fail attestation or enter a degraded state, cryptographic key material is withheld and downstream dataflows are automatically blocked via sovereign policy enforcement (SF7) and runtime monitors.

This tightly integrated approach ensures that all workloads — whether cloud-based, fog-deployed or edge-embedded — execute within a coherent, verifiably trusted

runtime that satisfies both operational and regulatory expectations across NATO, BSI and hyperscaler compliance frameworks.

While TEEs represent the central enforcement mechanism for *data-in-use* protection, the overall security architecture of the *OmniAware* platform is supported by a set of complementary measures that ensure compliance, observability and operational control across all lifecycle stages. These mechanisms are aligned with the security objectives defined in NATO D32 [20], BSI C5 [11] and hyperscaler-specific best practices [81].

Key Lifecycle and Rotation Strategy To uphold cryptographic hygiene and prevent long-lived key misuse, all data encryption keys, certificate chains and signing secrets are subject to mandatory lifecycle policies. The platform integrates AWS Cloud Hardware Security Module (CloudHSM) and multi-account KMS key segregation. Key rotation intervals, access control and key material destruction are governed by sovereign policy engines and logged in a centralised audit trail as required by NPV-3.

Ingress Protection and TLS Enforcement All external interfaces are secured through a layered perimeter defence architecture including API Gateway (GW), network firewalls, Web Application Firewall rules and optional integration with AWS Shield Advanced. Mutual TLS (mTLS) is enforced for service-to-service communication, with root-of-trust anchored in AWS Certificate Manager (ACM)-managed certificates. API schemas and payload validation routines are integrated into zero-trust boundaries.

Runtime Threat Detection and Vulnerability Monitoring Continuous security posture validation is achieved via runtime telemetry from Amazon Inspector, Security Hub and GuardDuty. These tools scan for vulnerability exposure (e.g. in container images or Lambda deployments), monitor abnormal execution patterns and propagate findings into the NSV-6 state machine, allowing for policy-bound remediation or enclave revocation.

Auditability and Forensic Traceability All critical security operations — such as key access, attestation results, trust policy violations and classification transitions — are logged via CloudTrail and exported to a dedicated *audit account*. This log stream is standardised for Security Information and Event Management (SIEM) integration and supports evidence preservation in the context of NATO forensics and compliance procedures.

Data-at-Rest Encryption and Labelling In addition to enclave memory encryption, persistent data is secured using Server-Side Encryption with Key Management Service SSE-KMS, classification-bound access policies and object-level metadata labelling (e.g. NATO-restricted, public, redacted). Data lake partitions, imagery buckets and structured datasets are bound to STANAG 4774/4778 policies and checked at ingestion and access time.

Cross-Domain Role Isolation and Least Privilege Design System accounts are split into dedicated security domains (e.g. ingest, datalake, audit, output) to support workload isolation and compliance with NATO cross-domain execution policies. Role-based access is enforced via least-privilege identities, scoped session permissions and dynamic role assumption mechanisms (e.g. via OIDC tokens or federation brokers).

Zero Trust API Control and Federated Identity The platform integrates a multi-level API GW for enforcing zero-trust control across all ingress and internal interfaces. Identity validation is performed using a federated trust broker compatible with SAML/OIDC, ensuring authenticated and classified role propagation across national domains. All data flow interactions are evaluated against sovereign rules via a policy enforcement engine (SF7), which applies runtime decisions and logs provenance in SF6.

Secure Metadata and Provenance Infrastructure To guarantee data integrity and traceability, all mission-critical metadata (e.g. classification labels, data source, encryption state, processing history) is tracked using a distributed metadata management system. These metadata records feed the platform's audit backbone and enable cross-domain policy enforcement, redaction tracking and classification processing — particularly within the CIVS and PHM service chains.

Taken together, these additional security controls reinforce the platform's TEE-centric foundation and demonstrate the maturity and mission-readiness of the *OmniAware* architecture. While they are not all elaborated individually in the following sections, their presence underpins key capabilities for compliance, resilience and zero-trust enforcement.

The architectural embodiment of these controls is formalised across selected NAFv4 views — most notably NSV-4a (Service Policy), NSV-6 (System States), NPV-3 (Progress Assessment) and NSOV-3 (Security Services). Where appropriate, representative examples are highlighted to illustrate foundational design principles. However, the primary focus of the following section remains on the Trusted Execution Environment (TEE) strategy as the core enforcement model for secure and sovereign workload execution.

3.3.2 *Trusted Execution Environment Implementation*

The integration of Trusted Execution Environments (TEEs) within the *OmniAware* platform adheres to a design methodology aligned with selected viewpoints of the NAFv4. Central to this approach are the architecture views `NSV-4a` (Service Policy), `NSV-6` (System States) and `NPV-3` (Progress Assessment), each capturing a specific facet of the system's security architecture, trust lifecycle and compliance instrumentation.

The **NSV-4a (Service Policy)** encapsulates the rules governing enclave placement, workload trust domains and classification boundaries. In accordance with NATO modelling guidelines [74], all mission-sensitive workloads bearing NATO-restricted classification must execute within TEE-secured environments. The model formalises constraints such as:

- *No execution without successful remote attestation.*
- *Key release only if enclave state and measurement hash are verified.*

These policy rules are enforced through sovereign orchestration agents and policy-bound scheduling engines that govern workload deployment across cloud and fog infrastructures.

Given the extensive service portfolio of the *OmniAware* platform, this model deliberately narrows its focus to a critical architectural excerpt: the **PHM Landing Zone** and its policy enforcement mechanisms for classified telemetry. A *Landing Zone* (LZ) refers

to a pre-configured, secure and governed cloud environment that enables compliant deployment of mission workloads. Typically, it includes multi-account structures, centralised audit logging, identity federation, encrypted networking and policy enforcement mechanisms. In the context of AWS, an LZ represents the baseline governance perimeter for infrastructure-as-code rollouts and zero-trust control domains [30].

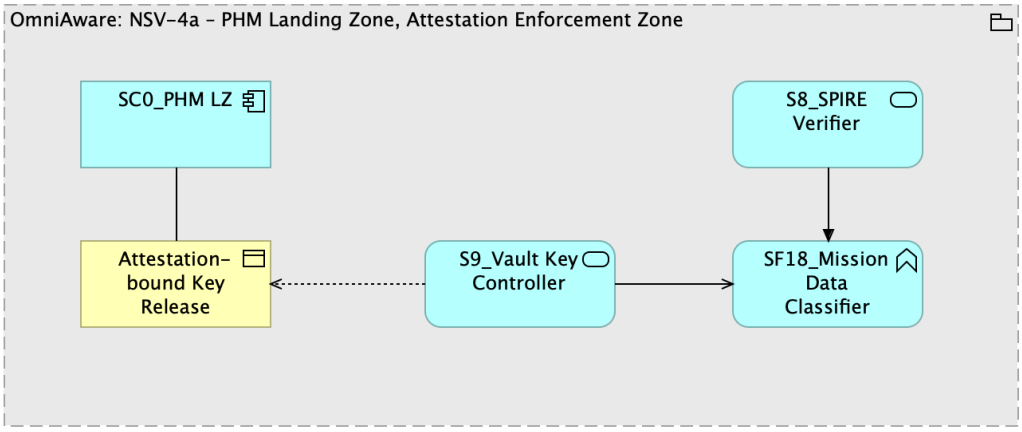


Figure 3.18: NSV-4a - PHM LZ Policy Enforcement

As shown in Figure 3.18, the PHM LZ acts as the sovereign ingress interface for telemetry that has passed upstream validation. At this stage, service-level policies enforce classification-aware workload gating, cryptographic key protection and authorisation. These controls are rooted in Confidential Computing principles and implemented via a trust pipeline built on remote attestation and policy-bound key management.

Incoming workloads are executed as TEE-protected virtual machines on fog infrastructure supporting SEV-SNP. Prior to decryption or processing, the system initiates a full remote attestation via SPIRE, verifying enclave measurement, identity and integrity.

Upon successful verification, HashiCorp `Vault` releases decryption keys only to those workloads whose attested identity matches a mission-bound allowlist. This ensures that only authorised and cryptographically validated services may access or persist sensitive telemetry.

The key model elements in this architecture include:

- **Application Component:** `PHM LZ`, responsible for secure intake and contextual decryption of mission telemetry.
- **Application Function:** `Mission Data Classifier`, enforcing classification- and tenant-aware routing.
- **Policy Element:** `Attestation-bound Key Release Policy`, ensuring enclave trust is validated before any key material is issued.
- **Supporting Services:** `SPIRE Verifier` and `Vault Key Controller`, together enforcing policy-driven runtime validation.

The selection of the PHM LZ as an illustrative example is grounded in its architectural role: it marks the point at which encrypted mission data intersects with policy enforcement and classification compliance. Moreover, it demonstrates how enclave-based services can be integrated with remote attestation workflows to realise sovereign, zero-trust access control.

While the view captures only a minimal slice of the overall platform, it effectively illustrates the application of NAFv4-conformant policy enforcement—spanning identity-based decryption control, attestation-gated key release and classification-aware workload orchestration. Further NSV-4a models (e.g. for downstream analytics or audit log protection) could be constructed in similar fashion but are excluded due to scope limitations.

This architectural configuration exemplifies a key assurance strategy for next-generation mission systems: trust must not be statically declared but dynamically derived from runtime identity, enclave measurement and policy context. By implementing these verifications at the point of data landing, *OmniAware* enforces both sovereign policy intent and compliance with NATO data protection directives [42], [43].

To capture the dynamic trust lifecycle of secure workloads, the NSV-6 view formalises the system state transitions that govern enclave execution, integrity validation and policy-compliant key usage. Specifically, this model describes the runtime phases a TEE-protected workload must traverse, enforcing cryptographic controls and telemetry-bound verification checks within the PHM LZ (cf. `NSV-4a`). These transitions are critical to ensuring that decryption and execution only occur within verified enclave configurations.

Figure 3.19 illustrates the `PHM LZ System State Transition Pipeline` for trusted enclave-based telemetry workloads. The state model describes how secure workloads are instantiated, attested and executed on SEV-SNP-enabled fog infrastructure.

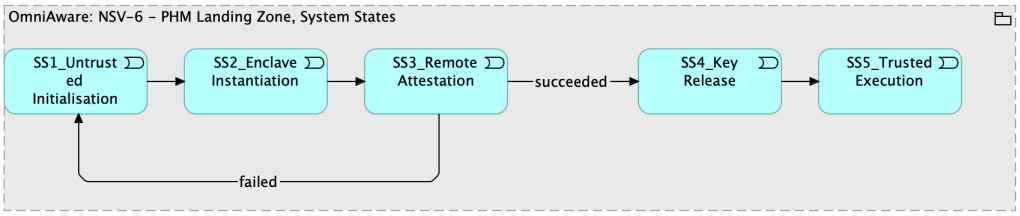


Figure 3.19: NSV-6 - PHM LZ System State Lifecycle

The illustrated model defines the following state transitions:

- `Untrusted Initialisation`: The workload is launched in an unverifiable state, encompassing bootloader, hypervisor and OS setup.
- `Enclave Instantiation`: The enclave context is provisioned with measured configuration parameters (virtual Secure Processing Layer (vSPL), memory layout, policy fingerprint).
- `Remote Attestation`: The enclave submits a cryptographic attestation via SPIFFE/SPIRE, verifying code integrity, configuration and identity.
- `Key Release and Policy Enforcement`: The attestation is evaluated against a mission-bound allowlist. Only upon successful validation does the `Vault Key Controller` issue the decryption keys.
- `Trusted Execution and Audit Logging`: If validation is successful, the workload decrypts mission telemetry and transitions into a verified enclave state. Audit trails are continuously exported to immutable storage.

The diagram also incorporates a conditional feedback loop to represent policy-based rejection: should attestation fail (e.g. misconfigured state or unknown fingerprint), the workload reverts to `SS1_Untrusted Initialisation` and no cryptographic material is released. This fallback pathway reflects NATO D32-mandated attestation policies [43].

Although the modelling environment does not natively support System State constructs, this thesis adopts the `Application Event` element as a semantic workaround to represent system states. This modelling choice preserves the logical integrity of state transitions while maintaining consistency with the NAFv4 meta-model implementation.

The use of NSV-6 for modelling the PHM LZ lifecycle tightly integrates with the enforcement logic presented in the NSV-4a service policy. Together, they establish runtime governance for all telemetry workloads and provide the operational anchor point for subsequent validation logic, as discussed in the `NPV-3` view.

The `NPV-3` view models the telemetry-driven compliance assessment infrastructure responsible for monitoring the operational integrity of TEE-protected workloads across the *OmniAware* platform. It acts as a continuous evidence stream for forensic inspection, runtime validation and regulatory attestation. The view is tightly integrated with the enclave lifecycle defined in `NSV-6`, ensuring that each system state transition — such as *remote attestation*, *key release* and *trusted execution* — is observable, verifiable and policy-evaluable. By formally integrating the audit control logic within the NPV-3 view, this model operationalises the concept of Progress Assessment beyond static compliance reports, embedding enforcement policy directly into telemetry runtime transitions.

Key metrics are extracted via telemetry services and encompass:

- **Enclave Measurement State:** Captures integrity measurements and configuration metadata during runtime.
- **Attestation Events:** Logs evidence and response chains for VCEK- or Nitro-based remote attestation.
- **Key Access Events:** Records access and issuance of secrets, gated by policy evaluation.
- **Policy Deviations:** Detects and flags deviations such as enclave reinitialisation, expired attestations or denied key releases.

All collected events are committed to a tamper-evident, immutable audit log, enabling compliance verification against frameworks such as BSI `C5` and ISO/IEC-27001. These logs are structured and indexed for post-mission review, automated alerting and cross-domain traceability, supporting both sovereign and coalition-level reporting requirements.

The audit pipeline is anchored within the **PHM Audit Account**, a dedicated multi-account configuration with separate telemetry ingestion, metadata tagging and compliance logging services. To ensure workload-level provenance, metadata — such as mission ID, classification scope and node identity — is enriched via runtime attributes and validated through signature verification. Where applicable, Zero Trust policy engines are used to trigger alerts, isolate enclaves or revoke credentials upon policy violation detection.

By formalising operational oversight as an integral part of the deployment architecture, `NPV-3` ensures continuous situational awareness and mission compliance — not only during but also after enclave execution. This aligns with directives defined in

NATO *AC322D(2021)0032-REV1* and supports proactive threat detection, auditability and forensic accountability across cloud, fog and tactical infrastructure layers.

The NPV-3 view captures how this telemetry is evaluated within an audit and policy enforcement framework. Each state transition (e.g. `Attestation → Key Release`) is tied to an attestation verdict, policy check and cryptographic validation result. Deviations from defined workload policies (e.g. enclave mismatch, timing irregularities, geographic dislocation) trigger audit events and telemetry quarantine.

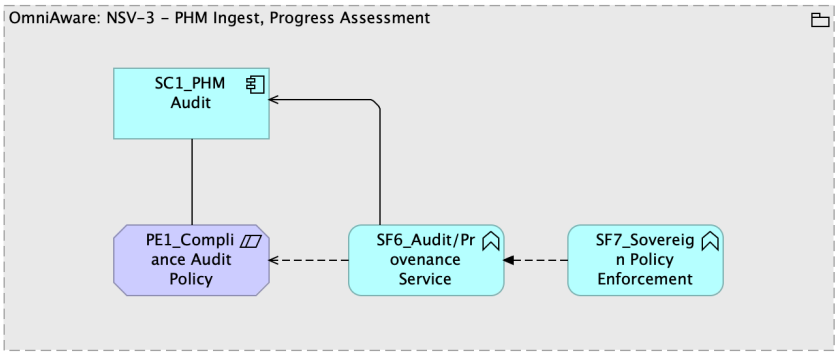


Figure 3.20: NPV-3 — PHM Landing Zone, Progress Assessment

Figure 3.20 depicts the compliance audit service implemented within the PHM landing zone. This architectural view illustrates how compliance logic is enforced at the telemetry persistence boundary using runtime logging and attestation-bound provenance extraction. The central `Audit/Provenance Service` (`SF6`) orchestrates log generation, metadata tagging and event forwarding based on input from policy evaluation layers.

The `Sovereign Policy Enforcement` function (`SF7`) provides attestation-gated triggers that determine which events must be captured and evaluated. These two functions are semantically bound to the `PE1_Compliance Audit Policy`, modelled as a `Constraint`, which governs classification-aware forensic inspection. The Compliance Audit Policy defines the enforcement criteria used to validate attestation events, key releases and telemetry anomalies. The resulting data is persisted within the `SC1_PHM_Audit` component and constrained by said policy.

Due to modelling environment constraints, this model leverages a `Constraint` object in place of a dedicated `Policy Element`, as the ArchiMate language does not define native first-class elements for dynamic runtime enforcement logic. Nevertheless, this construct semantically captures the enforcement logic derived from mission-specific audit constraints and zero-trust telemetry validation.

Each attestation-gated transition — introduced in NSV-6 — is monitored for policy compliance and any deviation triggers enforcement actions defined in the associated `PE1_Compliance Audit Policy`. This enables a real-time *progress evaluation* of trusted workloads, transforming audit services into an integral component of runtime policy assurance and sovereign mission telemetry governance.

Cross-Cutting Controls: Remote Attestation and Key Enforcement
Remote attestation is mandatory for all workloads processing classified mission data. It is anchored in hardware-level identity (e.g. SEV-SNP VCEK, Nitro Attestation Document) and validated against a sovereign or cloud-native policy engine. Secrets (e.g. TLS keys, access tokens, inference weights) are withheld until the attestation succeeds. Sovereign deployments use `HashiCorp Vault` with policy-tied allowlists, while hyperscaler environments integrate `AWS KMS` to restrict access to validated workloads only.

Interoperability and Portability of Trust
In heterogeneous mission environments spanning sovereign cloud, fog and edge, workload portability is essential. This requires consistent remote attestation workflows, harmonised identity frameworks and trust decoupling across TEE and CPU architectures. When enclave migration is not feasible, Zero Trust proxies are introduced to preserve verification boundaries and enforce key release conditions — all of which are reflected in the NPV-3 provenance layer.

Platform Policy: Mandatory Use of SEV-SNP for High-Assurance Workloads
OmniAware mandates that all classified workloads operate exclusively on SEV-SNP-enabled hosts to unify key lifecycle enforcement. Each state transition is cryptographically anchored using nested paging, platform-diffie-Hellman key binding and verified VCEK signatures. This ensures consistent runtime posture across deployments, reduces policy fragmentation and simplifies trust enforcement even across federated environments.

- Memory encryption and integrity via Nested Paging.
- Secure VM launch using Platform Diffie-Hellman Key (PDHK) exchange.
- Cryptographic remote attestation reporting using VCEK and CEK identity chains.

Implication for Compliance and Auditability
Audit logs tied to state transitions are stored immutably and exported to a mission-bound audit controller. These logs support continuous compliance validation against regulatory baselines such as BSI C5 and ISO/IEC 27001 and are referenced in post-mission forensic reviews. In edge scenarios lacking full enclave support, deviations (e.g. fallback to telemetry-only mode) require risk acceptance and attestation override authorisation.

From a modelling perspective, this view links `NSV-6` state telemetry with compliance enforcement and post-execution validation. It provides the structural backbone for runtime monitoring, cryptographic auditability and trust propagation across sovereign and coalition-controlled execution zones.

In summary, confidential computing constitutes a foundational design pillar of the *OmniAware* architecture. Through the structured implementation of enclave-based trust workflows, remote attestation pipelines and policy-bound key enforcement, this chapter has demonstrated how mission workloads can be securely orchestrated, auditable and sovereign — even across untrusted or coalition environments. The following sections build upon this trusted computing layer to address further system concerns such as telemetry portability, policy orchestration and cross-domain workload integration.

3.4 INTERFACES

Summary: The final section of Chapter 3 introduced the design of core interfaces underpinning interoperability within the PHM deployment context. While the implementation remains limited to selected endpoints, the model captures key interface roles between data producers, processing components and consumer services. Exemplary interfaces include telemetry ingestion, secure secret exchange and attestation token validation. The design considerations were guided by the principles of modularity, secure data flow and future extensibility toward multi-domain integration. This section concludes the

architectural composition by ensuring interface-level connectivity across the previously introduced components. In modern mission platforms such as PHM, interfaces are not mere integration artefacts, but serve as critical enforcement points for classification, attestation and zero-trust control. Each interaction between workloads, services and trust-enforcing components must be explicitly modelled, as every interface may constitute a potential attack surface. To illustrate the intersection of system design and policy enforcement within defence-grade architectures, this thesis models a *single security-relevant interface* as a representative example. Further interfaces may be added depending on available resources but are not part of the architectural core.

In alignment with the NAFv4, this section introduces three formally defined viewpoints:

- **NSOV-2** — Describes the intended service behaviour and semantic interface characteristics.
- **NSV-2** — Identifies the interface within the system architecture and its associated service functions.
- **NSV-4c** — Specifies interface-related policy constraints andre additional technical enforced security measures.

The *PHM Ingest Gateway* represents a secured external service interface from vehicles or edge devices responsible for receiving and validating classified telemetry data. As the system's ingress point for mission-critical inputs, it plays a central role in enforcing the platform's zero-trust architecture.

- **Interface Type:** External Service Interface
- **Purpose:** Secure reception of mission telemetry and enforcement of classification and access policies
- **Security Characteristics:**
  - Mutual TLS authentication for all inbound traffic
  - Structured metadata validation (e.g. NATO classification tags)
  - Policy-based routing and conditional data rejection

From an architectural perspective, the interface is modelled in the **NSV-2** view via the `Confidential Data Ingestion` service function. It acts as a mediation layer between external data sources and internal analytic pipelines, ensuring compliance with trust and classification constraints.

The **NSOV-2** view specifies functional behaviours such as validation failure modes, exception handling and guaranteed schema conformity. Complementing this, the **NSV-4c** view defines concrete policy constraints, including transport layer encryption requirements and enforced metadata schemas for classified data.

The focused modelling of a single secured interface serves as an architectural exemplar demonstrating NAFv4-compliant interface design in cloud-native defence systems. The *PHM Ingest Gateway* highlights how zero-trust communication patterns and metadata-aware enforcement can be implemented at the interface level.

Depending on time and resource constraints, a secondary interface such as the *Attestation Key Gate* may be introduced at a later stage. However, this thesis prioritises a

single, methodologically complete example to ensure architectural clarity and relevance to the overall PHM use case.

Interfaces in defence-grade platforms such as *OmniAware* can be broadly categorised as external, internal or context-aware. External interfaces include secure API gateways such as the `PHM Ingest Gateway`, while internal interfaces typically link key components like SPIRE, `Vault` and classified workloads. In context-aware use cases, e.g. within CIVS, metadata tagging interfaces support classification enforcement and downstream auditability.

This guidance is exemplified by the explicitly modelled *PHM Ingest Gateway* interface, which demonstrates how architecture-level trust principles are operationalised in practice.

Although only a single interface is fully modelled in this thesis, all security-relevant interfaces are designed following Security-by-Design principles. These include access control (e.g. RBAC, ABAC), data classification enforcement (e.g. NATO-Restricted) and validation logic via schema-based verification (e.g. JSON, Protobuf). Authentication and authorisation mechanisms rely on SPIFFE identities and token-based systems, ensuring confidentiality and auditability across all interface endpoints.

This focused yet methodical treatment ensures that even minimal interface modelling aligns with the system's overall zero-trust posture, providing clarity for downstream implementation and future extension. For a structured overview of additional interface candidates that were not fully modelled in this thesis but may be prioritised for future implementation, refer to Appendix 5 and 6. These tables summarise further NSOV-2 service interface concepts derived from the PHM and CIVS use cases respectively.

The modelling and implementation of secure interfaces in the *OmniAware* platform adheres to a number of architecture-level design guidelines, particularly in the context of NAFv4-based defence systems:

- **Interface Classification:** Interfaces are explicitly distinguished as *Service Interfaces* (`NSV-2`) or, where applicable, as *Application or Technology Interfaces* (`NSV-3`), depending on their location in the architecture stack and enforcement scope.

- **Trust Anchors via Interface Policies:** Each interface must implement policy-bound enforcement mechanisms, including mutual TLS, SPIFFE/SPIRE-based workload identity validation or schema-based contract verification (e.g. JSON schema, Protobuf).

- **Communication Patterns:** Interfaces are secured using encrypted ingress/egress channels (e.g. mTLS) and apply Zero Trust principles for data-at-rest, in-transit and in-use protection.

- **Security Contracts:** All interface definitions must specify RBAC/ABAC rules, classification tags (e.g. NATO-Restricted), validation logic and auditability for policy violations.

- **NAFv4 Viewpoint Modelling:** Interfaces are formally represented and cross-linked in the architecture using the `NSV-2` and `NSV-4c` viewpoints.

These architectural patterns ensure that each interface acts not only as a communication mechanism, but also as an enforcement point for confidentiality, classification

compliance and runtime policy validation. This ensures a verifiable and modular modular trust perimeter within the PHM Landing Zone, demonstrating how mission-critical enforcement logic can be made explicit and auditable at the interface level.

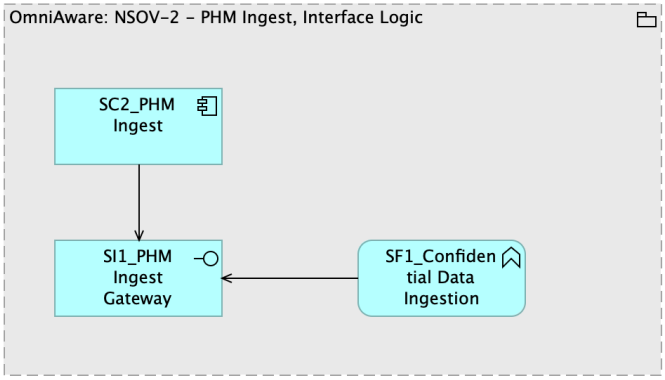


Figure 3.21: NSOV-2 - PHM Ingest, Service Interface

As illustrated in Figure 3.21, the `SI1_PHM Ingest Gateway` represents a formal **Service Interface** within the NSOV-2 viewpoint. This interface acts as the trusted ingress point for all mission-critical telemetry flows within the PHM use case. Its architectural role is twofold: (1) it enforces inbound *interface-level* security guarantees (e.g. TLS, metadata validation) and (2) it bridges the externally-facing ingestion logic with internal platform services.

The interface is logically `served` by the function `SF1_Confidential Data Ingestion`, which handles validation, decryption and semantic classification of received telemetry. This function ensures the correct application of downstream policies such as access control, tagging and audit enforcement. According to ArchiMate semantics, this relationship is modelled via a `serves` association.

Additionally, the interface is semantically `associated` to the service contract `SC2_PHM Ingest`, which defines the architectural guarantees, behavioural expectations and trust boundaries required for the ingestion process. The contract ensures that only metadata-conformant, attestation-bound and authenticated requests may traverse the interface. It acts as the formal service logic definition within the NSOV-2 model.

In summary, this model reflects a minimal but security-critical subset of interface logic. It demonstrates how service functions (`SF1`) expose interface endpoints (`SI1`) in accordance with a defined contract (`SC2`), thereby ensuring modular traceability and runtime policy enforcement in line with NAFv4-compliant design logic.

Figure 3.22 models the **NSV-2** interface logic of the `PHM Ingest Gateway`, representing a secured and policy-bound entry point for mission telemetry. The model focuses on the binding between the externally exposed service interface `SI1_PHM Ingest Gateway`, the functional service logic `SF1_Confidential Data Ingestion` and the implementing application and service components.

At the centre of this view lies the **Service Interface** `SI1_PHM Ingest Gateway`, which is served by the internal logic of `SF1_Confidential Data Ingestion`. The service logic enforces validation, classification compliance and policy-based routing. These behaviours are realised within the **Application Component** `AC1_PHM Ingest Handler`, which is directly *assigned to* the function and provides the concrete implementation logic for zero-trust data ingestion.

To explicitly separate architectural responsibilities, this model introduces a second internal **Service Component**, `SC2_PHM Ingest`. It captures the structural aggregation

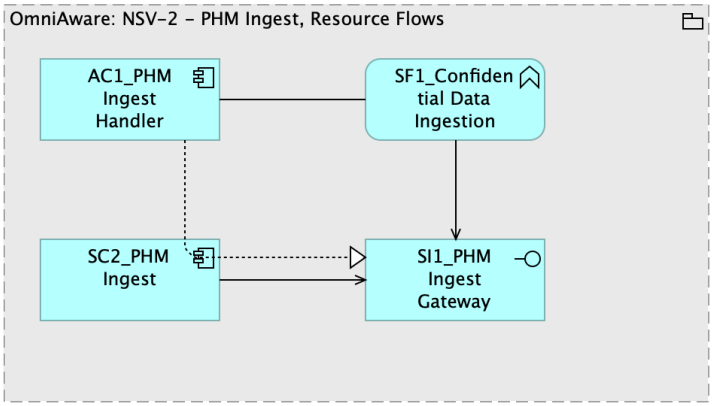


Figure 3.22: NSV-2 - PHM Ingest, Resource Connectivity

of the interface layer, acting as the architectural anchor point for workload-bound interface policies and policy enforcement mechanisms. While both `AC1` and `SC2` realise parts of the same overall capability, the separation reflects modelling constraints in `Archi`, where `Application Components` and `Service Components` must be instantiated as distinct element types. This decision ensures consistency with NAFv4-aligned tool support, even in the absence of native multi-role components.

The `SF1` service logic is *served* into the interface `SI1`, enabling structured inbound telemetry flows. The `SC2_PHM Ingest` component orchestrates this process by *serving* the exposed interface. Semantically, this connection symbolises the secure exposure of mission-relevant service logic to authenticated upstream systems. A further *association* between `AC1` and `SF1` underlines their structural linkage without conflating their distinct modelling semantics.

This focused NSV-2 representation illustrates how a single interface may be embedded within a secure service chain while remaining decomposable across component and service layers. It highlights the architectural principle that *every security-relevant interface must be traceably realised* within the functional service landscape, preserving compliance, traceability and implementation integrity.

Unlike NSOV-2, which emphasises the behavioural semantics and operational purpose of the interface from a service function perspective, the NSV-2 view captures the *resource flow* between architectural building blocks. Here, interfaces are not merely abstract contracts but are tightly linked to the application and service components responsible for their realisation. The service is thus contextualised within the deployment architecture, making NSV-2 essential for understanding how logical interface logic is operationalised as part of the platform's concrete service landscape.

Figure 3.23 presents the **NSV-4c** service interaction model for the `SI1_PHM Ingest Gateway` interface. The view highlights the interaction between service interfaces and interface-relevant security policies, describing the non-functional requirements that govern the classification-compliant and trust-preserving operation of the system.

This model defines three policy artefacts that influence the security posture of the ingest interface:

- `PE2_TLS Mutual Authentication`, ensuring bidirectional authentication via mTLS.
- `PE3_Classification Tag Enforcement`, enforcing structured classification of inbound mission telemetry via NATO labelling schemes.

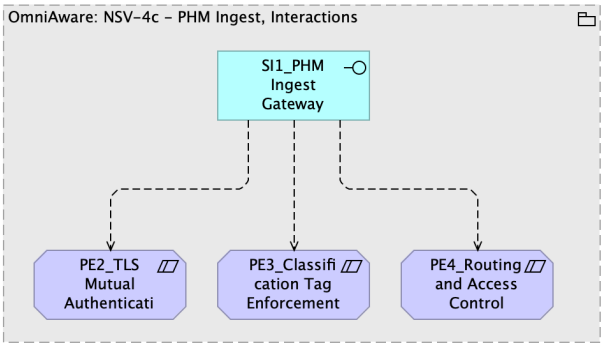


Figure 3.23: NSV-4c - PHM Ingest, Service Interactions

- `PE4_Routing and Access Control`, enabling policy-based rejection, routing or redirection based on classification level and contextual metadata.

These policies are formally modelled as `Constraint` elements and semantically linked to the `SI1` service interface using the `influenced by` relation. Although this deviates from the idealised *constrained by* relation found in conceptual modelling, the ArchiMate language does not provide a native construct for semantic policy constraints on service interfaces. The chosen workaround aligns with the methodological intent of NAFv4-compliant NSV-4c artefacts, which focus on explicitly declaring the interface-level impact of runtime policies.

The modelling rationale stems from the NAFv4 definition of NSV-4c, which emphasises the external **observable interaction behaviour of services under policy influence**. In zero-trust architectures, this includes authentication enforcement, schema validation and access control prior to function execution. Therefore, policies in NSV-4c are not implemented at the function or component layer, but are *exposed and enforced* at the interface boundary - where adversarial interaction may occur and where runtime rejection or gating mechanisms are required.

The `SI1_PHM Ingest Gateway` interface thus serves as the policy enforcement frontier, integrating classification constraints, trust anchors and zero-trust communication patterns at the system's entry point. The use of the `influenced by` relation is a deliberate modelling choice to preserve semantic fidelity while operating within the constraints of the ArchiMate metamodel.

The interface design presented in this section, exemplified by the `SI1_PHM Ingest Gateway`, reflects a methodologically sound and security-conscious approach to system interfacing within the *OmniAware* architecture. It demonstrates how a semantically traceable interface can be derived from capability-based planning and refined through service and component modelling, in full alignment with the NAFv4 methodology. The interface itself is designed to support zero-trust communication, remote attestation and policy-based routing of telemetry data towards confidential computing environments — all of which are indispensable features in mission-critical military scenarios.

While the current architectural modelling ensures compliance, traceability and conceptual integrity, it is important to acknowledge that the actual implementation of such interfaces may require further elaboration beyond what is currently specified. Specifically, aspects such as precise protocol bindings (e.g. gRPC, Representational State Transfer or message queues), schema validation (e.g. JSON Schema, Protobuf) and the handling of transport-layer exceptions or retries must be addressed during software implementation. Similarly, integration with identity providers (e.g. SPIFFE/SPIRE), key

management systems (e.g. HashiCorp Vault) and runtime policy engines (e.g. Open Policy Agent) requires concrete configuration artefacts and automation logic, typically expressed through IaC or declarative Kubernetes manifests.

In essence, the modelling of interfaces — while grounded in formal system architecture — should be seen as a launchpad rather than an endpoint. The design of the SI1_PHM Ingest Gateway provides a validated architectural anchor that ensures semantic consistency and security alignment. However, actual deployment will likely necessitate iterative refinement and expansion of the interface specification, particularly as additional dependencies and implementation constraints emerge. These may include telemetry data contracts, dynamic access policies, integration test frameworks or logging and observability mechanisms tailored for defence-grade assurance.

Future work could thus focus on the complete operationalisation of this interface within a prototype system, supported by continuous validation against compliance artefacts, threat models and stakeholder requirements. By formalising interfaces as integral parts of the architecture rather than isolated technical details, this thesis contributes a scalable and security-aligned blueprint for data-centric service integration within federated defence environments.

This chapter has demonstrated how the architectural principles of modularity, policy enforcement and zero-trust resilience can be systematically realised through NAFv4-conformant design patterns. Using the PHM scenario as a guiding use case, selected architectural artefacts such as NSV-4a, NSV-6, NPV-3 and NSOV-2 have been modelled to illustrate the interplay between secure execution environments, policy-driven trust enforcement and interface specification. Together, they provide a high-assurance blueprint for mission-critical telemetry flows in sovereign cloud environments.

While the architectural views presented in this chapter establish a rigorous foundation, they deliberately prioritise traceability and security semantics over full deployment coverage. The implemented subset of the architecture is designed to support forward compatibility with the next stage of development — the implementation and validation of a Minimum Viable Product (MVP).

In the following chapter, selected aspects of this architecture will be realised within a functioning prototype. The implementation will focus on enabling attested enclave execution, policy-bound key management and telemetry interface orchestration. In doing so, it aims to translate architectural integrity into deployable, measurable and operationally meaningful system components.

4 IMPLEMENTATION

4.1 DEPLOYMENT

4.1.1 *Deployment Strategy and Infrastructure Automation*

The *OmniAware* platform is deployed within the AWS Public Cloud to leverage its global infrastructure, high availability and scalable architecture patterns. During implementation, two distinct AWS account environments were provided by Capgemini to separate experimental prototyping from production-grade deployments: one provided by the *AWS Guild Germany* and one set of accounts managed by Capgemini's internal *GroupIT (GIT)* organisation.

The **AWS Guild account** constitutes a flexible, standalone AWS account that is managed within the AWS Guild's own budgetary framework. It was primarily used to design, implement and evaluate PoC components such as secure ingest APIs, `Vault`-based JWT token validation and the underlying confidential computing infrastructure. Due to limitations in the availability of AMD SEV-SNP, the deployment region for this account was set to `eu-west-1` (Ireland), which — alongside `us-east-2` (Ohio) —– is currently one of the only supported regions for SEV-SNP workloads [63]. This ensured that the early-stage architecture could be validated using enclave-based attestation flows.

In contrast, the **GIT accounts** represent enterprise-managed AWS environments that are subject to Capgemini's security and compliance guardrails. They are directly mapped to the internal project identifier of *OmniAware* and are linked to dedicated cost centres for accountability and billing transparency. These accounts are closer to production readiness and reflect the expected customer deployment context. For the current MVP, five GIT-managed accounts were used:

- `OmniAware-Ingest`
- `OmniAware-Datalake`
- `OmniAware-Security`
- `OmniAware-Audit`
- `OmniAware-Consumer`

The deployment region for these accounts was `eu-central-1` (Frankfurt), which offers the most comprehensive set of AWS services within the EU and ensures close alignment with European data protection regulations. While `eu-central-1` does not yet fulfil all requirements for strict sovereign data residency, it was selected to support jurisdictional alignment and to enable the application of mission-relevant compliance baselines, such as ISO/IEC 27001, Center for Internet Security (CIS) Benchmarks and WAF principles. The first ESC-compliant region in Germany is anticipated to be established in the state of Brandenburg, which may serve as the target zone for future production deployments once regional feature parity is achieved [64].

All components of the deployed architecture are provisioned via Infrastructure-as-Code (IaC) using AWS CloudFormation. This guarantees repeatability, traceability and

integrity of the deployment pipeline across development and operational stages. Each IaC template integrates compliance logic such as policy constraints, tag enforcement, and environment scoping. Full templates are included in Appendix 6.2.

IaC Framework Selection and Automation Strategy. Modern cloud-native platforms require infrastructure provisioning to be reproducible, traceable and policy-compliant by design. In this context, the selection of an Infrastructure-as-Code (IaC) framework becomes a critical architectural decision. Various frameworks exist, including Terraform, Pulumi, Helm and AWS CloudFormation — each with distinct trade-offs in terms of lifecycle integration, compliance alignment and operational security.

Terraform is widely used and supports multi-cloud orchestration, but introduces an external state backend and an execution context decoupled from the AWS control plane. This requires additional components such as remote state locking, backends (e.g. S3 with DynamoDB) and external orchestration logic. Pulumi introduces support for modern programming languages but suffers from similar external execution constraints. Helm, while natively integrated with Kubernetes, is declarative and release-oriented, but not suited for managing low-level infrastructure resources beyond container deployment layers.

By contrast, AWS CloudFormation operates natively within the AWS control plane. It provides tight integration with internal security services (such as IAM), enforces policy-linked guardrails, supports scoped parameterisation and allows direct referencing of service-linked roles. Its tight coupling with AWS Config, Audit Manager and Security Hub enables context-aware remediation and compliance validation during deployment.

These capabilities are particularly valuable for mission-critical systems with strict security requirements and air-gapped operating conditions. For these reasons, CloudFormation was selected as the IaC framework for this thesis, as it ensures full reproducibility, avoids external orchestration surfaces, reduces supply chain exposure and supports WAF-aligned governance principles throughout the deployment lifecycle.

CI/CD Integration. To operationalise the deployment process and ensure architectural reproducibility, a multi-stage CI/CD mechanism based on Bash scripting and CloudFormation orchestration was implemented. The pipeline integrates environment-specific parameters, template validations and conditional deployment logic to reflect project stage, target account and region.

Currently, the CI/CD system relies on networked execution from within trusted development environments, supporting cross-account deployments and infrastructure initialisation via explicitly defined IAM roles and permission boundaries. While not yet optimised for full air-gapped operation, the pipeline enforces strict execution roles, avoids external orchestration layers and supports regional deployments aligned to internal governance standards.

This strategic setup enables progressive automation of the deployment lifecycle and lays the foundation for future enhancements, such as full offline support and secure pipeline containerisation.

4.1.2 *Infrastructure-as-Code and Automation Pipelines*

At the architectural compliance layer, a modular CI/CD pipeline governs the automated provisioning, validation and deployment of infrastructure components. The design follows a lightweight approach based on AWS CloudFormation and cross-account role

delegation, deliberately avoiding external orchestration platforms and DevOps SaaS tooling. This ensures controlled execution within trusted environments and minimises the external dependency surface —– particularly relevant in defence-related or classified settings.

CI/CD Pipeline Design. The implemented CI/CD pipeline is defined as a set of Bash-based orchestration scripts invoking parameterised CloudFormation templates within isolated AWS accounts. Role assumptions are strictly scoped via permissions boundaries and IAM policies, ensuring secure deployment propagation across the defined landing zone architecture (cf. Section 3.2).

While the current pipeline is not yet designed for fully air-gapped or enclave-verified execution, it follows clear principles of reproducibility, policy-based validation and infrastructure provenance. Build and deployment artefacts are version-controlled, parameterised and injected per execution context (region, account, stage). Manual triggers ensure developer control over each deployment lifecycle phase.

Strategically, the design reflects a compromise between architectural clarity and implementation feasibility. It provides minimal yet sufficient automation to support consistent deployments, without introducing third-party control planes or external service dependencies. Future iterations may adopt secure runners or confidential build stages to cryptographically attest deployment provenance (cf. Section 3.2).

Pipeline Stages. The pipeline comprises the following structured execution stages:

- **Source and Template Management:** Declarative templates reside in version-controlled Git repositories. Deployment is initiated manually by executing the CI/CD orchestration scripts.
- **Runner Invocation:** Within each deployment call, environment-scoped parameters are passed to invoke region-specific CloudFormation stacks using pre-authorised IAM roles (e.g. `AssumeRole` with scoped permissions).
- **Validation and Policy Enforcement:** Templates are statically validated before deployment. Parameter integrity and template consistency are ensured through script-based checks and naming convention enforcement.
- **Orchestrated Deployment:** Templates are deployed to their respective landing zone accounts (e.g. Ingest, Datalake, Audit) using stack-based orchestration logic.
- **Post-Deployment Integration:** Manual post-deployment actions finalise tagging, CloudTrail enablement and GuardDuty baseline integration. These steps are modularised for future automation.

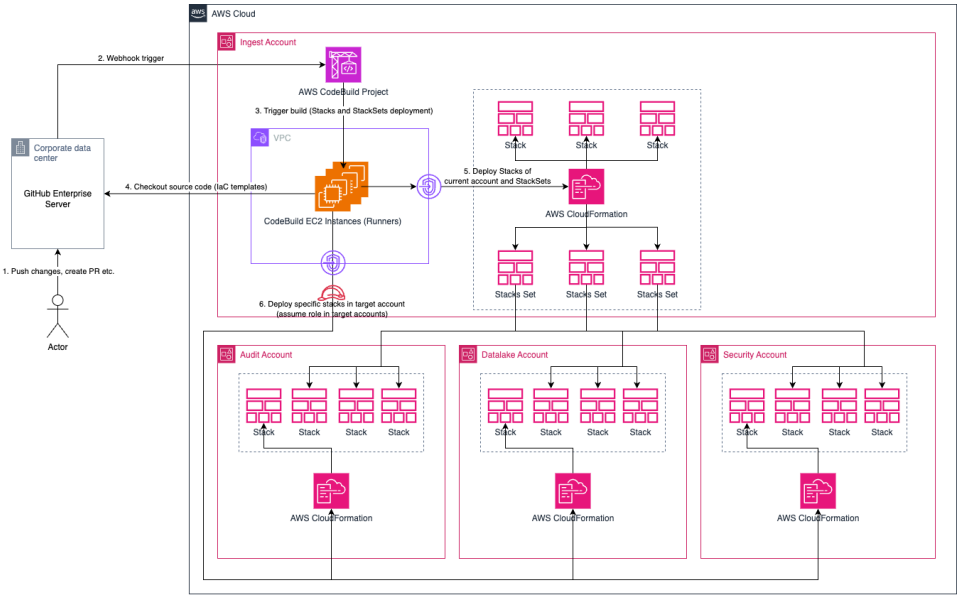


Figure 4.1: CI/CD Pipeline for Secure Deployment of Landing Zone Components

As shown in Figure 4.1, the pipeline encapsulates source management, parameter injection and orchestrated deployment across multiple isolated accounts. It establishes a coherent foundation for future enhancements —– such as build-time attestation, dynamic policy overlays or zero-trust provisioning stages —– without compromising the current project's reproducibility or compliance objectives.

Note on Implementation Status. While Figure 4.1 illustrates the strategic CI/CD target architecture for *OmniAware*, the current PoC implementation does not yet leverage fully managed AWS CodeBuild runners or webhook-based integration. Instead, infrastructure provisioning is performed via parameterised Bash scripts executed manually or from secure bastion hosts (cf. Listings 4.1.2, 6.2). This approach ensures transparent execution context and credential control during early-phase deployments, while remaining consistent with the role assumption, account scoping and landing zone propagation illustrated in the architectural model. As the platform matures, a transition to managed runner orchestration is anticipated.

CI/CD Pipeline Structure. The directory layout of the CI/CD pipeline reflects a logically structured orchestration architecture that mirrors the scoping of the multi-account AWS Landing Zone setup. The Git-based repository is segmented into reusable Bash-based orchestration scripts, such as `deploy_stacks.sh`, `deploy_stack_sets.sh` and `deploy_stacks_wrapper.sh`, which abstract the deployment logic across different trust zones. Each script is designed to inject mission-aware deployment metadata and enforce IAM-scoped identity assumptions via consistent naming, tagging and permissions constructs.

The core components of the deployment logic are divided into three primary scopes:

- **Scoped Execution Contexts**: The `init_stack.yaml` template defines execution roles with IAM trust boundaries and permission boundaries for both direct stack deployments and delegated stack set executions (cf. 4.1.2). This initialisation anchors the role assumption logic that governs the orchestration lifecycle.

- **Isolated Orchestration Scripts**: Wrapper scripts (e.g. `deploy_stacks.sh`) are designed to operate in tightly scoped identity contexts, typically from bastion hosts or designated deployment environments. The use of `aws sts get-caller-identity` ensures contextual account resolution and prevents misrouting across accounts. Deployment metadata (e.g. `ProjectName`, `VpcId`, `Environment`) is injected via externalised `parameters.json` files.

- **StackSet Propagation and Compliance Enforcement**: The `deploy_stack_sets.sh` script allows for propagation of compliance-critical stacks (e.g. `10_guardrails.yaml`) across multiple accounts using the AWS StackSet service. By leveraging scoped delegation roles and predefined trust paths, the design ensures policy-aligned baseline enforcement from the control plane account without compromising the separation of trust domains.

The `deploy_stack_sets.sh` orchestration script leverages the native AWS StackSet service to propagate compliance-critical infrastructure components (e.g. `10_guardrails.yaml`) across all governed target accounts from a central control plane.

StackSet enables the management of CloudFormation stacks across multiple AWS accounts and regions from a single administrator account. It supports both self-managed and service-managed permission models, with the latter enabling delegation via predefined IAM roles. In the current deployment design, StackSets are configured using the delegated administrator model with scoped identity roles, allowing secure propagation while enforcing strict cross-account boundaries.

Alternative propagation mechanisms — such as invoking individual stack deployments via scripting loops or orchestrating through third-party tooling (e.g. Terraform Workspaces, Ansible Tower or AWS Control Tower) —– either introduce external dependencies, increase operational overhead or violate the design principle of scoped credential delegation. Unlike AWS Control Tower, which provides a fully managed setup and lifecycle for multi-account environments, the use of StackSet offers greater flexibility and control. While Control Tower automates account creation and baseline governance, it enforces a fixed structure and prescriptive workflows. In contrast, StackSet-based orchestration allows custom stack placement, scoped identity assumptions and tailored propagation logic — making it better suited for fine-grained, mission-driven deployments.

The use of StackSet is therefore deliberate and aligned with architectural objectives such as:

- *Policy consistency*: Governance overlays are deployed uniformly, reducing configuration drift and ensuring compliance with project-wide security baselines.
- *Minimal blast radius*: Delegation roles constrain the propagation scope, ensuring that only pre-authorised accounts receive and execute the stacks.
- *Compatibility with compliance frameworks*: The declarative and centrally controlled nature of StackSet deployments supports traceability and auditability, which are critical under frameworks such as C5, TSE-SE and NATO accreditation procedures.

This method offers a robust balance between automation and control, ensuring that policy-aligned infrastructure components are enforced across the multi-account Landing Zone without compromising the separation of trust domains or identity integrity.

The orchestration layout aligns with the Landing Zone directory hierarchy introduced in Section 4.1.3, with deployment scripts grouped according to their responsibility and trust scope. Shared deployment artefacts such as `init_stack.yaml` and `10_guardrails.yaml` reside in `shared/stacks` and `shared/stacksets`, while regional execution wrappers such as `deploy_stacks-eu-west-1.sh` reflect localisation within the corresponding environment tier.

Implementation Artefacts and Design Decisions. The CI/CD pipeline's secure and reproducible behaviour is operationalised through minimal but expressive orchestration artefacts. The following listings demonstrate two core components: A Bash-based deployment wrapper for invoking CloudFormation templates within scoped contexts and an identity template that provisions IAM roles under tightly bound permission structures.

`deploy_stacks.sh`
Bash-based wrapper for scoped deployment of parameterised CloudFormation templates using AWS CLI. Ensures strict account targeting and avoids external toolchain dependencies.

```
#!/bin/bash
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION="eu-west-1"
TEMPLATE_FILE="15_cc-vault-poc.yaml"
STACK_NAME="vault-poc-stack"

aws cloudformation deploy \
  --stack-name $STACK_NAME \
  --template-file $TEMPLATE_FILE \
  --region $REGION \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    ProjectName=OmniAware \
    Environment=dev \
    VpcId=vpc-xxxxxxxx \
    AdminRoleName=GuildAdminRole
```

[1]

1 This wrapper script deploys parameterised CloudFormation templates into a designated AWS environment using manually scoped identity context. External runtimes are avoided by invoking AWS CLI locally. This listing illustrates a concrete `Vault` PoC instantiation; the script itself supports generalised use.

The `deploy_stacks.sh` script exemplifies the platform's minimalistic orchestration principle. Rather than relying on fully managed services like AWS CodePipeline, the approach uses shell-based invocation to maintain fine-grained control over execution scope, parameter injection and credential context.

By querying the current `AccountId` at runtime (`aws sts get-caller-identity`), the script ensures that deployment is tightly bound to the executing identity and cannot be misrouted to unintended accounts. The `-parameter-overrides` section allows full injection of contextual deployment metadata such as `ProjectName`, `Environment` and `VpcId`, which aligns each stack to a specific landing zone tier.

This local, parameterised deployment model supports transparency, developer-led iteration and a clear separation of execution environments without relying on remote build runners or opaque deployment logic.

`init_stack.yaml`

Minimal CloudFormation template for establishing permissions boundary, IAM roles and resource tagging foundation.

```
Resources:
  VaultExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub "VaultExec-${Environment}"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess
      PermissionsBoundary: !Ref PermissionBoundaryArn
```

[2]

The `init_stack.yaml` template establishes the foundational identity constructs for the deployment lifecycle. It defines a scoped IAM role (`VaultExecutionRole`) with explicit trust boundaries and a named permissions boundary reference.

The `AssumeRolePolicyDocument` enforces identity constraints, allowing only EC2 instances to assume this role. The use of a named permissions boundary—passed as a parameter—ensures that even future extensions of the policy cannot exceed predefined limits, preserving least-privilege semantics.

This design minimises the blast radius of potential misconfigurations or lateral privilege escalation, which is particularly relevant in sensitive multi-account environments such as those used in the *OmniAware* landing zone setup. It also supports cross-account deployment patterns by embedding trust anchors that align with the scoped deployment permissions invoked in the orchestration script.

Deviation from Target Architecture. While the architecture diagram (cf. Figure 4.1) illustrates a target deployment model based on AWS CodeBuild and webhook-triggered pipeline execution, the current PoC implementation does not yet provision a fully managed CI/CD framework.

2 IAM roles are defined with scoped permission sets and a permissions boundary to enforce least-privilege execution. This prevents privilege escalation across landing zone accounts.

Instead, infrastructure provisioning is operationalised via parameterised Bash scripts (cf. Listing 4.1.2), executed manually or from controlled environments such as bastion hosts. These scripts invoke `aws cloudformation deploy` with environment-specific overrides and scoped IAM role assumptions, ensuring secure and isolated deployment logic.

This interim setup offers full transparency and control during the prototyping phase and lays the groundwork for later transition into managed CI/CD environments. The architectural consistency of parameter injection, role-based isolation and landing zone account structure is maintained throughout, enabling a seamless upgrade path toward CodeBuild-based automation.

The current pipeline implementation was partially provided by a project collaborator. While foundational scripting and CI/CD orchestration were preconfigured, all environment-specific adaptations, deployment parameterisation and compliance-aligned execution logic were developed by the author. These include scoped IAM role assumptions, isolated landing zone targeting and reproducible deployment primitives for mission-grade environments.

4.1.3 *Core Infrastructure Deployment: Landing Zone Foundation*

Following validation of the account-wide initialisation phase, the core infrastructure of the `Landing Zone` is deployed through a structured sequence of modular IaC components version-controlled within the project's GIT repository. Each stack defines a scoped and reproducible infrastructure primitive that collectively establishes the operational substrate for mission-grade workloads. The deployment pipeline integrates compliance-focused parameterisation, context-aware role assumptions and tagging conventions for consistent policy enforcement across all trust domains.

Landing Zone Structure. The layout of the `05_iac` directory within the GIT setup mirrors the logical structure of the AWS multi-account Landing Zone. Account-specific components are mapped to subdirectories such as `audit/`, `datalake/` or `ingest/`, reflecting the distinct organisational units of the landing zone. Shared or cross-account logic resides in `shared/`, including global control layers such as `init/` and `guardrails.yaml`. This separation enforces a clear architectural distinction between globally applicable governance overlays and account — scoped modules such as `00_kms_datalake.yaml`. The approach is compatible with the principles of the AWS Landing Zone Accelerator, while maintaining the flexibility to tailor stack placement and sequencing based on mission needs.

- `shared/stacks/init/init_stack.yaml`: Initialises cross-account trust boundaries and defines delegation roles for stack set execution.
- `security/stacks/00_kmsKeys.yaml`, `00_kms_ingest.yaml`, `00_kms_datalake.yaml`: Define mission-specific KMS encryption boundaries for logging, telemetry and analytics.
- `shared/stacksets/10_guardrails.yaml`: Defines preventive governance overlays (e.g. SCPs), compliance rules and security baselining.

Deployment Sequence and Modules. The foundational deployment sequence reflects a layered approach to establishing core Landing Zone infrastructure. Each component

targets specific trust domains, control planes and compliance overlays. The sequence is optimised to enforce separation of concerns, support security baselining and enable workload — aligned policy overlays:

- `00_kmsKeys.yaml`: Declares centralised KMS keys to protect log aggregation, telemetry streams and shared analytic pipelines. Access control is limited to scoped roles via `KeyUsers`, `KeyAdmins` and `KeyServiceRoles`, enforcing least-privilege access and policy separation.
- `00_kms_ingest.yaml`: Instantiates telemetry-scoped KMS keys within the `Ingest` account to enable encryption of sensor streams and ingestion buffers. Workload-specific tags (e.g. `Workload=Telemetry`) and ARNs-based boundaries support inheritance of downstream policy overlays and fine-grained auditability.
- `00_kms_datalake.yaml`: Deploys a mission-specific KMS key in the `Datalake` account. The configuration enables `EnableKeyRotation`, aligns with CIS recommendations and supports structured tagging for compliance propagation across analytic zones.
- `10_guardrails.yaml`: Provides account-wide preventive and detective controls through SCPs, AWS Config and additional overlays. It enforces MFA, disables inline IAM, blocks RDP/SSH access points, mandates SSE-KMS encryption and enforces IMDSv2-only configuration for EC2 instances.

Deployment orchestration is implemented via Bash-based wrapper scripts —— `deploy_stacks.sh` and `deploy_stack_sets.sh` —— which encapsulate AWS CLI logic for secure parameter injection and role-based delegation. These scripts assume pre-scoped roles via `AssumeRole`, enforce consistent naming patterns (e.g. `ProjectName`, `Environment`, `AccountId`) and retrieve parameters from version-controlled `parameters.json` files. The resulting infrastructure is securely and repeatably deployed across trust boundaries. This setup remains extensible for sovereign and mission-specific use cases and supports a seamless upgrade path toward policy-enforced CI/CD-driven automation.

`00_kms_datalake.yaml`
Minimal CloudFormation template to exemplify the KMS foundation with secure key provisioning scoped to analytic workloads. The configuration below enforces key rotation and workload tagging.

```
Resources:
  DatalakeKmsKey:
    Type: AWS::KMS::Key
    Properties:
      EnableKeyRotation: true
      Description: "KMS key for Datalake telemetry and analytics"
      KeyPolicy: !Sub ...
      Tags:
        - Key: Workload
          Value: Analytics
```

`10_guardrails.yaml`
Minimal CloudFormation template to define a portion of guardrails with preventive controls via explicit SCP overlays, including enforcement of modern identity and access management practices:

```
Policies:
  DenyLegacyIamActions:
    Type: AWS::Organizations::Policy
    Properties:
      PolicyName: "DenyLegacyIAM"
      Content:
        Version: "2012-10-17"
        Statement:
          - Effect: Deny
            Action:
              - iam:CreateUser
              - iam:PutUserPolicy
            Resource: "*"
```

Stacks are executed either via isolated environments (e.g. bastion hosts or ephemeral EC2 runners) or orchestrated from the `Ingest` account using `deploy_stack_sets.sh`. Each stack incorporates a unique `ProjectName`, `Environment` and `AccountId` to ensure compliance with project-wide naming schemas. The parameterisation is sourced from external `parameters.json` files, which are version-controlled and validated in pre-deployment stages.

This modular deployment ensures strict separation of concerns by isolating encryption domains, access controls and governance logic per functional domain. Tagging schemes are applied consistently to support automated policy overlays and enable audit trail validation. This results in an infrastructure foundation upon which critical workloads (e.g. PHM, CIVS) can be securely deployed and cryptographically attested across multiple trust domains and enclave-backed security zones.

Security Considerations. The decision to deploy mission-specific KMS keys reflects their pivotal role in enforcing encryption-at-rest policies, managing cryptographic access separation and underpinning audit traceability across all landing zone domains. Compared to alternative services such as `AWS Secrets Manager` or `Parameter Store`, KMS integrates more directly with IAM-scoped permissions, supports automatic key rotation and aligns natively with advanced compliance regimes (e.g. ISO/IEC 27001, NIST 800–53).

The `guardrails.yaml` template embodies a governance-first approach to preventive and detective control enforcement. The term *guardrails* originates from the principle of embedding non-negotiable policies early in the infrastructure lifecycle to constrain misconfigurations and security drift. These controls — codified via SCPs, AWS Config Rules and CloudTrail insights — extend the shared responsibility model by enforcing least-privilege, zero-trust assumptions from the outset.

Alternative strategies —– such as using service-linked roles, organisation-wide tagging policies or reactive drift detection —– offer varying levels of coverage but lack the proactive enforcement guarantees provided by SCP-based guardrails. In multi-tenant or federated coalition contexts, such as those governed by NATO or EU structures, guardrails help to assert workload boundary isolation across sovereign domains.

When adapting this setup for mission owners or sovereign MoD customers, additional security layers may be required. Customers operating under NATO directives or frameworks such as C5 or TSE-SE may mandate enclave-verified deployments, remote attestation workflows or constrained infrastructure-as-code execution under certified policies.

For example, TSE-SE deployments may necessitate integration with `Nitro Enclaves` or `AMD SEV-SNP`, wherein even KMS access must be bound to attestation tokens. Furthermore, C5-aligned operations often prohibit externally managed orchestration (e.g. third-party CI/CD runners) and require audit-friendly artefacts such as signed build manifests and reproducible deployment logs.

Therefore, while the current PoC pipeline offers high reproducibility and composability, deployment within a customer-controlled environment may require tightening of execution

boundaries, increased segregation of duties and policy-aligned customisations. These adaptations are essential to align with sovereign control expectations and defence-grade certification regimes.

4.2 SECURITY AND COMPLIANCE CONTROLS

The increasing complexity of mission systems and their distributed execution across hybrid and sovereign cloud environments demands rigorous security and compliance controls. These must not only reflect prevailing security standards (e.g. CIS, NIST, ISO 27001) but also incorporate cutting-edge enforcement techniques and traceability mechanisms aligned with operational needs. Within the scope of the `OmniAware` platform, a defence-in-depth approach is adopted that spans infrastructure encryption, privileged access management, cross-account identity controls and runtime integrity attestation.

While foundational security mechanisms such as IAM permissions boundaries, SCPs and logging policies were already established in Section 4.1, this section extends the compliance baseline with deeper technical enforcement across core workloads. Notably, hardware-assisted trust anchors via TEEs were examined to fulfil emerging demands for runtime integrity and cross-domain provenance — particularly in sensitive mission deployments. The investigation focused on AMD SEV-SNP as the primary attestation mechanism, while AWS Nitro Enclaves were concurrently explored for comparison and potential dual-mode enforcement. Based on controlled deployment trials, the project successfully implemented a validated *Remote Attestation* workflow via SEV-SNP in a secure testbed, establishing a verifiable trust boundary for future enclave-integrated systems.

4.2.1 *Guardrail Enforcement and Extended Compliance Scope*

To operationalise baseline security principles, a dedicated StackSet template (`10_guardrails.yaml`) was implemented. The guardrails concept extends standard SCP-based preventive controls with AWS Config-based detective and responsive mechanisms. These enforce security posture across all `Landing Zone` accounts in alignment with platform-level policies.

The following controls were actively monitored and enforced:

- `S3 Public Access Block`: Ensures explicit denial of public access at both account and bucket level to avoid accidental data exposure.
- **EBS Volume Encryption**: Validates mandatory use of SSE-KMS-encrypted volumes for all EC2 instances, scoped to mission-specific KMS keys.
- **EC2 IMDSv2 Enforcement**: Mandates exclusive usage of IMDSv2, thereby disabling vulnerable legacy metadata endpoints.
- **CloudTrail Multi-Region Enablement**: Activates cross-region CloudTrail logging to provide consistent traceability and support forensic incident response.

These guardrails exemplify defence-in-depth: while preventive boundaries avoid misconfigurations at deployment, detective policies enforce continual runtime compliance. The StackSet propagation ensures that even dynamic workloads across AWS accounts remain policy-aligned.

4.2.2 *Secure Infrastructure Foundation: Guild Account Deployment*

To provide a controlled and extensible context for validating TEE trust anchors and ensuring compatibility with SEV-SNP, the remote attestation environment was instantiated within a separate, guild-linked AWS account. The foundational deployment was defined in the `10_cc-secure-infra-attestation.yaml` stack 6.2, designed to overcome permission issues, role propagation limitations and deployment obstacles observed in the primary GIT-linked development account.

This infrastructure setup offered improved isolation, streamlined compliance enforcement and simplified integration with downstream modules — ultimately making it the preferred platform for implementing and testing hardware-based attestation logic within project time constraints.

The deployment comprises a complete security-first infrastructure baseline, including segregated VPC layout, subnet isolation, NAT routing boundaries, scoped IAM roles, project-specific KMS keys and restrictive SecurityGroup definitions. The following example illustrates the configuration of a dedicated KMS key for use within attestation workflows:

10_cc-secure-infra-attestation.yaml
Scoped KMS Key for Attestation Workflows.

```
AttestationKMSKey:
  Type: "AWS::KMS::Key"
  Properties:
    Description: "KMS Key for Remote Attestation Test Secrets"
    KeyPolicy:
      Version: "2012-10-17"
      Statement:
        - Sid: "Enable IAM User Permissions"
          Effect: "Allow"
          Principal:
            AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root"
          Action: "kms:*"
          Resource: "*"
        - Sid: "Allow EC2 Service"
          Effect: "Allow"
          Principal:
            Service: "ec2.amazonaws.com"
          Action:
            - "kms:Decrypt"
            - "kms:GenerateDataKey"
            - "kms:CreateGrant"
```

`Vault` is an open-source system for secret management, access control and dynamic credential provisioning. In this project, it is used to verify remote attestation claims (e.g. via SEV-SNP) and release secrets only to trusted workloads, establishing a cryptographic trust anchor for workload integrity. Although `Vault` manages secrets internally, a dedicated KMS key is integrated into the attestation flow to enforce scoped cryptographic permissions, support temporary grant logic and ensure audit traceability. This key complements `Vault`'s access model by acting as a policy-bound control point for sensitive operations post-attestation.

In alignment with zero-trust and least-privilege principles, the networking layout was tightly scoped: public and private subnets were logically separated, EC2 traffic routed via a managed NAT gateway and communication with `Vault` services was constrained to internal routes protected by a self-referencing SecurityGroup. This ensured that lateral movements were only possible within scoped identity domains.

To meet the runtime requirements of SEV-SNP-enabled attestation hosts, dedicated EC2 instances were configured using vetted Ubuntu or Amazon Linux 2 AMIs with matching CPU configurations. The following `LaunchTemplate` snippet defines the core properties of the instance configuration:

10_cc-secure-infra-attestation.yaml
SEV-SNP EC2 Launch Configuration.

```
SEVSNPLaunchTemplate:
  Type: "AWS::EC2::LaunchTemplate"
  Properties:
    LaunchTemplateData:
      InstanceType: "c6a.large"
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AmazonLinuxAMI]
      CpuOptions:
        AmdSevSnp: "enabled"
```

The selection of AMIs was guided by the compatibility requirements of both hardware-based

attestation technologies evaluated during the project. For SEV-SNP, only recent Ubuntu LTS distributions and hardened Amazon Linux 2 (AL2) images provide the required kernel versions and firmware interfaces to interact with AMD SEV-SNP's extensions. In contrast, Nitro Enclaves mandates AL2 or custom-built enclave-compatible AMIs to support enclave configuration via `nitro-cli`. Given the exploratory nature of the Nitro path and the production maturity of SEV-SNP for Linux-based telemetry workloads, the final deployment standardised on a vetted Ubuntu AMI with built-in SNP support.

This secure deployment blueprint laid the technical foundation for the Remote Attestation module presented in Section 4.2.3. Preliminary investigations with Nitro Enclaves were also performed to assess their applicability, though limitations in enclave-specific tooling, networking constraints and compatibility with `Vault` attestation flows made SEV-SNP the preferred enforcement model under the given conditions.

The resulting baseline complies with common SCP-based restrictions (e.g. `iam:CreateUser` denial, blocked inline policies) and is extensible to mission-specific compliance mandates, including aforementioned accreditation frameworks.

By embedding policy enforcement mechanisms into both structural and procedural layers — including tagging conventions, identity boundaries and parameterisation — the platform lays a consistent foundation for advanced attestation workflows, which are dissected in the subsequent section.

4.2.3 *Remote Attestation and Key Management*

Summary: Remote attestation constitutes a foundational capability for enforcing cryptographic trust boundaries in confidential computing. It enables the integrity validation of sensitive workloads prior to secret disclosure, binding runtime and deployment context to mission-specific access conditions. In the context of military-grade edge and cloud infrastructure, this mechanism is particularly critical for safeguarding data sovereignty and operational trust in coalition-based or untrusted domains.

This section introduces a comprehensive blueprint for implementing attestation-driven access control using SEV-SNP, a dedicated `Vault`-based key management setup and supporting infrastructure components. The deployment builds upon the secure environment defined in `10_cc-secure-infra-attestation.yaml`, where attestation-ready EC2 instances, isolated networking zones and KMS-bound trust anchors provide the structural enforcement substrate. Complementary to this, the `15_cc-vault-poc.yaml` stack provisions a hardened `Vault` instance, configured to validate cryptographic claims and selectively release secrets based on successful attestation.

All artefacts and trust bindings are orchestrated via a lightweight CI/CD pipeline and parameterised through `parameters.json`. Critical operations such as token signing, claim encapsulation and verifier communication are encapsulated within a custom-built Python module (`PyJWT.py`), which serves as the logical interface between attesting workloads and the verifier service.

While the project also explored AWS Nitro Enclaves as a potential alternative for runtime isolation, implementation constraints — particularly the need to embed and adapt the Nitro Enclaves C-SDK [45] into mission workloads — led to the decision to deprioritise this path within the PoC. Corresponding insights are discussed in the validation section, but no functional integration of Nitro Enclaves was achieved within the available timeframe.

The remainder of this section dissects the attestation workflow in detail, including attester preparation, verifier design, enclave-bound policy enforcement and secure secret delivery — providing a reproducible, policy-driven model for mission-grade remote attestation in distributed defence systems.

Core Components of Remote Attestation Architecture

An attestation-enabled confidential computing environment typically comprises the following key components:

- **Attester:** The TEE or enclave-enabled workload that generates a signed attestation report (e.g. using AMD Secure Processor or Nitro Enclave Attestation Document).
- **Verifier:** A trusted external entity that evaluates the attestation report against a set of policies or allowlists (e.g. expected measurements, build IDs, firmware versions).
- **Key Broker or Key Management Service (KMS):** A system that releases cryptographic secrets only upon successful attestation validation.
- **Policy Engine:** Optional component enforcing additional conditions such as geolocation, time constraints or mission role attributes.

Key and Attestation Services

Several existing services are suitable for implementing this architecture in a defence context:

1. `HashiCorp Vault`
   - Provides `Transit Secrets Engine` for envelope encryption and policy-controlled key release.
   - Integrates with custom or built-in attestation verifiers via plugins or REST-based workflows.
   - Supports external integration with SEV-SNP and Nitro Enclaves via project-specific extensions [61].
2. **AWS Key Management Service (KMS)**
   - Supports **KMS for Nitro Enclaves** through the Enclave SDK, which validates the *Attestation Document* before granting access to KMS-delegated secrets [6], [40].
   - Limited to AWS-specific deployments; attestation logic is embedded in the Nitro SDK.
   - Can be extended with AWS Secrets Manager or custom KMS proxy with policy constraints.
3. **Confidential Consortium Framework (CCF)**
   - An open-source Microsoft initiative to support attested execution and ledger-backed policy enforcement.
   - Supports confidential ledgers and programmable policy control based on attestation evidence.
   - Can integrate with AMD SEV-SNP attestation reports and external certificate chains.
4. **SPIRE/SPIFFE (Secure Production Identity Framework for Everyone)**
   - Provides attestation-based identity issuance for workloads.
   - Supports integration with TPMs, TEEs and cloud-native enclaves.
   - Can serve as a *trust anchor* for downstream key release flows or mTLS-based service meshes.
5. **Keylime**
   - An open-source project for TPM- and TEE-backed remote attestation.
   - Compatible with AMD SEV-SNP through custom plugin development.
   - Focused on policy-driven continuous attestation and secure key bootstrapping.

6. **Azure Attestation and Managed HSM (optional for NATO-internal hybrid scenarios)**

    - Supports SGX-based attestation workflows and conditional access to Hardware Security Modules.
    - While Azure HSMs are not usable in EU Sovereign Cloud scenarios, the attestation mechanism can serve as reference architecture for coalition environments.

Table 4.1: Key Management Tools for confidential computing Integration [6], [40]

Criterion	HashiCorp Vault	AWS KMS	Fortanix DSM	Keylime
SEV-SNP Support	Yes (Custom)	No	Partial	Yes
Nitro Enclave Support	Partial	Yes	Yes	No
Integration with PoC	Flexible	Native	API-based	CLI
Policy Enforcement	High	High	High	Medium
Air-Gap / Sovereign-Ready	Yes	No	Yes	Yes
Open Source Available	Partial	No	No	Yes

Compatibility and Operational Considerations

- For AMD SEV-SNP, the attestation flow includes the use of `Versioned Chip Endorsement Keys (VCEKs)` and attestation reports signed by the Secure Processor. These can be validated against AMD's public certificate infrastructure and evaluated by a verifier such as `Vault`, SPIRE or Keylime [61]. While technically mature, SEV-SNP is currently only available in a limited number of AWS regions (namely `eu-west-1` and `us-east-2`), making global deployment less flexible compared to Nitro Enclaves [61].

- For Nitro Enclaves, the `Attestation Document` is structured as a JSON Web Signature (JWS), verifiable against AWS' public key. The attestation document can be passed to AWS KMS or a custom proxy for secret release [46].

For mission-critical deployments in the defence sector, the following requirements must be fulfilled:

- **Sovereign control of the attestation verification step**, ensuring that only national or NATO-accredited verifiers control secret release.
- **Offline-capable policy evaluation**, especially for edge or battlefield environments with intermittent connectivity.
- **Auditability and compliance traceability**, in accordance with STANAG 4774/4778 and *AC/322-D(2021)0032-REV1*.
- **Key release granularity**, tailored per mission, device or operator role.

The combination of SEV-SNP and Nitro Enclaves provides a powerful hybrid for trusted computing in NATO or EU defence scenarios. Key services such as `HashiCorp Vault`, SPIRE and AWS KMS with enclave integration offer flexible and extensible foundations for remote attestation workflows. Future work should focus on integrating sovereign attestation backends with hardware-rooted policies and formalising their compliance alignment under NATO cyber certification tracks.

HashiCorp Vault OSS. For the implementation of the prototype architecture, HashiCorp Vault in its open-source variant was selected as the key management solution. This decision was based on a combination of technical flexibility, security capabilities and integration feasibility under mission-relevant constraints. Vault OSS offers robust support for SEV-SNP-based attestation workflows through its extensible transit secrets engine, allowing for policy-enforced key release and cryptographic operations tied to attested workloads. Although native integration with Nitro Enclaves is not fully supported out-of-the-box, Vault can be extended via the exec plugin interface or integrated with the AWS Enclave SDK to build enclave-aware release logic.

Compared to other key management services, Vault OSS provides a unique balance between auditability, open customisation and sovereignty readiness. It can be deployed in air-gapped environments and supports full on-premises operation, thereby satisfying the strategic requirement of sovereign control over key material and attestation logic. Furthermore, the policy language (HCL) enables fine-grained access control and key release conditions, which can be enforced at runtime without reliance on external control planes.

Due to its open-source nature, Vault also supports transparent validation, which is essential for defence-grade deployments requiring verifiability of the attestation and key release logic. Unlike managed services such as AWS KMS or Fortanix DSM, Vault OSS incurs no licensing cost and avoids runtime dependencies on proprietary ecosystems — making it particularly suitable for PoC deployments in hybrid or sovereign mission domains.

In summary, Vault OSS was chosen to ensure technical alignment with attested confidential computing environments, to support sovereign control over cryptographic primitives and to maximise transparency and extensibility in the implementation phase of the platform.

Prototype Implementation: Confidential Key Release with Remote Attestation

To validate the feasibility of secure workload provisioning in defence cloud environments, a prototype system was implemented that demonstrates remote attestation-driven key release using TEE-based confidential computing. The prototype establishes a controlled execution environment in which cryptographic secrets are only provisioned to compute instances that have been successfully attested according to predefined security policies. This mechanism serves as a foundational security primitive for mission applications that require strong runtime guarantees and sovereign trust anchoring.

The implementation is based on a minimal but representative architecture comprising one instance each of an AMD SEV-SNP-enabled virtual machine and an AWS Nitro Enclave. These two platforms exemplify distinct attestation mechanisms — one targeting infrastructure-level VMs and the other focused on application-level enclaves within cloud-native contexts. A standalone HashiCorp Vault instance acts as the key management system (KMS) and policy enforcement point, equipped with transport layer security (TLS) and an attestation verification plugin.

At the core of the prototype lies a remote attestation workflow that includes the following stages: enclave instantiation, measurement generation, evidence signing by a hardware root of trust, validation by the verifier and conditional release of a high-sensitivity test secret (e.g. a symmetric AES-256 encryption key). This test secret is designed to emulate mission data or access credentials and is used to validate the complete key release chain.

Policy enforcement is realised through a declarative configuration within the Vault environment. The verifier module compares attestation reports against allowlisted measurements and metadata constraints such as enclave version, origin or launch timestamp. Only upon successful verification is the wrapped encryption key transmitted into the trusted memory space of the requesting enclave. At no point is the key exposed to the host operating system, hypervisor or any external observer.

This minimal attestation stack enables the emulation of defence scenarios in which access to critical mission data is cryptographically bound to the attestation state of the compute node. By leveraging attestation as a dynamic policy mechanism, the architecture allows for fine-grained trust decisions, including geofencing, mission time-boxing and platform-specific policy binding.

The prototype serves as a practical verification of the architecture proposed in Chapter 3, demonstrating how confidential computing mechanisms can be concretely implemented and

evaluated within a defence context. It also provides a blueprint for future extension toward more complex deployment topologies, including Kubernetes-integrated confidential workloads, multi-tenant enclave isolation and fully decentralised key provisioning.

Table 4.2: Remote Attestation and Key Management Prototype

Component Layer	Role in Attestation Workflow	Remarks
Confidential Runtime Environment	Hosts the trusted workload within a hardware-rooted enclave	**1x EC2 instance with SEV-SNP** **1x EC2 instance with Nitro Enclave-enabled**
`HashiCorp Vault (OSS)`	Key management service that enforces attestation-gated secret release	Deployed with TLS; runs standalone (or in dev mode for PoC)
Verifier Component	Validates attestation evidence against expected measurements and metadata	Implemented via `Vault` plugin or external policy enforcement module
Attestation Evidence Generator	Produces signed reports reflecting enclave state and identity	`sev-tool` (SEV-SNP) or Nitro Enclave SDK attestation interface
Secrets Policy Engine	Applies constraints for key release (e.g. PCR hash, enclave measurement, expiry)	Implemented via `Vault` HCL policy or custom validation logic
TLS Certificate Infrastructure	Secures communication between `Vault` and clients/verifiers	Self-signed or CA-issued; configured for `Vault` API endpoints
Test Secret (AES-256 key)	Validates the complete attestation-driven release workflow	Rotated regularly, used for decrypting synthetic mission payload

Methodological Approach

To demonstrate the practical feasibility of attestation-based key management in confidential computing environments, a minimal service deployment was selected. This configuration aims to capture the essential control flow for cryptographically verifiable secret release using remote attestation. The chosen stack combines a TEE-enabled runtime (e.g. SEV-SNP or Nitro Enclave) with `HashiCorp Vault OSS` as a flexible and extensible key management system.

The implementation follows a stepwise approach:

1. **Enclave Launch and Evidence Generation**: The attested workload is deployed in a TEE (e.g. SEV-SNP VM or Nitro Enclave). Upon launch, it generates a signed attestation report using the platform's root-of-trust (e.g. AMD-SP or Nitro Enclave SDK).
2. **Attestation Report Submission**: The workload transmits the attestation report to an external verifier or directly to `Vault` via a pre-defined API endpoint.
3. **Verification and Policy Matching**: The report is evaluated against an allowlist of expected enclave measurements and runtime parameters. `Vault` enforces policy conditions — such as measurement hashes, launch time constraints or mission roles — before proceeding.
4. **Key Release**: If verification is successful, the secret (e.g. symmetric key, config blob) is released into the enclave via short-lived, in-memory transfer. All actions are logged for compliance.

5. **Optional Audit and Replay Protection**: Additional metadata — such as request nonces, enclave ID or attestation timestamps — can be included to enforce auditability and replay resistance.

This architecture forms the foundational building block for trusted key provisioning in sovereign and mission-critical cloud environments. It ensures that cryptographic material is only accessible to verified workloads operating in trusted enclaves, thereby aligning with NATO and EU confidentiality, auditability and data sovereignty requirements.

Deployment Methodology for the Prototype

The implementation of remote attestation capabilities in the prototype followed a dual-path deployment methodology, reflecting two competing approaches to establishing confidential workload enforcement: AWS Nitro Enclaves and AMD SEV-SNP. Both approaches were evaluated against a shared infrastructure foundation that includes isolated network domains, scoped KMS policies and `Vault`-integrated secret delivery. The technical artefacts and pipelines for both paths are orchestrated via parameterised templates and shell wrappers as part of the broader Infrastructure-as-Code workflow. Given the reliance on AWS-managed services such as EC2, KMS and Nitro Enclaves, the deployment was executed in the `eu-west-1` region. This region was selected based on proximity, service availability and native support for SEV-SNP-enabled instance types during implementation. The entire setup was provisioned within a dedicated AWS account operated through the internal sandbox environment of the *AWS's Guild Germany*.

Path A: Nitro Enclave-Based Remote Attestation. Due to its native integration with AWS services and presumed ease of deployment, Nitro Enclaves were initially prioritised. A Nitro-compatible EC2 instance was provisioned and enclave instantiation validated using `nitro-cli` and vsock-based communication. However, while the enclave runtime was operable, the integration of the Nitro Enclaves C-SDK into mission-specific workloads, along with the need for custom JWT claim handling and `Vault`-compatible attestation logic, introduced significant complexity. Given time constraints and the early prototype scope, this path was deprioritised. Selected implementation details and limitations are revisited in the validation section.

Path B: SEV-SNP-Based Remote Attestation. Following the constraints encountered with Nitro, the deployment pivoted to AMD SEV-SNP, leveraging a Ubuntu-based EC2 instance in `eu-west-1` with SNP-enabled CPU configuration. The attesting instance generated JWT tokens using the `PyJWT.py` module, incorporating claims derived from the SEV-SNP attestation report via `snpguest`. A `Vault` instance deployed via `15_cc-vault-poc.yaml` validated these tokens against a configured role and released secrets based on successful integrity checks. The entire attestation pipeline, including `Vault` token issuance, policy binding and secret encryption, was fully integrated and validated within the prototype.

To operationalise the proposed prototype, a structured deployment methodology is applied to ensure reproducibility, traceability and mission-context alignment. The procedure focuses on deploying a minimal confidential key release architecture with integrated TEE-based attestation and policy-enforced key provisioning.

The deployment methodology initially followed a dual-path strategy, reflecting two distinct confidential computing approaches using AWS Nitro Enclaves (Path A) and AMD SEV-SNP (Path B). While Path A commenced first due to native integration advantages, it was halted upon encountering significant complexity in the initial SDK integration step. Consequently, Path B was introduced using SEV-SNP instances, proceeding successfully through enclave instantiation and attestation workflow implementation. The final stage of `Vault` deployment and configuration was completed jointly, integrating learnings from both paths.

The Nitro Enclave approach initially prioritised leveraging native AWS integration. Early phases including infrastructure provisioning, secure networking configuration and enclave instantiation using `nitro-cli` and `vsock`-based communication were completed. The path was halted during the initial integration phase of the Nitro Enclaves SDK, due to high complexity and resource demands.

1. **Environment Preparation (Completed):** Provisioned a Nitro-compatible EC2 instance, validated enclave instantiation with `nitro-cli` and established secure vsock communication. Configured isolated networking, TLS termination and role-based access aligned to Zero Trust principles.
2. **Attestation Document Generation (Implementation Halted):** Initiated integration of the Nitro Enclaves SDK for attestation document generation, halted due to complexity in custom claim parsing and enclave-specific handler development.
3. **Verifier Binding (Planned):** Intended definition of a verifier module for evaluating attestation documents against a known-good baseline was planned but deprioritised due to halted SDK integration efforts.
4. **Key Policy Enforcement (Planned):** Envisioned `Vault` policies enforcing enclave attestation claims, issuing JWT tokens for policy-driven key release.
5. **Test Secret Provisioning and Access (Planned):** Planned verification of policy-compliant secret handling within enclaves.
6. **Validation and Logging (Planned):** Anticipated validation of logs and attestation tokens ensuring traceable enclave verification under Zero Trust controls.

Pivoting after Path A was paused, Path B leveraged AMD SEV-SNP-enabled infrastructure and successfully established a full attestation workflow.

1. **Environment Preparation (Completed):** Provisioned SEV-SNP-enabled EC2 instances (e.g. `c6a`) within a secure networking environment, including robust TLS termination and role-based access control.
2. **Attestation Channel Setup (Completed):** Configured the `libsnpguest`-based attestation logic on Ubuntu instances to securely generate JWT tokens embedding attestation claims from the SNP attestation reports.
3. `Vault` **Deployment and Joint Configuration (Completed):** Deployed a centralised `HashiCorp Vault` instance using the stack `15_cc-vault-poc.yaml`, integrating learnings from Path A infrastructure setup. Configured the Transit Secrets Engine, audit logging, JWT authentication backend and attestation-specific policies to validate tokens issued from SEV-SNP environments, enabling conditional secret access.
4. **Key Policy Enforcement (Completed):** Established `Vault` policies enforcing strict constraints on attestation claims (e.g. PCR, enclave hash) verified during token validation.
5. **Test Secret Provisioning and Access (Completed):** Successfully validated conditional secret encryption, decryption and secure rotation operations through `Vault`'s transit API, strictly controlled by attestation status.
6. **Validation and Logging (Completed):** Implemented systematic logging and token validation workflows for replay protection and auditable traceability of enclave verification outcomes.

This deployment methodology accurately represents the integrated workflow, highlighting the shift from initial Nitro Enclave exploration to the fully realised SEV-SNP-based solution, culminating in a consolidated `Vault` deployment. Insights, challenges and key learnings are detailed in the validation section.

Deployment for the Prototype

This section outlines an extraction of the deployment process, focusing on the key steps and configurations that enabled the successful implementation of the remote attestation and key management prototype, based on the SEV-SNP architecture. The prototype deployment followed an automated and partially automated approach, leveraging AWS CloudFormation templates to provision foundational infrastructure components while manually configuring runtime-specific elements. This hybrid approach was chosen to balance flexibility, complexity management and iterative development agility in early prototyping phases.

Infrastructure Provisioning (Automated). Infrastructure provisioning was entirely automated via CloudFormation stacks `10_cc-secure-infra-attestation.yaml` and `15_cc-vault-poc.yaml`. The complete source code for these is provided in full detail in Annex 6.2. These listings serve as a comprehensive reference to facilitate reproducibility, transparency and detailed verification of the infrastructure provisioning process.
Specifically, the automation provided:

- `Secure Networking and Subnet Architecture`
  Excerpt of the automated provisioning of the foundational network architecture, including isolated subnets.

```
Resources:
  PrivateSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: 10.0.2.0/24
      AvailabilityZone: !Select [0, !GetAZs '']
      Tags:
        - Key: Name
          Value: !Sub "${ProjectName}-${Environment}-private-subnet"

  InternalSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupName: !Sub "${ProjectName}-${Environment}-internal-sg"
      GroupDescription: "Internal communication between attestation
      ↪  components"
      VpcId: !Ref VPC
      SecurityGroupEgress:
        # Outbound Internet for Updates
        - IpProtocol: "-1"
          CidrIp: "0.0.0.0/0"
          Description: "Outbound Internet"
      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-internal-sg"
```

The configuration of the `PrivateSubnet` and the associated `InternalSecurityGroup` within this template reflects essential principles of secure-by-design network segmentation and least privilege access control. The `PrivateSubnet` is defined with an explicitly scoped `CIDR` block `10.0.2.0/24` and availability zone selection using `!Select` and `!GetAZs`, enabling deterministic and scalable subnet placement across deployment regions. Tagging conventions using parameterised `${ProjectName}` and `${Environment}` variables promote environment-specific resource labelling for improved traceability and policy enforcement.

The `InternalSecurityGroup` is configured to restrict inbound traffic entirely and to permit outbound traffic only to `0.0.0.0/0`, thereby enforcing an egress-only pattern by default. This allows instances to pull updates or communicate externally without exposing internal services to unsolicited inbound access. The use of the tag "`Role: Internal SG`" supports automated role-based policies and dynamic security posture evaluation. Together, these definitions enforce baseline isolation for attested compute nodes while supporting controlled outbound interactions necessary for update mechanisms and trust bootstrapping.

- IAM roles, instance profiles and security group configurations
  Essential runtime permissions for attestation, cryptographic operations and secure instance management were provisioned via scoped IAM roles and associated instance profiles.

```
  Resources:
    EC2Role:
      Type: "AWS::IAM::Role"
      Properties:
        RoleName: !Sub "${ProjectName}-${Environment}-ec2-role"
        AssumeRolePolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: "Allow"
              Principal:
                Service: "ec2.amazonaws.com"
              Action: "sts:AssumeRole"
        ManagedPolicyArns:
          - "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        Policies:
          - PolicyName: "AttestationPermissions"
            PolicyDocument:
              Version: "2012-10-17"
              Statement:
                - Effect: "Allow"
                  Action:
                    - "kms:Decrypt"
                    - "kms:GenerateDataKey"
                    - "kms:CreateGrant"
                  Resource: !GetAtt AttestationKMSKey.Arn
                - Effect: "Allow"
                  Action:
                    - "ec2:CreateTags"
                  Resource: !Sub "arn:aws:ec2:${AWS::Region}:${AWS::AccountId}
                  ↪  :instance/*" # Allows tagging of EC2 instances
    EC2InstanceProfile:
      Type: "AWS::IAM::InstanceProfile"
      Properties:
        Roles:
          - !Ref EC2Role
```

The configuration of the `EC2Role` and its associated `InstanceProfile` establishes granular permission boundaries to enforce secure workload execution and access governance. The role is explicitly scoped to allow secure communication with the KMS, supporting decryption, data key generation and grant creation operations restricted to the attestation key resource. Additionally, permission to create EC2 tags ensures traceable resource labelling in alignment with audit and compliance requirements. The attachment of the `AmazonSSMManagedInstanceCore` managed policy facilitates integration with SSM for secure post-deployment management. Through parameterised naming and tightly scoped access policies, this configuration realises a mission-aligned Zero Trust posture across compute instances deployed for attestation workflows.

- Instance Deployment - OmniAware-EC2-SEV-SNP
  Initial system preparation, dependency management and installation of attestation components.

```
    Resources:
      SEVSNPLaunchTemplate:
        Type: "AWS::EC2::LaunchTemplate"
        Properties:
          LaunchTemplateData:
            InstanceType: "c6a.large"
            ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AmazonLinuxAMI]
            IamInstanceProfile:
              Name: !Ref EC2InstanceProfile
            KeyName: !Ref EC2KeyPair
            SecurityGroupIds:
              - !Ref InternalSecurityGroup
            CpuOptions:
              AmdSevSnp: "enabled"
            UserData:
              Fn::Base64: !Sub |
                #!/bin/bash
                set -e

                # Set hostname
                hostnamectl set-hostname OmniAware-EC2-SEV-SNP
                echo '127.0.0.1 OmniAware-EC2-SEV-SNP' >> /etc/hosts

                # Install development tools and dependencies
                dnf groupinstall -y "Development Tools"
                dnf install -y cmake git wget jq openssl-devel \
                  protobuf-compiler libtool autoconf automake \
                  kernel-headers kernel-devel awscli
                [...]

                # Install snpguest
                cd /opt
                git clone https://github.com/virtee/snpguest.git
                cd snpguest
                cargo build --release
                cp target/release/snpguest /usr/local/bin/

                # Install sevctl
                cd /opt
                git clone https://github.com/virtee/sevctl.git
                cd sevctl
                cargo build --release
                cp target/release/sevctl /usr/local/bin/

                # Vault CLI installation
                apt-get update -y
                apt-get install -y gnupg software-properties-common curl unzip
                curl -fsSL https://apt.releases.hashicorp.com/gpg | gpg --dearmor
                ↪ -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

                echo "deb
                ↪  [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
                ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main" |
                ↪  tee /etc/apt/sources.list.d/hashicorp.list

                # Vault Dependencies
                apt-get install -y python3-pip
```

```
54                apt-get install -y python3-full
55
56
57                # Set environment variable for Vault address
58                echo 'export VAULT_ADDR="http://<!Ref VaultInstancePrivateIP>"'
                  ↪  >> ~/.bashrc
59                [...]
60        SEVSNPInstance:
61          Type: "AWS::EC2::Instance"
62          Properties:
63            SubnetId: !Ref PrivateSubnet
64            LaunchTemplate:
65              LaunchTemplateId: !Ref SEVSNPLaunchTemplate
66              Version: !GetAtt SEVSNPLaunchTemplate.LatestVersionNumber
67            Tags:
68              - Key: "Name"
69                Value: !Sub "${ProjectName}-${Environment}-sev-snp"
70              - Key: "Role"
71                Value: "SEV-SNP-Attester"
```

The `SEVSNPLaunchTemplate` and its associated `SEVSNPInstance` define the automated provisioning and initial configuration of a confidential compute node within a SEV-SNP-enabled EC2 environment. The launch template encapsulates all essential bootstrapping logic via embedded `UserData`, ensuring deterministic preparation of the attestation runtime. This includes the installation of attestation-specific tools such as `snpguest` and `sevctl`, which are required to extract and validate SEV-SNP attestation reports directly from the CPU's firmware interface. `snpguest` interacts with the Secure Nested Paging Guest Request via Guest-Hypervisor Communication Block (GHCB) protocol to retrieve measurement data, while `sevctl` supports auxiliary verification and integration workflows. Their successful compilation requires low-level development toolchains, including `rust`, `cmake` and system headers, which are provisioned as part of the launch phase. The instance runs on AL2.

While this template enables end-to-end environment preparation, key limitations persist regarding full automation of the remote attestation flow with `HashiCorp Vault`. Specifically, the generation of JWT tokens, `Vault` policy definition and secret key release mechanisms must be executed outside the EC2 lifecycle — either manually or via dedicated post-deployment scripts. These constraints reflect a separation between system bootstrapping and trust policy enforcement, highlighting the need for orchestration layers beyond the scope of `UserData`-based initialisation.

A functionally equivalent and additional variant of this launch template based on Ubuntu 24.04 LTS is provided in Annex 6.2, showcasing interoperability across base operating systems while preserving the structural integrity of the provisioning workflow.

- `Instance Deployment - OmniAware-EC2-Vault`
  Automated `Vault` provisioning pipeline, including bootstrap, Transit Secret Engine setup, JWT auth configuration and policy-based Remote Attestation validation.

```
  Resources:
    VaultInstance:
      Type: AWS::EC2::Instance
      Properties:
        InstanceType: t3.micro
        ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", UbuntuAMI]
        KeyName:
          !ImportValue
            Fn::Sub: "${InfraStackName}-KeyPair-Name"
        SubnetId:
          !ImportValue
            Fn::Sub: "${InfraStackName}-PrivateSubnet-ID"
        SecurityGroupIds:
          - !ImportValue
              Fn::Sub: "${InfraStackName}-Internal-Security-Group-ID"
        IamInstanceProfile:
          !ImportValue
            Fn::Sub: "${InfraStackName}-InstanceProfile-Name"
        UserData:
          Fn::Base64: !Sub |
            #!/bin/bash
            set -e

            hostnamectl set-hostname OmniAware-EC2-Vault
            echo '127.0.0.1 OmniAware-EC2-Vault' >> /etc/hosts

            snap install aws-cli --classic
            apt-get update && apt-get install -y jq curl wget git cmake
            ↪  build-essential \
              linux-headers-$(uname -r) libssl-dev pkg-config autoconf automake
              ↪  libtool \
              protobuf-compiler libprotobuf-dev gnupg
              ↪  software-properties-common unzip

            curl -fsSL https://apt.releases.hashicorp.com/gpg | gpg --dearmor
            ↪  -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
            echo "deb
            ↪  [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
            ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main" |
            ↪  tee /etc/apt/sources.list.d/hashicorp.list
            apt-get update && apt-get install -y vault net-tools

            useradd --system --home /etc/vault.d --shell /usr/sbin/nologin vault
            mkdir -p /opt/vault/data /etc/vault.d
            chown -R vault:vault /opt/vault /etc/vault.d

            # Write Vault Config
            cat <<VAULTCFGEOF > /etc/vault.d/vault.hcl
            storage "file" {
              path = "/opt/vault/data"
            }

            listener "tcp" {
              address     = "0.0.0.0:8200"
              tls_disable = true
            }

```

```
51              api_addr = "http://127.0.0.1:8200"
52              cluster_addr = "https://127.0.0.1:8201"
53              ui = true
54              VAULTCFGEOF
55
56              # Write Systemd Unit File
57              cat <<VAULTUNITEOF > /etc/systemd/system/vault.service
58              [Unit]
59              Description=HashiCorp Vault - A tool for managing secrets
60              Documentation=https://www.vaultproject.io/docs/
61              Requires=network-online.target
62              After=network-online.target
63              ConditionFileNotEmpty=/etc/vault.d/vault.hcl
64
65              [Service]
66              User=vault
67              Group=vault
68              ExecStart=/usr/bin/vault server -config=/etc/vault.d/vault.hcl
69              Restart=on-failure
70
71              [Install]
72              WantedBy=multi-user.target
73              VAULTUNITEOF
74
75              systemctl daemon-reload
76              systemctl enable vault
77              systemctl start vault
78              sleep 10
79
80              export VAULT_ADDR="http://127.0.0.1:8200"
81              vault operator init -key-shares=1 -key-threshold=1 >
                ↪  /home/ubuntu/vault-keys.txt
82              UNSEAL_KEY=$(grep 'Unseal Key 1' /home/ubuntu/vault-keys.txt | awk
                ↪  '{print $NF}')
83              ROOT_TOKEN=$(grep 'Initial Root Token' /home/ubuntu/vault-keys.txt |
                ↪  awk '{print $NF}')
84              vault operator unseal "$UNSEAL_KEY"
85              vault login "$ROOT_TOKEN"
86              vault secrets enable transit
87              vault write -f transit/keys/attestation-test
88
89              # Create Transit Key for Attestation
90              cat <<POLICY > /tmp/attestation-policy.hcl
91              path "transit/encrypt/attestation-test" {
92                capabilities = ["update"]
93              }
94              path "transit/decrypt/attestation-test" {
95                capabilities = ["update"]
96              }
97              path "transit/keys/attestation-test" {
98                capabilities = ["read"]
99              }
100             POLICY
101
102             vault policy write attestation-policy /tmp/attestation-policy.hcl
103             echo "export VAULT_ADDR=http://127.0.0.1:8200" >>
                ↪  /home/ubuntu/.bashrc
104             echo "export VAULT_TOKEN=$ROOT_TOKEN" >> /home/ubuntu/.bashrc
105             chown ubuntu:ubuntu /home/ubuntu/vault-keys.txt
106             chmod 600 /home/ubuntu/vault-keys.txt
107
```

```
108                 # JWT Validation Setup
109                 vault auth enable jwt
110
111                 # Structure Setup
112                 mkdir -p /etc/vault.d/jwt
113           Tags:
114             - Key: "Name"
115               Value: !Sub "${ProjectName}-${Environment}-vault"
116             - Key: "Role"
117               Value: "Vault-Server"
```

The `VaultInstance` resource encapsulates the automated provisioning and configuration of a dedicated EC2 instance designated to host the `HashiCorp Vault` service as a critical enabler for remote attestation and key management within the platform architecture. Leveraging a hardened Ubuntu LTS base image, the instance is fully bootstrapped via embedded `UserData` logic that executes a multi-phase installation sequence, covering binary provisioning, runtime configuration, service orchestration and policy preloading.

The instance is deployed within an isolated private subnet and tightly coupled to privileged IAM instance profiles and restricted VPC security groups, imported dynamically via stack references. Core server behaviour is governed by three key configuration artefacts generated during instance initialisation:

- `/etc/vault.d/vault.hcl` defines the backend storage (`file`), local listener binding on port 8200 and operational parameters such as the API and cluster address;
- `/etc/systemd/system/vault.service` ensures resilient, system-managed startup of the `Vault` process with dependency ordering and restart policies;
- `/tmp/attestation-policy.hcl` encapsulates fine-grained access control logic for the `transit` engine and is applied immediately post-initialisation.

Upon first boot, `Vault` is automatically initialised using a one-share scheme, unsealed via extracted credentials and authenticated using the root token. Both the unseal key and root token are securely persisted in `/home/ubuntu/vault-keys.txt`, with restricted file permissions and ownership. These credentials are also exported into the user's environment for downstream access by follow-up automation scripts.

A deliberate architectural decision was made to omit HTTPS encryption on the `Vault` listener, despite its native support for TLS. This choice was informed by the desire to exclude certificate lifecycle complexity from the initial PoC. Introducing TLS would have necessitated additional operational workflows, such as certificate issuance, renewal scheduling, secret rotation and secure propagation across the infrastructure. While alternative solutions—including central certificate authorities, AWS Certificate Manager (ACM) or S3-based deployment—were considered, they were excluded in favour of a minimal, reproducible and agile deployment path. The trade-off is effectively mitigated by the deployment within a tightly controlled private subnet and may be reevaluated during future production hardening phases.

This design simplification also affects downstream cryptographic tooling. In particular, the `PyJWT.py` script — responsible for constructing SEV-SNP-backed JWT tokens for attestation — would require certificate validation under a strict HTTPS regime. This would imply the existence of a trusted public key infrastructure or synchronised certificate distribution process, which may conflict with the *air-gapped* or enclave-constrained nature of such deployments. By retaining an HTTP-only local context, the deployment achieves frictionless integration with attestation-capable workloads while maintaining extensibility for future TLS-based transitions.

Beyond basic setup, the template also automates activation of the `transit` secrets engine and the provisioning of a cryptographically scoped key labelled `attestation-test`, which is subject to the previously defined access policy. The `Vault` JWT authentication backend is enabled and structurally prepared for future integration steps, including JWKS URI

registration and claim-to-role mappings, which can be added manually or via follow-up IaC enhancements.

Vault Deployment Characteristics and Operational Constraints. The `vault` instance deployment encompasses EC2 instance initialisation, installation of required dependencies, `Vault` installation and essential bootstrapping tasks, such as operator unseal, root token export and activation of the Transit Secrets Engine. Although this automated sequence substantially alleviates the operational burden of manual provisioning, several runtime-critical components necessitate explicit post-deployment configuration. This is primarily due to `Vault`'s inherently stateful design and the use of the open-source version, which lacks support for persisted dynamic runtime artefacts. In contrast to the enterprise-grade edition, *Vault OSS* does not retain authentication roles, JWT validation configurations or OIDC-related metadata across restarts. Consequently, runtime artefacts such as JWT roles and their associated cryptographic validation logic must be re-established manually after each service restart or infrastructure redeployment.

Runtime Configuration (Partially Automated). While the `vault` instance is automatically deployed and bootstrapped, additional runtime steps were required to ensure secure integration with the SEV-SNP guest. Due to the stateless design of the open-source `Vault` version and the absence of Enterprise-grade configuration persistence, dynamic elements such as JWT auth roles and validation logic could not be retained across reboots.

Moreover, HTTPS was deliberately omitted to avoid operational complexity associated with certificate management, including TLS provisioning and trust chain validation. Enabling HTTPS would have required either integrating a certificate authority service or distributing signed certificates via external channels (e.g. S3), both of which would have introduced coupling and deployment overhead.

To maintain a reproducible and lightweight prototype environment, these elements were instead configured manually post-deployment. The manual configuration included:

- Re-establishing `JWT` authentication backends
- Uploading and parsing SEV-SNP JWT claims
- Binding `Vault` policies to attested claims
- Verifying the Transit Secrets Engine integration

To maintain architectural simplicity and maximise reproducibility during early-phase development, these design trade-offs were consciously accepted. Consequently, the following manual runtime steps were performed:

- `JWT Authentication Backend Configuration - OmniAware-EC2-Vault`
  The JWT authentication backend must be re-enabled and configured with the appropriate public key and issuer details. This includes providing the `public.pem` key file and defining the expected `bound_issuer` string used for SEV-SNP attestation tokens.

```
vault auth enable jwt
vault write auth/jwt/config \
  jwt_validation_pubkeys=@/etc/vault.d/public.pem \
  bound_issuer="sev-snp"
```

In order to validate SEV-SNP-based JWT tokens, the `Vault` authentication backend must be explicitly enabled and configured with the correct cryptographic material. This includes the public key that corresponds to the attestation report signature, which must first be generated and stored in an accessible location on the `Vault` instance. A typical key generation command is:

```
openssl genrsa -out sev-snp-key.pem 2048
openssl rsa -in sev-snp-key.pem -pubout -out /etc/vault.d/public.pem
```

This public key file is referenced in the `Vault jwt/config` path to establish a trusted source for bound issuers and claims validation.

The public key used for JWT validation in `Vault` must correspond to the private key on the SEV-SNP instance that generates the attestation token. This ensures cryptographic verifiability of the enclave identity and its associated claims. To obtain the public key from the SEV-SNP instance, several transport mechanisms can be employed depending on the security posture and automation level. In development settings, a secure copy can be performed manually using `scp`, e.g.:

```
scp /home/ubuntu/public.pem vault-admin@vault-instance:/etc/vault.d/public.pem
```

In operational contexts leveraging AMD SEV-SNP, the transfer of the public verification key from the attester node to the `Vault` instance can be fully automated as part of a secure bootstrapping pipeline. The public key, which is typically derived from a pre-generated asymmetric key pair within the confidential guest, must be made accessible to `Vault` in order to verify the integrity and authenticity of signed attestation tokens.

To avoid insecure transport mechanisms or manual intervention, this process can be integrated with AWS Systems Manager (SSM) capabilities, which allow direct and secure injection of the enclave's public key into the `Vault` host. This ensures cryptographic alignment between the SEV-SNP attestation signer and the `Vault` JWT verifier, establishing a consistent trust chain. The automated handover of the public key strengthens the integrity of the Remote Attestation workflow, reduces the operational attack surface and supports reproducibility in high-assurance environments.

The entire process must be tightly controlled and logged, particularly in regulated defence contexts, to fulfil auditability and traceability requirements. In future iterations, a CI/CD pipeline may facilitate automatic provisioning of enclave-generated keys, further reducing manual intervention and increasing deployment security.

- `Role Recreation Post-Restart - OmniAware-EC2-Vault`
  Since `Vault` does not persist JWT roles across restarts unless explicitly stored in an external backend, the attestation role must be recreated manually to bind claims (such as `sub`, `aud`) to specific token policies.

```
vault write auth/jwt/role/sev-snp-role \
  role_type="jwt" \
  user_claim="sub" \
  bound_subject="attester-001" \
  bound_audiences="vault" \
  token_policies="attestation-policy" \
  ttl="1h"
```

Likewise, the presence of cryptographic keys within the Transit Engine should be verified to ensure subsequent encryption requests do not fail due to missing or misconfigured key entries. This validation step is particularly important for attestation-based workloads, where runtime encryption depends on a correctly registered key. The example shown issues a write operation to create or verify the `attestation-test` key within the Transit namespace.

  - `user_claim` specifies which JWT claim (e.g. `sub`) is to be used as the identity anchor.
  - `bound_subject` enforces that only JWTs with a specific subject value (e.g. `attester-001`) are accepted.
  - `bound_audiences` restricts tokens to a predefined audience (e.g. `vault`).
  - `token_policies` assigns the `Vault` policy (e.g. `attestation-policy`) to be granted upon successful authentication.
  - `ttl` sets the time-to-live for the issued token.

These attributes ensure that only authorised and attested entities — typically confidential workloads running inside enclaves or confidential VMs — can access `Vault`'s capabilities.

- `Transit Secrets Engine Verification - OmniAware-EC2-Vault`
  Although the Transit engine is enabled at deployment, it is recommended to verify its availability and ensure the presence of the designated key.

```
vault secrets enable -path=transit transit
vault write -f transit/keys/attestation-test
```

The verification of the `transit` secrets engine ensures that `Vault` is not only capable of handling cryptographic operations, but also correctly initialised to manage designated keys for remote attestation scenarios. While the `transit` backend is enabled automatically during provisioning, the explicit creation of a namespaced key (`attestation-test`) serves as a validation checkpoint and a functional prerequisite for subsequent encryption and decryption operations tied to enclave-based workflows.

The invocation of

```
vault secrets enable -path=transit transit
```

re-establishes the backend (if necessary), while the command

```
vault write -f transit/keys/attestation-test
```

provisions the required key under the defined path. This key becomes the central entity for all encryption, decryption and signing processes that are triggered by enclave verifiers or policy-enforced routines during runtime. In a production setup, key configuration options such as `derived`, `exportable` or `auto_rotate` could be used to tailor lifecycle behaviour and compliance characteristics.

- `SEV-SNP Remote Attestation Script Configuration - OmniAware-EC2-SEV-SNP`
  Shell script to regenerate the SEV-SNP attestation report and interact with `Vault`.

```
#!/bin/bash
export VAULT_ADDR=<OmniAware-EC2-Vault-IP>:8200
export VAULT_SKIP_VERIFY=true

cd /opt/snpguest-test/

snpguest report /tmp/guest_report.bin /tmp/request.txt --random \
  && base64 /tmp/guest_report.bin > /tmp/guest_report.b64

python3 /opt/snpguest-test/PyJWT.py > jwt.txt
export JWT_TOKEN=$(cat jwt.txt)

export VAULT_TOKEN=$(curl -sk --request POST \
  --url "$VAULT_ADDR/v1/auth/jwt/login" \
  --header "Content-Type: application/json" \
  --data "{\"jwt\": \"$JWT_TOKEN\", \"role\": \"sev-snp-role\"}" \
  | jq -r '.auth.client_token')

vault token lookup
```

The above shell script implements the final runtime logic required to initialise and complete the remote attestation flow from an SEV-SNP-enabled confidential compute node. After navigating into the attestation working directory, the `snpguest report` command is used to generate a signed SEV-SNP report, which is subsequently base64-encoded and embedded into a signed JWT using a local Python utility (`PyJWT.py`). This token, containing cryptographically bound metadata about the confidential guest (e.g. report, `sub`, `aud`, `iss`, and expiration time), is then submitted to the `Vault` instance for verification via the JWT authentication backend.

Upon successful login, `Vault` returns a short-lived session token scoped to the previously configured policy (`attestation-policy`). The variable `VAULT_TOKEN` captures this result and enables further authenticated interaction with `Vault`, such as requesting or decrypting secrets through the `transit` engine. The entire process demonstrates a secure and reproducible attestation workflow on runtime level and is intended for direct integration in post-bootstrapping automation pipelines on the enclave host.

The script assumes the existence of a valid `Vault` configuration accepting JWTs signed with the same private key used by the `PyJWT.py` script. For completeness, the JWT construction logic is shown below.

- `PyJWT.py - OmniAware-EC2-SEV-SNP`
  Minimal Python tool to generate a signed SEV-SNP attestation JWT.

```
import jwt
from datetime import datetime, timedelta, timezone

private_key = open("private.key", "r").read()
payload = {
    "sub": "attester-001",
    "aud": "vault",
    "iss": "sev-snp",
    "nonce": "abc123",
    "iat": datetime.now(timezone.utc),
    "exp": datetime.now(timezone.utc) + timedelta(minutes=5),
    "report": open("/tmp/guest_report.b64", "rb").read().hex()
}
token = jwt.encode(payload, private_key, algorithm="RS256")
print(token)
```

This utility script constructs a standards-compliant JWT for SEV-SNP attestation based on the `RS256` algorithm. The payload includes semantic metadata required by the `Vault` verifier, such as the enclave's subject identifier (`sub`), the designated audience (`aud`), the issuer label (`iss`) and the embedded base64-encoded SEV-SNP attestation report. Additionally, the token includes a nonce and expiration time to ensure replay resistance and short-lived trust anchors. The resulting token is printed to `stdout` and subsequently used for `Vault` login.

For hardened environments, the private key should reside within a secure enclave, HSM or transient filesystem and ideally be rotated or generated per workload. Production-grade setups may also include derived claims and hardware-bound attestation metadata.

- `Vault Audit Logging Activation - OmniAware-EC2-Vault`
  `Vault`'s audit subsystem was programmatically activated to ensure complete traceability and post-deployment introspection of sensitive operations such as secret reads, token generation or policy application. The following configuration was embedded into the deployment pipeline to persist audit logs locally:

```
vault audit enable file file_path=/var/log/vault/audit.log
```

This configuration ensures that all authenticated `Vault` operations are recorded in structured log files, including metadata such as requestor identity, token scopes and timestamped action trails. The audit logs are locally persisted on the EC2 host and can be accessed post-deployment using secure remote access mechanisms (e.g. `aws ssm start-session`). The logs are serialised in JSON and support automated parsing via tools such as `jq`, as demonstrated below:

```
cat /var/log/vault/audit.log | jq
```

To maintain separation of duties and support forensic readiness in regulated deployments, the audit logs were configured for restricted access, ensuring they can only be read by authorised operational personnel or collected via external compliance agents. Optional extensions include forwarding to centralised log aggregation services (e.g. CloudWatch Logs or SIEM backends), which were intentionally omitted to minimise PoC complexity.

The deployment approach implemented in the prototype followed a deliberately pragmatic methodology, combining automated infrastructure provisioning with partially automated runtime configuration steps. While the foundational components — including network architecture, EC2 instance provisioning and essential IAM permissions — were provisioned through reproducible IaC templates, several runtime-specific configurations, such as `Vault` role definitions, JWT public key handling and trust policy bindings, were executed manually or via shell scripts external to the template pipeline.

This hybrid approach enabled controlled experimentation with critical security components and allowed for iterative validation of system behaviour under realistic operational conditions. Particularly in early prototyping stages, the selective use of manual steps was instrumental in achieving visibility, traceability and fine-grained control over individual configuration states.

However, the reliance on partially automated procedures also introduced limitations in terms of repeatability, audit assurance and error tolerance. Manual post-deployment actions — especially those related to security policy enforcement — are inherently prone to human error and complicate validation in regulated or scaled environments.

From a methodological perspective, this highlights the trade-off between architectural flexibility and operational reproducibility. Future iterations may revisit the degree of automation depending on the maturity of the system, the assurance requirements of the target environment and the available organisational support for continuous configuration governance.

In conclusion, while the implemented deployment strategy effectively demonstrated the feasibility and technical soundness of the proposed architecture, its partially automated character should be viewed as an interim solution. A progressive transition towards declarative and fully automated runtime configuration — aligned with the foundational automation layers — is recommended to minimise operational risk and increase maintainability for production-grade deployments.

4.3 INTERFACES

Summary: This section presents the interface design and implementation strategy for secure, schema-compliant data ingestion within the *OmniAware* platform. A lightweight API gateway setup, tailored for the PHM scenario, was implemented in alignment with cloud-native IaC practices and foundational zero-trust principles. The prototype enables structured JSON-based schema validation, modular exposure of ingestion endpoints and introduces the architectural groundwork for identity-bound access control using JWTs — although integration of `Vault`-based token validation remained out of scope.

The design was inspired by NGVA interface principles, allowing future extensibility towards coalition-compliant deployments without introducing platform dependencies. By maintaining a minimal and testable service footprint, the approach supports rapid prototyping and schema evolution while remaining compatible with high-assurance, federated environments. This establishes a hardened, policy-enforced ingress layer as a foundational enabler for sovereign and mission-oriented multi-cloud architectures.

4.3.1 *Data Flow Design and Deployment Strategy*

To illustrate the ingestion interface's role within the broader system, its architectural integration is depicted in Figure 3.9. The diagram outlines the secure data flow from external producers through the API Gateway, schema-based validation and modular forwarding mechanisms towards downstream consumers. While full identity-bound access control via JWTs and policy enforcement is not yet active, the architectural layout anticipates their integration. This visual framing positions the ingestion interface as a future trust-enforcing boundary between untrusted data sources and mission-grade, policy-aware cloud-native services — aligned with zero-trust design principles.

A structured design approach is required that accommodates both real-time operational requirements and long-term standardisation goals to enable secure and interoperable data ingestion in the *OmniAware* platform. This section introduces the ingestion interface architecture developed for the Platform Health Monitoring (PHM) scenario. It represents the first step towards a federated, extensible and policy-enforced data flow architecture, aligned with zero trust and Confidential Computing principles.

The ingestion pipeline follows a cloud-native IaC-based approach using AWS primitives and integrates authentication, encryption, validation and policy enforcement as foundational control points. By embedding JWT-enabled identity propagation and attestation token validation into the interface logic, the design anticipates secure multi-party data exchange even in untrusted or coalition-operated environments.

Importantly, the design also draws methodological inspiration from the NATO Generic Vehicle Architecture (NGVA), which promotes modularity, standardised interfaces and decoupled sensor integration in land-based tactical systems. While NGVA specifications were not implemented in full, the core design philosophy — namely a separation of concerns between data producers, ingress endpoints and secure compute domains — has been adopted and refined for cloud-native deployments. This allows seamless extension to future NGVA-compliant systems without architectural rework.

While STANAG 4754 [7] — which formalises architectural patterns for tactical sensor systems — is not yet fully implemented in the PoC stack, it provides a valuable guideline for future extensions and is referenced in the validation context for interface compliance testing. In particular, the API Gateway developed in this section can later be mapped against the interface validation checkpoints outlined in Volume VI of the standard.

Federated Readiness and Coalition Interoperability. To support federated deployment scenarios, the ingest interface must remain agnostic to underlying account boundaries, enabling schema-aligned validation and token-based access control across distinct security domains. While these goals were not fully implemented, foundational design principles — such as decoupling

the interface logic from consumer-specific processing workflows and planning for policy-bound identity enforcement at the point of ingress — were introduced. This paves the way for integrating trust enforcement strategies across federated or coalition-operated environments.

In summary, this section presents the initial ingestion service blueprint, implemented as a parameterised and deployable CloudFormation template. The interface serves as the foundation for secure telemetry flow, audit logging and subsequent confidential processing and is referenced in multiple architecture views including NSOV-2 and NSOV-3. The implementation is intentionally kept modular and minimal to support rapid iteration and testing under PoC conditions, while offering a migration path to production-grade deployments under alliance-aligned governance.

4.3.2 *Deployment and Automation*

As an excerpt of the full deployment stack, the listing is designed to illustrate foundational aspects such as identity-based request validation, encrypted logging and controlled integration of telemetry schemas.

To ensure reproducibility and compatibility with existing infrastructure-as-code practices, the deployment follows the `AWS::CloudFormation` format and implements a minimal, yet functional service definition tailored to PoC conditions. This includes a secure API Gateway endpoint, schema validation for incoming telemetry data and integration with IAM and KMS-based audit controls.

The excerpt is deployed within the dedicated AWS account of the *AWS Guild Germany*, allowing for controlled and isolated experimentation. This facilitates the implementation of advanced features such as the extension of API Gateway endpoints with `Vault`-based key provisioning and *Remote Attestation* mechanisms, without impacting production-like environments. In this way, a simplified deployment version demonstrates essential functions while enabling low-risk innovation.

By referencing only essential components, the excerpt remains readable and adaptable. It serves as a practical template to implement more advanced ingestion pipelines (e.g. classified ingestion, dynamic source partitioning, etc.) in later development stages. In optimised deployment methodologies, mirroring of production environments could be considered — provided the security and resource constraints justify such efforts.

To support alignment with NGVA-aligned architectures, the section also discusses the use of `JSON Schema` and outlines future integration options for NGVA data models and validation.

20_secure-ingest-api.yaml - Excerpt

Secure Ingest API for telemetry and sensor data.

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: >
  Secure Ingest API Stack for AWS Guild Account -
  reduced to essential components for Proof of Concept and experimental
  ↪ Confidential Computing setup.

Parameters:
  Application:
    Type: String
    Default: OmniAware
  Stage:
    Type: String
    Default: dev
  Region:
    Type: String
    Default: eu-west-1
  VpcId:
    Type: String
    Description: The ID of the VPC to deploy the API into

Resources:
  IngestRestApi:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: !Sub "${Application}-${Stage}-IngestApi"
      Description: Private API for telemetry data ingestion (PoC)
      EndpointConfiguration:
        Types:
          - REGIONAL

  TelemetryResource:
    Type: AWS::ApiGateway::Resource
    Properties:
      RestApiId: !Ref IngestRestApi
      ParentId: !GetAtt IngestRestApi.RootResourceId
      PathPart: telemetry

  TelemetryModel:
    Type: AWS::ApiGateway::Model
    Properties:
      RestApiId: !Ref IngestRestApi
      ContentType: application/json
      Name: TelemetryDataModel
      Schema:
        "$schema": "http://json-schema.org/draft-04/schema#"
        type: object
        properties:
          timestamp:
            type: string
          payload:
            type: object
```

The presented excerpt outlines the CloudFormation-based provisioning of a simplified and isolated telemetry ingestion interface. It defines a minimal `AWS::ApiGateway::RestApi` stack, including endpoint resources and schema validation for sensor data via API Gateway models. Designed for experimental deployments within the AWS account of the *AWS Guild Germany*, the configuration intentionally omits integration with downstream consumers or upstream routing services. It serves as a blueprint for testing Confidential Computing-related extensions, such as

Vault-based attestation workflows and encryption validation via KMS. Core components such as tightly scoped IAM roles, minimal telemetry models and hardcoded configuration values make this a lightweight yet functional prototype, suitable for rapid iteration and future adaptation towards NGVA-aligned architectures.

Automation and Limitations. While the CloudFormation template automates key configuration steps for the API Gateway layer, including request validation and IAM-controlled execution, the ingestion backend remains abstracted. Full automation of downstream services — such as telemetry transformation, metadata enrichment or multi-channel delivery — would require additional `AWS::Lambda`, `AWS::Firehose` or `AWS::StepFunctions` resources. These are present in the full PHM reference architecture (cf. Fig. 3.9) but excluded from this excerpt for clarity and modularity. Furthermore, certificate management and policy-based JWT validation are intentionally left out in favour of testability.

Schema Validation and NGVA Compatibility. The deployment includes a request model definition using `JSON Schema` (Draft-04) [3], which is enforced by an `AWS::ApiGateway::Model` component and validated by a `RequestValidator`. `JSON Schema` provides a structured and interoperable means of enforcing data format compliance at runtime, allowing immediate rejection of malformed telemetry data and preventing downstream processing errors. The corresponding `JSON` schema, supporting test bed configuration and payload validation, is provided in the appendix for reference (cf. Appendix 6.2).

Future iterations of this deployment could integrate NGVA-aligned schemas by mapping incoming telemetry payloads to validated NGVA message types. In this context, the `TelemetryDataModel` may be extended to include additional metadata fields (e.g. platform ID, message type, encryption flags) or enforce inheritance constraints defined in a shared schema registry. Such evolution would allow direct integration of coalition data models while maintaining Zero Trust enforcement at the ingestion layer.
The full template draft is available in the appendix and can be reused for confidential ingestion pipelines in sovereign or coalition deployments.

Despite the architectural groundwork and modular service design, the realisation of a hardened and attestation-aware API Gateway for secure data ingestion remained out of scope due to time constraints. While foundational components such as `Vault` integration, token-based authorisation and schema enforcement were implemented, the envisioned extensions for deep remote attestation workflows — including runtime enclave verification and dynamic trust propagation — could not be completed within the timeframe of this thesis. Nevertheless, the present implementation establishes a starting point for future iterations, offering a validated deployment scaffold upon which production-grade and zero-trust compliant ingestion flows can be built.

4.4 VALIDATION

Summary: This section presents the validation of core components along the two attestation and ingestion paths introduced in the deployment chapter. The primary objectives were to test the partial implementation of remote attestation flows (**Path A** with Nitro Enclaves and **Path B** with SEV-SNP) as well as to validate secure ingestion mechanisms via a dedicated API Gateway. Each setup was verified through targeted test executions, runtime logs and manual inspection of cryptographic and functional outputs.

Path A validated the readiness of the Nitro-based TEE infrastructure and confirmed enclave runtime compatibility and `vsock`-based container execution. While `Vault` integration was intentionally excluded, this baseline validation established a solid foundation for future iterations.

Path B, by contrast, achieved a full end-to-end attestation flow leveraging SEV-SNP guest measurement reports. The reports were cryptographically signed, converted into a JWT and successfully submitted to `Vault` for authentication and attestation-bound decryption. Key con-

figuration, audit logs and manual token validation confirmed policy-matched secret handling and strict Zero Trust enforcement.

Additionally, the ingestion interface was evaluated using realistic NGVA-compliant data, simulating both structured telemetry and unstructured image payloads. A functionally extended variant of the Secure Ingestion Gateway — deployed under the GIT AWS ingest account — enabled early validation of production-adjacent ingestion flows. For telemetry, successful schema validation, Firehose handoff and DynamoDB persistence were demonstrated. For images, visual classification and metadata extraction were partially successful; however, schema constraints prevented full pipeline completion due to incomplete detections. This behaviour was consistent with design expectations and highlights the importance of stable inference output and fallback schema handling in future iterations.

Together, these results confirm the technical feasibility of Confidential Computing and Zero Trust enforcement using SEV-SNP attestation within cloud environments. The ingestion pipeline further demonstrated the ability to process multimodal mission inputs under realistic constraints, validating the practical applicability of the proposed platform architecture.

4.4.1 *Path A: Nitro Enclave Evaluation*

The first evaluation path focused on the deployment and runtime validation of a Nitro TEE-based execution environment. As illustrated in Figure 4.2, a minimal container workload was successfully built and transformed into a Nitro-compatible enclave image (EIF) using the Nitro CLI.

```
root@OmniAware-EC2-Nitro-Enclave:/opt/enclave-example$ docker build /usr/share/nitro_enclaves/examples/hello -t hello
          docker image ls
          nitro-cli build-enclave --docker-uri hello:latest --output-file hello.eif
[+] Building 1.5s (7/7) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                      0.0s
 => => transferring dockerfile: 307B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/busybox:latest                                                         1.0s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                           0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 307B                                                                                         0.0s
 => [1/2] FROM docker.io/library/busybox:latest@sha256:f85340bf132ae937d2c2a763b8335c9bab35d6e8293f70f606b9c6178d84f4  0.3s
 => => resolve docker.io/library/busybox:latest@sha256:f85340bf132ae937d2c2a763b8335c9bab35d6e8293f70f606b9c6178d84f4  0.0s
 => => sha256:f85340bf132ae937d2c2a763b8335c9bab35d6e8293f70f606b9c6178d84f42b 10.20kB / 10.20kB                          0.0s
 => => sha256:7c0ffe5751238c8479f952f3fbc3b719d47bccac0e9bf0a21c77a27cba9ef12d 610B / 610B                                0.0s
 => => sha256:6d3e4188a38af91b0c1577b9e88c53368926b2fe0e1fb985d6e8a70040520c4d 459B / 459B                                0.0s
 => => sha256:90b9666d4aed1893ff122f238948dfd5e8efdcf6c444fe92371ea0f01750bf8c 2.15MB / 2.15MB                            0.2s
 => => extracting sha256:90b9666d4aed1893ff122f238948dfd5e8efdcf6c444fe92371ea0f01750bf8c                                  0.1s
 => [2/2] COPY hello.sh /bin/hello.sh                                                                                     0.0s
 => exporting to image                                                                                                    0.0s
 => => exporting layers                                                                                                   0.0s
 => => writing image sha256:84f6a07fc4c2e3a134caa81831f6c61e30bc253e311c11a03d3e2a43c2c5d9b5                              0.0s
 => => naming to docker.io/library/hello                                                                                  0.0s
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
hello         latest    84f6a07fc4c2   Less than a second ago   4.28MB
amazonlinux   2         f3b2a1a8c945   6 days ago               166MB
Start building the Enclave Image...
Using the locally available Docker image...
Enclave Image successfully created.
{
  "Measurements": {
    "HashAlgorithm": "Sha384 { ... }",
    "PCR0": "0517899efb06ffeb3ffb96fbd9d4e40a3aab85ce656c723c90ff32e1e496c5d8ac5a70ba6a7b658850c7a9ac073b005c",
    "PCR1": "4b4d5b3661b3efc12920900c80e126e4ce783c522de6c02a2a5bf7af3a2b9327b86776f188e4be1c1c404a129dbda493",
    "PCR2": "81fbc850edc614d6055d0ab2f5a8a4b408fa329a820f0f5c108648ea4e3873cc77c1e944cbb3a8b3438b983d3a3502fa"
  }
}
root@OmniAware-EC2-Nitro-Enclave:/opt/enclave-example$
```

Figure 4.2: OmniAware-EC2-Nitro-Enclave - Building a Nitro Enclave-compatible Container (EIF) using Nitro CLI

This step verified the correct conversion from Docker to enclave image format and the compatibility of the Ubuntu-based example with the Nitro execution environment. Subsequently, the enclave image was launched using vsock-enabled runtime parameters. Figure 4.3 shows the successful instantiation of the enclave, confirming vsock runtime compatibility and container isolation.

While this prototype did not include integration with the `Vault` attestation pipeline, it validated the fundamental readiness of the Nitro-based SDK environment to support future remote attestation workflows. In particular, the correct operation of the vsock channel, container execution and enclave runtime isolation provides a reliable baseline for building attestation-capable service logic in subsequent iterations.

Figure 4.3: OmniAware-EC2-Nitro-Enclave - Launching the Nitro Enclave Runtime with vsock-enabled Parameters

During the deployment and validation process, several technical characteristics of the Nitro architecture emerged as practically relevant. One key insight was the critical importance of accurate resource allocation, particularly regarding `-memory` and `-cpu-count` flags, which must be set explicitly to match the enclave's runtime requirements. The Nitro CLI enforces these limits strictly and insufficient memory allocation may silently prevent successful enclave startup. Even in the minimalistic "Hello" example provided by the official SDK, tuning these parameters was required to ensure successful enclave execution and vsock-based communication.

Furthermore, it was noted that Nitro Enclaves are available in most AWS regions globally, making them operationally more accessible than SEV-SNP, which is currently restricted to select instance families in limited data centres. This broader availability may be advantageous for multinational and distributed coalition deployments, where geographic flexibility and data residency constraints apply.

Unlike SEV-SNP, Nitro Enclaves demand that container workloads be pre-transformed into a Nitro-compatible EIF image format. This process requires tight integration with the Nitro SDK and implies an additional packaging step in the development workflow. The resulting enclaves operate in a tightly constrained environment without network access, necessitating explicit implementation of secure vsock channels and tailored runtime environments.

While the SDK provides primitives for implementing remote attestation flows — including JWT claim verification, vsock forwarding and attested key exchange — the complexity of integrating these components proved non-trivial. The engineering effort required to implement the SDK-based remote attestation pipeline, including `vsock-proxy`, enclave-side server logic and policy-bound key handling, was deemed disproportionately high in comparison to the overarching thesis objective of validating SEV-SNP-based confidential computing. As a result, Path A was scoped to runtime validation only, deliberately omitting SDK integration. This decision ensured architectural focus and effort alignment while still yielding a technically valid baseline for enclave execution readiness.

4.4.2 *Path B: SEV-SNP and `Vault` Runtime Validation*

The validation pipeline referred to as **Path B** aimed to verify the feasibility of deploying and attesting a dedicated SEV-SNP guest instance on an EC2 node within AMD-powered infrastructure. This scenario supports the broader PoC objective of enabling hardware-backed confidential computing and attestation-based access control. As shown in Figure 4.4, the system kernel reported active support for SEV, SEV-ES and SEV-SNP, thus confirming the foundational prerequisites for memory encryption.

To validate the hardware trust anchor, the `sevctl` tool was executed (Figure 4.5) to illustrate the expected platform capabilities in checklist format. While this tool provides valuable insight into the support status of SEV-SNP features, it is primarily intended for execution on host systems (i.e. hypervisors) and not guest VMs, where full green-pass results are not guaranteed due to virtualisation layer constraints.

As seen in the output, basic SEV features were marked as supported, but advanced SNP functionalities failed due to missing kernel module parameters (e.g. `kvm_amd.sev=1`) and un-

```
root@OmniAware-EC2-SEV-SNP:/usr/bin$ dmesg | grep -i sev
[    0.652292] Memory Encryption Features active: AMD SEV SEV-ES SEV-SNP
[    0.880178] SEV: Using SNP CPUID table, 64 entries present.
[    1.411038] SEV: SNP guest platform device initialized.
[    6.173181] systemd[1]: Hostname set to <OmniAware-EC2-SEV-SNP>.
[    8.920888] sev-guest sev-guest: Initialized SEV guest driver (using vmpck_id 0)
```

Figure 4.4: OmniAware-EC2-SEV-SNP - Kernel Confirmation of SEV-SNP Activation: Verified via `dmesg`, the guest kernel reports SEV, SEV-ES and SEV-SNP support.

available device nodes such as `/dev/sev`. These limitations are known side effects of restricted passthrough capabilities in guest environments and do not inherently indicate the absence of hardware-level SNP support. For this reason, the use of `snpguest` is generally recommended when assessing attestation capabilities from within a VM.

Nevertheless, `sevctl` remains a valuable tool for demonstrative purposes, as it highlights the status of key SNP components in a compact and verifiable format.

```
root@OmniAware-EC2-SEV-SNP:/usr/bin$ sevctl ok
[ PASS ] - AMD CPU
[ PASS ]   - Microcode support
[ FAIL ]   - Secure Memory Encryption (SME)
[ PASS ]   - Secure Encrypted Virtualization (SEV)
[ FAIL ]     - Encrypted State (SEV-ES)
[ FAIL ]     - Secure Nested Paging (SEV-SNP)
[ SKIP ]       - VM Permission Levels
[ SKIP ]         - Number of VMPLs
[ PASS ]     - Physical address bit reduction: 0
[ PASS ]     - C-bit location: 51
[ PASS ]     - Number of encrypted guests supported simultaneously: 0
[ PASS ]     - Minimum ASID value for SEV-enabled, SEV-ES disabled guest: 0
[ FAIL ]     - SEV enabled in KVM: Error - /sys/module/kvm_amd/parameters/sev does not exist
[ FAIL ]     - SEV-ES enabled in KVM: Error - /sys/module/kvm_amd/parameters/sev_es does not exist
[ FAIL ]     - Reading /dev/sev: /dev/sev not readable: No such file or directory (os error 2)
[ FAIL ]     - Writing /dev/sev: /dev/sev not writable: No such file or directory (os error 2)
[ PASS ]   - Page flush MSR: DISABLED
[ FAIL ] - KVM supported: Error reading /dev/kvm: (No such file or directory (os error 2))
[ PASS ] - Memlock resource limit: Soft: 18446744073709551615 | Hard: 18446744073709551615
```

Figure 4.5: OmniAware-EC2-SEV-SNP - SEV-SNP Capability Check via `sevctl`: Expected support for SEV was confirmed; however, full SNP functionality was unavailable due to missing kernel-level integration.

Despite these limitations, the guest instance successfully executed the `snpguest` binary to generate an attestation report (Figure 4.6), confirming the ability to produce cryptographically signed TCB measurements and platform metadata via the Secure Processor (SP).

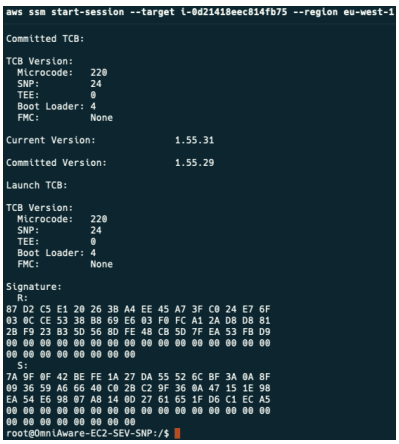


Figure 4.6: OmniAware-EC2-SEV-SNP - Successful Attestation Report via `snpguest`: TCB hashes and platform metadata were generated and cryptographically signed by the PSP.

For illustrative purposes, Figures 4.7 and 4.8 provide excerpts from the resulting attestation payload, showcasing policy flags, firmware identifiers and virtual machine privilege levels (VMPL) relevant to runtime security validation.

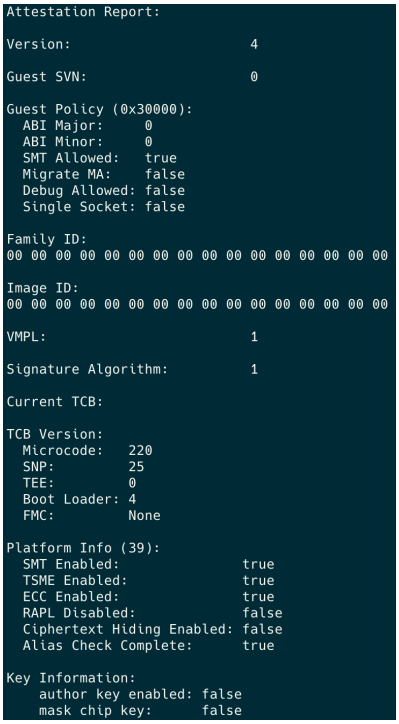


Figure 4.7: OmniAware-EC2-SEV-SNP - Attestation Report: Header and Guest Policy Values: SNP version, guest operating mode, VMPL and measurement configuration.

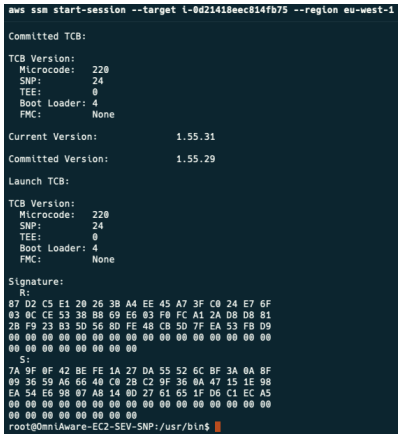


Figure 4.8: OmniAware-EC2-SEV-SNP - Attestation Report: TCB Measurements and Platform Info: Platform firmware version, TCB identifiers and flags indicating supported features such as SMT, migration and debug visibility.

The attestation payload was subsequently transformed into a JWT using a custom `PyJWT` script and employed in policy-bound interactions with the `Vault transit` secret engine, as described in Chapter 4.2. This demonstrated successful enclave-to-host trust propagation using signed SNP measurements.

Path B confirms that SEV-SNP can be operationalised for runtime attestation even in partially virtualised environments with limited device pass-through. Although some kernel integration steps were absent, the remote attestation pipeline remained intact and provides a robust foundation for integrating confidential workloads into sovereign defence cloud environments.

Despite these limitations, **Path B** demonstrated the feasibility of SEV-SNP-based attestation within AWS infrastructure and established a robust foundation for policy-bound key management in follow-up stages.

Building upon the local measurement capabilities established earlier, the second stage of **Path B** focused on validating the full SEV-SNP attestation chain, extending from guest report generation to dynamic key usage within `Vault`. To initiate this flow, the `snpguest` utility was used to trigger the creation of a signed TCB-bound attestation report, which was then processed using a custom `PyJWT` script and encoded into a JWT.

Figure 4.9: OmniAware-EC2-SEV-SNP-Ubuntu - Posting the JWT to `Vault` for Authentication via the `jwt/login` Endpoint: The attestation token is submitted alongside the associated role (`sev-snp-role`) to retrieve a scoped `Vault` token.

As illustrated in Figure 4.9, the JWT was submitted to the `Vault jwt/login` endpoint, triggering authentication and policy evaluation against the `attestation-policy` binding. Successful login yielded a short-lived scoped `Vault` token, enabling subsequent cryptographic operations under enclave-bound constraints.

To prepare the key material, a dedicated key ring (`attestation-test`) was configured within the `Vault transit` secrets engine. This key was defined to support decryption and derivation while disallowing export and plaintext backup.

```
root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320#          vault write -f transit/keys/attestation-tes
Key                       Value
---                       -----
allow_plaintext_backup    false
auto_rotate_period        0s
deletion_allowed          false
derived                   false
exportable                false
imported_key              false
keys                      map[1:1749907186]
latest_version            1
min_available_version     0
min_decryption_version    1
min_encryption_version    0
name                      attestation-test
supports_decryption       true
supports_derivation       true
supports_encryption       true
supports_signing          false
type                      aes256-gcm96
```

Figure 4.10: OmniAware-EC2-Vault - Key Configuration for Attestation Validation in `Vault`: The `attestation-test` key is restricted for enclave-based operations and bound to the attestation policy.

```
root@OmniAware-EC2-SEV-SNP-Ubuntu:/opt/snpguest-test# vault write transit/decrypt/attestation-test ciphertext="vau
lt:v1:VW1/P4nqSUHRDEb1CjEmiVAwNS6KtjThRjlj82tzTxI+GFMZ"
Key          Value
---          -----
plaintext    U0dWc2JnPT0=
root@OmniAware-EC2-SEV-SNP-Ubuntu:/opt/snpguest-test#
```

Figure 4.11: OmniAware-SEV-SNP-Ubuntu - Successful Remote Decryption: The ciphertext is decrypted using `Vault` and attestation-bound access control.

The generated token was then used to request plaintext decryption via the `Vault` `transit/decrypt` path. Figure 4.11 demonstrates a successful roundtrip, confirming that the `Vault` instance honoured the cryptographic operation request using the enclave-bound token identity.

```
root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320# cat /home/ubuntu/vault-keys.txt
root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320# export VAULT_ADDR=http://127.0.0.1:8200
vault operator init -key-shares=1 -key-threshold=1 > /home/ubuntu/vault-keys.txt
root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320# UNSEAL_KEY=$(grep 'Unseal Key 1' /home/ubuntu/vault-key
s.txt | awk '{print $NF}')
vault operator unseal "$UNSEAL_KEY"
Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             false
Total Shares       1
Threshold          1
Version            1.19.5
Build Date         2025-05-29T09:17:06Z
Storage Type       file
Cluster Name       vault-cluster-653d495b
Cluster ID         d229ccce-1e5c-3bbc-74d5-1195ce0307c9
HA Enabled         false
```

Figure 4.12: OmniAware-EC2-Vault - `Vault` Operator Unseal: Demonstrates successful initialisation and activation of the `Vault` instance.

Prior to this test, the `Vault` instance was initialised using a single-key `shamir` configuration and unsealed via operator token.

```
root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320# ROOT_TOKEN=$(grep 'Initial Root Token' /home/ubuntu/vau
lt-keys.txt | awk '{print $NF}')
vault login "$ROOT_TOKEN"
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                hvs.u9tNfwka8LLpthyQbUqtMIMQ
token_accessor       dWGuVK4jhDgI60biWslObVwB
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Figure 4.13: OmniAware-EC2-Vault - Root Token Login: Confirms unrestricted access to initialise and inspect audit logs.

Root login was performed manually to validate baseline access and inspect audit logs.

`Vault` audit logging confirmed that the decryption request was properly authorised and policy-matched, offering transparency into runtime activity and reinforcing policy-based observability.

```
aws ssm start-session --target i-05e8ce429e30b0fee --region eu-west-1

root@OmniAware-EC2-Vault:/var/snap/amazon-ssm-agent/11320# cat /var/log/vault/audit.log | jq
{
  "request": {
    "id": "2bf169ae-d3e3-f571-f4f2-d2f3974f8b34",
    "namespace": {
      "id": "root"
    },
    "operation": "update",
    "path": "sys/audit/test"
  },
  "time": "2025-06-15T17:13:31.375772546Z",
  "type": "request"
}
{
  "auth": {
    "accessor": "hmac-sha256:c85b42170c62be63fd91e27229e98bbed8014ffb6d7a587428271d1e3669da78",
    "client_token": "hmac-sha256:af102540ff28304357d3e5b516e2b1aa1b1c6afdc511d264a5979c6d1317ca29",
    "display_name": "root",
    "policies": [
      "root"
    ],
    "policy_results": {
      "allowed": true,
      "granting_policies": [
        {
          "type": ""
        },
        {
          "name": "root",
          "namespace_id": "root",
          "type": "acl"
        }
      ]
    },
    "token_policies": [
      "root"
    ],
    "token_issue_time": "2025-06-15T16:19:42Z",
    "token_type": "service"
  },
  "request": {
    "client_id": "0DHqvq2D77kL2/JTPSZkTMJbkFVmUu0TzMi0jiXcFy8=",
    "client_token": "hmac-sha256:af102540ff28304357d3e5b516e2b1aa1b1c6afdc511d264a5979c6d1317ca29",
    "client_token_accessor": "hmac-sha256:c85b42170c62be63fd91e27229e98bbed8014ffb6d7a587428271d1e3669da78",
    "data": {
      "description": "hmac-sha256:634591cd222aa42a1468a9c120895775986987303b1f046e0f5d2031a4fcde7f",
      "local": false,
      "options": {
        "file_path": "hmac-sha256:3cfc6436de25da5aeecbc61bb603acabab19dccd90f29f136b0c4dd5e5e0e15b"
      },
      "type": "hmac-sha256:f88d81b8146311c8fb10cabd6510b4bf0d27350cae02decea248cc7763080c6c"
    },
    "headers": {
      "user-agent": [
        "Go-http-client/2.0"
      ]
    },
    "id": "c8f6c202-7978-dcb8-a709-b868568b1664",
    "mount_accessor": "system_0841f0ed",
    "mount_class": "secret",
    "mount_point": "sys/",
    "mount_running_version": "v1.19.5+builtin.vault",
    "mount_type": "system",
    "namespace": {
      "id": "root"
    },
    "operation": "update",
    "path": "sys/audit/file",
    "remote_address": "127.0.0.1",
    "remote_port": 33526
  },
  "time": "2025-06-15T17:13:31.376711206Z",
  "type": "response"
}
```

Figure 4.14: OmniAware-EC2-Vault - `Vault` Audit Log View: The full token path and client metadata were logged during secure decryption.

These results validate the feasibility of SEV-SNP-based attestation workflows within a practical AWS deployment, including full token generation, authentication and attestation-bound secret access. Together, the sequence represents a working Confidential Computing chain from guest report to zero-trust `Vault` enforcement — successfully realising the intent of **Path B**.

4.4.3 *Interfaces: Secure Ingest Gateway Testing*

In order to enable sovereign, controlled and scalable ingestion of mission telemetry and operational metadata, the *OmniAware* platform incorporates a dedicated Secure API Gateway. This gateway is responsible for exposing pre-validated endpoints for data intake from mission components, ensuring that payloads are syntactically compliant, semantically scoped and securely transmitted. It is deployed in conjunction with AWS API Gateway, Lambda transformation functions and downstream ingestion pipelines (e.g. via Kinesis, Firehose and DynamoDB). The architectural configuration supports strict schema enforcement, authentication and role-based access control (RBAC), as well as near real-time operational validation of incoming data.

The ingestion mechanism is intentionally decoupled from sensor-facing APIs to enforce a clear security boundary between exposed interfaces and internal data pipelines. Each endpoint is modelled and validated according to domain-specific JSON schemas. For defence-specific telemetry, this includes the NGVA structure, aligned with STANAG 4754 [7]. The API Gateway transforms and verifies inbound telemetry in accordance with these NGVA-aligned schemas before triggering Lambda-based pre-processing and persistent storage.

In the current implementation, a functionally advanced yet non-final version of the Secure Ingestion Gateway was deployed within the dedicated AWS `ingest` account managed by GroupIT. While TLS integration and certificate lifecycle management remain pending, the deployed setup enabled the ingestion of realistic telemetry and image payloads. This allowed for early validation of production-adjacent ingestion flows and provided a practical opportunity to observe and assess the NGVA-aligned data model under near-operational conditions.

Validated Ingestion of Telemetry Data

To validate the ingestion of vehicle telemetry using the new NGVA-aligned data model, the Secure Ingestion API was evaluated using structured input compliant with a simplified version of STANAG 4754. The test input defined a single telemetry object consisting of timestamped configuration metadata:

```
1 { "DateTime": "2025-06-23T08:26:18Z", "Vehicle_Configuration":
  ↪ {"Actual_Configured_Vehicle": { "vehicleId": "Y-4242" }}}
```

This payload was submitted via an authenticated `POST` request to the `/v1/telemetry` endpoint. The API Gateway accepted the request with `Status 200`, performed request validation and forwarded the payload to the internal transformation Lambda (`OmniAware-TelemetryDataIngest`). Execution logs confirm the successful receipt, schema validation, processing and handoff to Firehose.

```
2025-06-23T09:06:08Z: “message”: “Record ingested successfull”
RequestId: 716c3858-f7bf-43ce-825d-caf10dd392f6, Integration latency: 2595ms
```

The transformed payload was subsequently ingested by a dedicated Firehose delivery stream, which wrote the validated data to the `TelemetryData` table in DynamoDB. As confirmed by DynamoDB query results, the data was persistently stored with the correct `vehicle_id` and timestamp metadata:

```
1 { "vehicle_id": "Y-4242", "timestamp": "2025-06-23T08:26:18Z", "data": { "DateTime":
  ↪ "2025-06-23T08:26:18Z",
2 "Vehicle_Configuration": { "Actual_Configured_Vehicle": { "vehicleId": "Y-4242" } },
  ↪ "transform": "Hello from Firehose
3 Transform Lambda." } }
```

This successful test validates not only the functionality of the ingestion flow, but also the system's ability to process and store structured mission telemetry in accordance with future-proof, NATO-aligned data models. The modular implementation also lays the groundwork for extending the schema to full NGVA compliance, including support for health monitoring, positional telemetry and multi-modal sensor fusion.

Validated Ingestion of Image Data

As a complementary capability to structured telemetry ingestion, the PoC pipeline was extended to support image-based intelligence capture and classification. While the Secure Ingestion Gateway deployed in the `AWS Guild` account remains a minimal prototype primarily intended for testing confidential computing workflows such as `Vault` integration and remote attestation, a more advanced and functionally extended version was provisioned within the `GroupIT` AWS `ingest` account. This extended setup enabled the validation of near-operational ingestion flows for unstructured image data, allowing for practical testing of NGVA-aligned payloads under realistic mission conditions.

Figure 4.15: Sample Image - Used for Ingestion Test: The image, featuring several tracked military vehicles, was encoded and submitted for processing via the ingestion API [68].

Image encoding was conducted client-side using `base64`, a common prerequisite for transmitting binary payloads over JSON-based REST APIs. The transformation command is shown below:

```
cat <Insert Image Path> | base64 > <Insert Base64 File Path>.txt
```

After submission, the processing pipeline — which includes AWS Lambda-backed processing logic — successfully triggered a classification job. As shown in Figure 4.16, the image was partially parsed by the inference engine and several visual attributes were extracted.

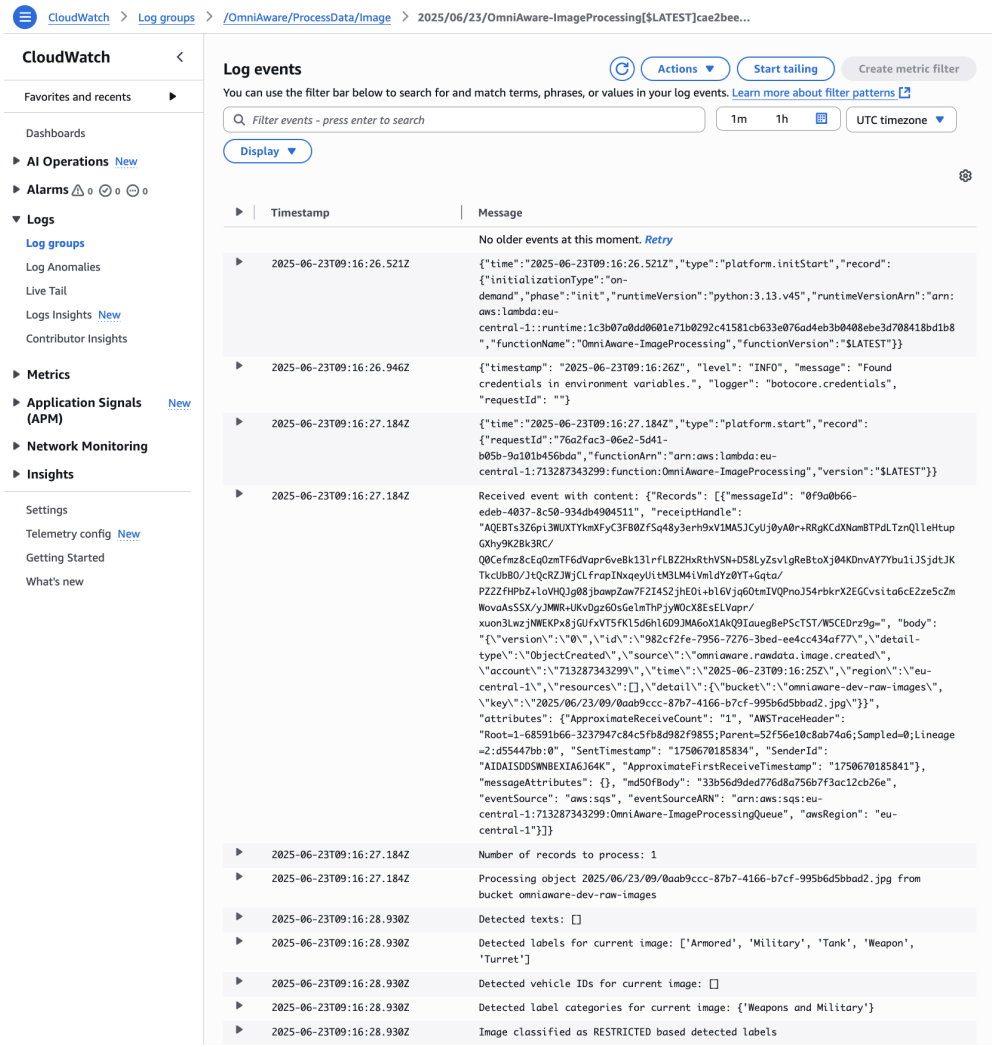


Figure 4.16: CloudWatch - Logs for Image Ingestion: Log entries confirm Lambda execution and partial labelling of visual features, including class labels such as `Tank` and `Armored`.

Despite these partial successes, the ingestion pipeline exhibited limitations in downstream handling. While the raw labelling results were correctly detected and logged, only incomplete metadata entries were generated — ultimately preventing a consistent write to DynamoDB. As shown in the logs, not all expected attributes (e.g. geolocation, structured vehicle descriptors) were inferable for each detected entity. Consequently, no finalised entry for the sample was persisted in the data store.

This behaviour aligns with design expectations: the underlying Lambda function halts record propagation when required fields remain undefined, preserving schema integrity. It also demonstrates the necessity of precise bounding box detection and attribute extraction for all identified objects within the image, particularly when multiple vehicles are present.

In contrast, a prior test image featuring a single object (a toy tank model with licence plate overlay) yielded a successful end-to-end flow. This suggests that image ingestion workflows currently favour atomic, well-separated scenes over complex compositions, which require more resilient object detection pipelines.

These results validate the ingestion interface from an operational perspective, while highlighting the importance of tighter coordination between image complexity, AI inference stability and downstream data model mapping. Future iterations will benefit from enriched inference metadata, stricter schema enforcement and fallback strategies for incomplete detections.

5 INSIGHTS

Summary: This chapter synthesises the empirical insights and design implications derived from the implementation and evaluation of the *OmniAware* platform. It focuses on three architectural pillars — cloud and edge computing, confidential computing and system interoperability — and consolidates the practical lessons learned across heterogeneous deployments, prototype validations and cross-account integrations.

The insights confirm the architectural soundness of enclave-backed secure infrastructure components, highlight systemic limitations in serverless paradigms under enclave constraints and validate the need for hybrid enforcement models combining managed cloud services with runtime attestation and identity-bound execution logic. The findings also expose operational trade-offs, including regional dependency of TEE services, orchestration overhead in enclave migration and compliance complexities in zero-trust integrations.

Moreover, the interoperability evaluation revealed critical schema-level and policy-bound integration challenges, both within standardised NGVA ingest flows and exploratory telemetry/image pipelines. Key lessons included the impact of runtime metadata enforcement, policy-bound JWT validation and the strategic role of `Vault` in establishing hardened API gateways for secure multi-account control.

Taken together, the findings offer tactical guidance for sovereign, mission-grade platform design and establish a viable blueprint for MDO-aligned deployments, bridging cloud-native technologies with coalition-ready enforcement primitives.

5.1 CLOUD AND EDGE COMPUTING

The implementation of cloud and edge capabilities within the *OmniAware* platform highlighted both the operational complexity and architectural trade-offs inherent in hybrid deployments. Using AWS EC2 instances and modular account separation for components such as Ingest, Secure Infrastructure and `Vault`, the project reinforced the value of IaC-based repeatability and environment-specific control. However, several deployment-related insights emerged during real-world implementation that hold direct relevance for sovereign, secure mission platforms.

One key observation was the strict dependency of certain AWS confidential computing services on region-specific availability. For example, AMD SEV-SNP-enabled instance types and AWS Nitro Enclaves are limited to selected regions, requiring careful planning of resource locations during architecture design. This affects both scalability and cross-region portability — a critical constraint when aiming for multinational or federated mission clouds. Additionally, enabling these services is contingent not only on region and instance type, but also on AMI version, hypervisor configuration and kernel compatibility, adding significant pre-deployment complexity [56].

While SEV-SNP was successfully integrated and validated through remote attestation workflows using native attestation reports and JWT generation, AWS Nitro Enclaves introduced additional hurdles due to the requirement for building and attaching enclave-compatible Docker containers. This process demands tight alignment with AWS' Nitro Enclaves C SDK and enclave definition — raising the implementation barrier, especially for projects requiring rapid prototyping.

From a security architecture perspective, e.g. the GIT (Ingest) account served as a proving ground for enforcing native security controls via CloudFormation. By leveraging AWS Guardrails and IAM permissions boundaries, critical policies could be embedded directly into the stack templates — demonstrating that policy-as-code and compliance-by-design principles can be achieved natively within the AWS ecosystem. This contrasts favourably with external compliance tooling often used in Terraform or Azure-based deployments, where organisational overlays are typically required.

A key operational insight relates to the native security capabilities embedded in the AWS

IaC stack — in particular the use of *Guardrails* via CloudFormation templates. These guardrails enable the declarative enforcement of policies (e.g. mandatory encryption, network restrictions, role scoping) during infrastructure provisioning. This approach aligns with the *compliance-by-design* paradigm and significantly reduces the attack surface by integrating security controls as code. Compared to external policy engines in Terraform-based workflows or post-deployment configuration scripts in other cloud environments, this mechanism offers a tightly coupled, verifiable enforcement layer. Especially in regulated or defence-aligned deployments, such IaC-level controls provide a robust foundation for auditable and zero-trust infrastructure baselines. Unlike Terraform or Azure ARM-based deployments — where compliance logic often resides in loosely coupled pipelines or third-party tooling — the AWS approach enables inline constraint embedding directly within the deployment artefacts. This tight integration supports reproducibility, enforces consistency across environments and allows modular policy reuse across accounts and regions.

An operational insight relates to SSM, which enabled automated bootstrapping and remote control of cloud resources without requiring direct SSH access. This streamlined deployment processes and reduced exposure risk, particularly in secure zones. Compared to traditional management interfaces, SSM offered improved integration with attestation-triggered workflows and reduced the need for manually maintained bastion host architectures.

Regarding edge computing, the *OmniAware* architecture included conceptual extensions for forward-deployed edge devices, particularly in the PHM scenario. However, practical deployment and validation were not feasible due to constraints on physical infrastructure and runtime environments. Despite this, design-time evaluations allowed for several key considerations. Notably, edge platforms must account for limited or air-gapped connectivity, which may inhibit online Remote Attestation, policy validation or secret provisioning. This introduces a stark divergence from cloud-native assumptions and necessitates alternative strategies such as embedded trust roots or pre-validated image bundles.

The broader comparison of platforms and tooling revealed noteworthy implications for future mission platform design. The native cohesiveness of AWS (with integrated identity, telemetry and encryption services) proved advantageous over alternative solutions involving external orchestration (e.g. Kubernetes-based TEE runtimes or cross-provider Terraform deployments). These findings suggest that cloud-native security and automation can be leveraged effectively — provided that architectural awareness of service constraints, regional availability and compliance boundaries is embedded early in the design process. For sovereign deployments (e.g. NATO private clouds or coalition-owned platforms), this necessitates explicit portability strategies and abstraction layers to mitigate vendor lock-in and capability fragmentation.

5.2 CONFIDENTIAL COMPUTING

Confidential computing was implemented as a foundational architectural principle, not as a post-hoc security enhancement. The prototype validated secure enclave bootstrapping, remote attestation and conditional secret release by integrating AMD SEV-SNP-enabled EC2 instances, JWT-based attestation and a policy-enforced `Vault` deployment. A dedicated CloudFormation stack (`15_cc-vault-poc.yaml`) provisioned a fully functional `Vault` server capable of managing encrypted payloads, policy-based transit key logic and attestation-anchored access control via validated JWT claims.

While JWT formed the core of the attestation logic, alternative mechanisms such as TPM-backed assertions and Intel SGX quoting protocols were conceptually assessed. In practice, only SEV-SNP and AWS Nitro Enclave attestation mechanisms offered fully cloud-integrated trust anchors via AWS APIs. SEV-SNP provided the most transparent and repeatable flow, using the native `snpguest` utility for claim generation and report validation. While `Vault` was selected for its maturity, extensibility and native ecosystem support, it was benchmarked against lighter alternatives such as SPIRE and EnclaveOS-based attestation agents — highlighting trade-offs between deployment complexity, trust depth and cloud-native alignment.

The implementation surfaced multiple operational challenges. These included strict requirements for AMI kernel versions, enclave launch flags, Nitro Enclave definition constraints

and `Vault` policy design. Establishing a reliable trust chain required iterative debugging across enclave measurement reporting, JWT decoding, `Vault` AppRole and Transit Engine configurations. Inconsistencies — such as mismatched 'aud' claims or invalid timestamps — repeatedly triggered policy rejection until resolved through custom debugging scripts and log-augmented policy evaluations.

Crucially, TLS encryption was not fully implemented during the PoC phase. While `Vault` supported TLS endpoints, the setup lacked an integrated certificate lifecycle manager (e.g. AWS Certificate Manager or HashiCorp `Vault` PKI Engine). This omission represented a critical gap in trust anchor binding, exposing the system to potential Man-in-the-Middle (MitM) vectors in multi-domain deployments. Similarly, public key rotation, revocation and authority management were not operationalised — highlighting the importance of embedded PKI logic in production-grade deployments.

Nonetheless, the implemented trust chain validated the end-to-end flow from enclave measurement to conditional secret provisioning. This was most evident in deployments within the Secure Infrastructure environment, where both a minimal `Vault` instance and a fully integrated ingestion pipeline were tested. Verified JWTs from SEV-SNP-enabled instances were successfully parsed and enforced via `Vault`, establishing auditable and cryptographically anchored runtime policies.

In contrast, the GIT account provided the technical foundation for the broader multi-account *OmniAware* platform, hosting the development of key components such as the Ingest, Consumer, Datalake, Audit and Security stacks. Within the scope of this PoC, it served to validate advanced API gateway deployments and demonstrate interoperability with ingestion flows and telemetry pipelines. While `Vault` was not deployed in the GIT environment, the groundwork was established to enable `Vault`-hardened API endpoints — positioning the GIT account as a future-ready target for integrated policy enforcement and confidential ingress control.

Taken together, the findings reaffirm confidential computing as a mission-ready, extensible security paradigm. However, real-world deployments must incorporate complete PKI management, automated certificate provisioning and full TLS enablement to harden the underlying trust infrastructure. With these enhancements, the blueprint established by this PoC can scale into coalition-grade defence platforms that are verifiable, composable and operational under adversarial conditions.

Yet, adopting SEV-SNP comprehensively across a heterogeneous cloud landscape introduces systemic design challenges. Serverless services —– such as Lambda functions, Step Functions or managed control plane components —– do not natively run within TEE-enabled environments. This limitation arises from the very nature of serverless: workloads are dynamically instantiated in isolated, short-lived compute containers managed entirely by the platform provider, offering neither control over instance types nor the ability to inject custom launch flags or kernel parameters required for enclave initialisation.

Consequently, enforcing consistent enclave-based trust boundaries requires rearchitecting such services to execute within dedicated SEV-SNP-backed compute instances. This transition would entail containerisation or workload migration, coupled with reimplementation of orchestration logic (e.g. retries, scaling, triggers) previously abstracted by platform services. Without direct control over the ephemeral compute layer, serverless paradigms remain fundamentally incompatible with attestation-driven trust chains, highlighting the trade-off between operational abstraction and fine-grained security enforcement in sovereign computing scenarios.

The implication is twofold: First, zero-trust guarantees tied to enclave-backed execution environments cannot yet be assumed for all control flow paths in serverless-first architectures. Second, the benefits of confidential computing — verifiable trust, runtime encryption and conditional secret access — can only be systematically extended through conscious trade-offs in automation, operational overhead and cloud-native abstraction layers.

Therefore, future deployments must strategically identify mission-critical services that justify enclave migration and selectively retain managed services where risk profiles permit. A hybrid enforcement architecture — combining enclave-anchored trust for core telemetry and policy logic with hardened, identity-bound platform services — may offer a realistic path forward until TEE-backed primitives gain broader platform integration.

5.3 INTEROPERABILITY

Interoperability within the *OmniAware* architecture was pursued at three levels: data model standardisation, interface design and deployment modularity. The ingestion interfaces were aligned with NGVA JSON formats, while telemetry and image data were prepared using consistent encoding, schema enforcement and metadata annotations.

The Secure Ingest API, validated within the PHM scenario, allowed structured JSON ingestion (as per the simplified `NGVA` sample) and unstructured image submission with base64-encoded payloads. CloudWatch traces confirmed Lambda invocations and data processing flows across all ingestion channels. `Vault`-based key management, though not yet active in this scenario, was fully prepared to support conditional decryption and validation per workload through attestation-bound JWTs.

A key insight was the necessity to define clear interface boundaries not only between producer and consumer systems, but also between policy-enforced and policy-free domains. The use of cross-account S3 bucket access, IAM boundary conditions and Lambda triggers enabled a high degree of modularity, while simultaneously introducing non-trivial complexity in policy validation and runtime enforcement.

Beyond the Secure Infrastructure deployment, the GIT account provided the required components. While `Vault` was not deployed in this environment, API gateway prototypes were extended to validate telemetry and image flows under more realistic and mission-oriented ingestion conditions. These tests, although slightly divergent from the original PHM use case, revealed critical insights into schema fidelity, runtime constraints and the interplay of service integration under federation-grade workloads.

Interoperability assessments highlighted several technical nuances:

- Enforcement of metadata consistency for image payloads was crucial to enable downstream control logic and ingestion traceability.
- Schema-bound interfaces exposed subtle alignment frictions within asynchronous AWS-native workflows and serverless ingestion channels.
- Early-stage security evaluations revealed the need for standardised tag structures, identity headers and future JWT-based validation at API ingress points.

Security controls applied during the PoC included S3 policy scoping, strict IAM enforcement and Lambda runtime constraints. These were sufficient for modular ingest validation, yet future deployments should incorporate `Vault`-integrated JWT decoding, signed telemetry claims and policy-bound enforcement logic. Additional hardening options include the use of SCPs conditional token validation and ingress pre-validation workflows.

Despite the fragmented state of federated standards in military systems, this PoC demonstrated that secure, modular and NAFv4-aligned interoperability can be achieved through schema alignment, runtime attestation and minimal interfaces backed by strong control logic. These results establish a blueprint for resilient API design and interoperability strategies in future MDO architectures, particularly under mission-grade, coalition-ready constraints.

Beyond telemetry and API gateway hardening, the results demonstrated the broader applicability of confidential computing. Runtime enforcement logic could be extended to ingress/egress control systems, encrypted image pipelines, operator dashboards, or any compute-bound component with secret-bound state logic. These trust extensions are programmable via `Vault`'s dynamic secrets model and scalable to operational blueprints.

From an architectural perspective, three extensibility levers were validated: (1) enclave-based validation for signed configuration files; (2) runtime authentication for control plane commands; and (3) zero-trust enforcement for microservice workloads. Each pathway builds upon the same attestation logic — providing a modular, reproducible enforcement pattern aligned with zero-trust and data-sovereign computing principles.

Taken together, the findings reaffirm confidential computing as a mission-ready, extensible security paradigm. However, real-world deployments must incorporate complete PKI management, automated certificate provisioning and full TLS enablement to harden the underlying trust infrastructure. With these enhancements, the blueprint established by this PoC can scale

into coalition-grade defence platforms that are verifiable, composable and operational under adversarial conditions.

Yet, adopting SEV-SNP comprehensively across a heterogeneous cloud landscape introduces systemic design challenges. Serverless services — such as Lambda functions, Step Functions or managed control plane components — do not natively run within TEE-enabled environments. Consequently, enforcing consistent enclave-based trust boundaries requires rearchitecting such services to execute within dedicated SEV-SNP-backed compute instances. This transition would entail containerisation or workload migration, coupled with reimplementation of orchestration logic (e.g. retries, scaling, triggers) previously abstracted by platform services.

The implication is twofold: First, zero-trust guarantees tied to enclave-backed execution environments cannot yet be assumed for all control flow paths in serverless-first architectures. Second, the benefits of Confidential Computing — verifiable trust, runtime encryption and conditional secret access — can only be systematically extended through conscious trade-offs in automation, operational overhead and cloud-native abstraction layers.

Therefore, future deployments must strategically identify mission-critical services that justify enclave migration and selectively retain managed services where risk profiles permit. A hybrid enforcement architecture — combining enclave-anchored trust for core telemetry and policy logic with hardened, identity-bound platform services — may offer a realistic path forward until TEE-backed primitives gain broader platform integration.

6 CONCLUSION

Summary: This concluding chapter synthesises the research findings by revisiting the three research questions and aligning them with the practical implementation results of the *OmniAware* PoC. The evaluation confirmed that a NAFv4-compliant defence cloud architecture can be systematically modelled using the ArchiMate language and open-source tooling, enabling traceable system design and capability-driven compliance. Furthermore, the thesis validated the applicability of confidential computing through hardware-enforced enclaves and remote attestation protocols, enabling secure workload protection and cryptographic key isolation even under adversarial threat conditions. Interoperability across cloud, edge and HPC domains was addressed via federated attestation mechanisms, decentralised identity management and enclave-integrated microcontroller concepts.

While the *PoC* demonstrates the technical viability of the proposed platform, the outlined *Outlook* section highlights several strategic extensions — including RT fleet monitoring, Digital Twin integration and AI-driven decision support — to further enhance mission readiness and operational agility.

From a defence transformation perspective, this work positions *OmniAware* as a transferable and doctrine-compatible blueprint for secure digital military infrastructures. Based on the achieved outcomes and demonstrator maturity, follow-on activities are expected — potentially as successive proof-of-concept implementations or scaled deployments with prospective defence stakeholders.

The lessons gained not only validate platform feasibility but also contribute to a broader methodological foundation for sovereign, resilient and alliance-integrated defence cloud systems. Looking ahead, the *OmniAware* platform provides a promising baseline for strategic alignment with future NATO, EU and national defence digitalisation agendas. As geopolitical pressures and technological complexity increase, the demand for verifiable, mission-ready platforms capable of operating across multi-domain theatres will grow. Continued iteration and alignment with operational feedback will be key to scaling the platform towards production-grade defence infrastructure.

6.1 EVALUATION

With the accomplished investigation, the research questions could be answered as follows.

RQ1: How can a cloud-native defence architecture be designed to ensure compliance with the NATO Architecture Framework Version 4 (NAFv4) while supporting secure and scalable mission-critical operations?

The research demonstrated that a NAFv4-aligned defence architecture can be effectively designed and deployed using a viewpoint-driven methodology grounded in the ArchiMate modelling language and the open-source modelling tool Archi. The architecture strictly follows the layered construct of NAFv4 — from capability-based planning (NCV) to deployment artefacts (NPV) — ensuring semantic traceability, design-time modularity and mission-driven extensibility.

The conceptual architecture incorporates Infrastructure-as-Code (IaC) principles for end-to-end automation, Kubernetes-based orchestration and containerised microservices — especially in the context of future platform-level deployments. However, the actual prototype implementation focused on EC2-based confidential nodes, specifically using AMD SEV-SNP and Nitro Enclaves to simulate trusted execution environments. These were evaluated in isolated test scenarios to explore the feasibility of attestation-enabled workloads and secure secret management in mission-relevant conditions.

Compared to traditional static architectures, the proposed approach facilitates dynamic service composition and enforces policy constraints already at deployment time. Moreover,

the explicit modelling of capability-to-service mappings supports coalition interoperability and auditability in multinational missions — an essential aspect given the federated nature of NATO-led operations.

Nevertheless, a current limitation lies in the absence of automated semantic validation between logical views (NLV) and platform deployments (NPV), which may lead to mismatches if not manually aligned. Future work could address this via model-driven policy enforcement or ontology-supported validation pipelines.

Overall, the proposed architecture meets NAFv4 compliance criteria by design and serves as a reference blueprint for secure, scalable and sovereign military-grade cloud environments. The applied methodology is viable and extensible, aligning with NATO's digitalisation goals and the operational demand for mission-centric system resilience.

RQ2: What are the key security challenges in defence cloud infrastructures and how can a confidential computing-based security model be validated to ensure compliance with defence security standards?

The key challenges in defence-oriented cloud infrastructures revolve around securing data-in-use, maintaining federated trust across sovereign domains and ensuring runtime protection under disconnected or adversarial network conditions. Traditional perimeter- or VM-based security controls prove insufficient under such mission conditions, prompting the need for hardware-anchored isolation strategies.

This research explored the use of confidential computing capabilities via SEV-SNP and Nitro Enclaves to provision secure execution environments for sensitive workloads. By leveraging hardware-backed TEEs, the architecture enabled the foundational setup for attestation-driven key release and policy-enforced access control through `Vault`. A partially automated Remote Attestation workflow was prototyped on EC2 instances, covering the generation of JWTs, `Vault` role binding and the use of the Transit Secret Engine for encryption and audit logging. While `vsock`-based communication between parent and enclave processes was functionally utilised (e.g. for retrieving container output), it remained outside the formal evaluation scope and was not subject to explicit security or performance assessment.

Rather than deploying a fully integrated container-based pipeline, the evaluation focused on standalone EC2-based test setups within the PHM and CIVS scenarios. These testbeds aimed to assess the feasibility of embedding attestation and secure secret management mechanisms in accordance with international defence standards, including `STANAG 4774/4778` and `AC322-D`, in order to establish auditable and assurance-ready infrastructure under mission-level constraints.

Compared to traditional TPM-centric approaches, the selected confidential computing paradigm offers enhanced flexibility for runtime validation and policy enforcement across sovereign boundaries. However, notable implementation gaps remain — including missing container-level attestation chains, limited policy granularity and the absence of a fully automated attestation-to-enforcement control flow — pointing to the need for abstracted trust anchors, more streamlined tooling and refined integration patterns.

However, several limitations must be acknowledged: (1) current implementations lack fine-grained user-level attestation semantics, (2) the complexity of onboarding confidential computing workloads remains high and (3) secure multi-tenancy under full isolation is not yet feasible without further platform extensions. These constraints highlight the need for simplified trust anchors, more abstracted developer tooling and automated attestation trust pipelines.

Therefore, the evaluated security concept illustrates the potential of confidential computing for sovereign-grade defence architectures and provides a validated entry point for future zero-trust compliant deployments in coalition-ready mission platforms.

In summary, the security model validates the practical use of confidential computing in real-world military scenarios, bridging Zero Trust principles with defence interoperability requirements. It sets a precedent for scalable, verifiable and standard-compliant mission workloads in coalition-grade cloud infrastructures.

RQ3: How can interoperability between cloud, edge and HPC environments be ensured in a defence cloud infrastructure while maintaining security and operational efficiency?

Ensuring interoperability across heterogeneous execution environments — including sovereign cloud regions, forward-deployed edge devices and HPC backends — poses substantial architectural challenges. These stem from differing trust domains, communication paradigms and performance expectations.

To address these challenges, the proposed defence cloud architecture incorporates federated attestation, secure API endpoints and decentralised identity management. A draft version of the Secure Ingestion Gateway was prepared to support encrypted telemetry via gRPC-based data pipelines and policy-controlled access via `Vault`. However, core hardening measures such as TLS enforcement and attestation-validated request filtering have not yet been fully integrated. Current edge TEEs remain limited in terms of microcontroller compatibility, attestation depth and cryptographic throughput. In particular, the absence of lightweight attestation frameworks hampers integration with resource-constrained sensor platforms.

Two variants of the ingestion gateway were prototyped in separate operational contexts: one focused on telemetry and image ingestion for validation in the *GroupIT* account and one targeting secure deployment and testing within the AWS Guild environment. While initial ingestion flows relied on enclave-enabled EC2 nodes, the full implementation of container-level attestation and `Vault`-backed key release for the gateway remains an open item for future work.

Standardised message schemas and interface definitions were followed where possible, laying a foundation for secure service integration. Nevertheless, current deployments only partially realise the envisioned trust model and highlight remaining gaps in operational maturity and zero-trust enforcement.

Future work should focus on incorporating support for emerging embedded TEEs platforms such as `OP-TEE` and on harmonising secure ingestion mechanisms across coalition partners via open standards and portable trust anchors.

Overall, the architecture presents a viable model for enabling secure and interoperable mission dataflows across tactical edge, operational cloud and strategic HPC layers — fulfilling core requirements for next-generation military intelligence and operations platforms.

6.2 OUTLOOK

While the presented implementation validates core components of the *OmniAware* platform, several opportunities for enhancement, operational scaling and strategic integration remain. These are categorised below into near-term extensions and long-term strategic recommendations.

The integration of secure telemetry pipelines with RT fleet and unit monitoring capabilities represents a foundational enhancement for mission-critical operations. Ingested sensor data — once cryptographically verified — can be used to assess platform readiness, operational reliability and tactical performance in near real time. This enables advanced mission oversight, anomaly detection and dynamic reconfiguration of assets within trusted compute environments.

As a potential extension, digital twin simulation systems could be incorporated to enable predictive maintenance, scenario-based training and tactical what-if simulations. Enclaved simulation agents may further ensure that sensitive mission models are executed within TEE-protected environments, preserving confidentiality and operational integrity.

The confidential analytics layer may be extended to include AI-based decision support, enabling secure inference on encrypted data streams for anomaly detection, command recommendations and mission-level optimisation — particularly in contested or coalition-led theatres. In the long term, integration with autonomous and semi-autonomous systems (e.g. UAVs, UGVs) could be explored, with TEEs enforcing operational boundaries for rules of engagement.

From a strategic standpoint, the convergence of confidential computing, data sovereignty and NAFv4-compliant architecture establishes a new doctrine for trusted digital defence platforms. It enables command structures to rely on verifiable system states, distribute trust across organisational boundaries and adopt flexible cloud-native deployments without compromising control or compliance. The proposed methodology can inform procurement guidelines, certi-

fication frameworks and future mission platform architectures — positioning *OmniAware* as a transferable blueprint for next-generation sovereign defence systems.

In this context, continued reference to the NAFv4 architectural model is encouraged — not only for aligning capability decomposition and view-driven planning but also as a strategic guide for implementation phases across federated deployments. Emerging solutions such as Kata Containers remain promising in enabling remote attestation within containerised Kubernetes environments, offering enhanced workload isolation and runtime integrity verification. Additionally, the integration of `HashiCorp Vault` and associated attestation workflows can be further expanded beyond the current use cases, serving as a robust building block for hardening identity, access and secret management across the entire system landscape.

The *OmniAware* platform represents a significant step towards realising a secure, scalable and interoperable defence cloud architecture. By embedding confidential computing principles and aligning with NAFv4 standards, the implementation provides a robust foundation for future military operations in multi-domain environments. The insights gained from this PoC are expected to directly inform follow-on engagements — whether in the form of successive proof-of-concept implementations or full-scale pilot projects with prospective end-users. As the demand for trustworthy digital platforms continues to rise, the adaptability of *OmniAware* ensures its viability for integration into evolving operational ecosystems and long-term capability roadmaps. Operational feedback loops will play a decisive role in transforming the platform from prototype to field-ready capability.

REFERENCES

[1] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, ISSN: 0276-7783. DOI: 10.2307/25148625.

[2] P. M. Mell and T. Grance, *The nist definition of cloud computing*, Gaithersburg, MD, 2011. DOI: 10.6028/NIST.SP.800-145.

[3] E. Zyp et al., *Json schema: A media type for describing json documents (draft-04)*, https://json-schema.org/draft-04/draft-zyp-json-schema-04, Accessed: 2025-06-25, 2013.

[4] M. C. Nguyen and H. S. Won, "Data storage adapter in big data platform," *Proceedings - 8th International Conference on Database Theory and Application, DTA 2015*, pp. 6–9, Mar. 2016. DOI: 10.1109/DTA.2015.9.

[5] "STANAG 4774: Confidentiality Metadata Labelling Structure," NATO Standardization Office, Brussels, Tech. Rep. STANAG 4774, Dec. 2017, Edition 1. Version 1.

[6] X. Huang and R. Chen, "A survey of key management service in cloud," *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, vol. 2018-November, pp. 916–919, Jul. 2018, ISSN: 23270594. DOI: 10.1109/ICSESS.2018.8663805.

[7] NATO Standardization Office, *AEP-4754 vol vi: Architecture framework for tactical land sensor systems*, https://nso.nato.int/, STANAG 4754 Edition 1, Volume VI, NATO Architecture Framework Guidance for Land-Based Tactical Sensor Systems, 2018. [Online]. Available: https://nso.nato.int/.

[8] "STANAG 4778: Confidentiality Metadata Handling," NATO Standardization Office, Brussels, Tech. Rep. STANAG 4778, Oct. 2018, Edition 1. Version 1.

[9] E. C. Yildiz, M. S. Aktas, O. Kalipsiz, A. N. Kanli, and U. O. Turgut, "Data mining library for big data processing platforms: A case study-sparkling water platform," *UBMK 2018 - 3rd International Conference on Computer Science and Engineering*, pp. 167–172, Dec. 2018. DOI: 10.1109/UBMK.2018.8566278.

[10] "AC/322-D/0048-REV3 (INV): CIS Security Technical and Implementation Directive," NATO Consultation, Command and Control Board (C3B), Brussels, Tech. Rep. AC/322-D/0048-REV3 (INV), Nov. 2019, Supersedes AC/322-D/0048-REV2.

[11] Bundesamt für Sicherheit in der Informationstechnik (BSI), "Cloud Computing Compliance Criteria Catalogue C5:2020," Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany, Tech. Rep., 2020, Version 2020, English Translation. Accessed: May 31, 2025. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/CloudComputing/Compliance_Criteria_Catalogue_C5/Cloud_Computing_Compliance_Criteria_Catalogue_C5_2020.pdf.

[12] T. Jiao and A. Luo, "Probabilistic framework for evaluating the capability and resilience of c4isr using bayesian networks," *Proceedings - 2020 7th International Conference on Information Science and Control Engineering, ICISCE 2020*, pp. 1031–1036, Dec. 2020. DOI: 10.1109/ICISCE50968.2020.00211.

[13] Z. Jiao, J. Zhang, P. Yao, L. Wan, and L. Ni, "Service deployment of c4isr based on genetic simulated annealing algorithm," *IEEE Access*, vol. 8, pp. 65 498–65 512, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2981624.

[14] F. Y. Rashid, "What Is Confidential Computing?" *IEEE Spectrum*, 2020. [Online]. Available: https://spectrum.ieee.org/what-is-confidential-computing.

[15] M. Rouse, *What is edge computing? [...]* TechTarget, SearchDataCenter, 2020. Accessed: Apr. 27, 2025. [Online]. Available: https://searchdatacenter.techtarget.com/.

[16] J. Barton, "High performance computing for science and engineering in the department of defense," *Computing in Science and Engineering*, vol. 23, pp. 58–62, 6 2021, ISSN: 1558366X. DOI: 10.1109/MCSE.2021.3112288.

[17] A. Hutomo et al., "Evaluating the interoperability of c4isr system using cyber six-ware framework," *2021 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2021*, 2021. DOI: 10.1109/ICACSIS53237.2021.9631359.

[18] Z. Jiao, J. Zhang, P. Yao, L. Wan, and X. Wang, "C4isr service deployment based on an improved quantum evolutionary algorithm," *IEEE Transactions on Network and Service Management*, vol. 18, pp. 2405–2419, 2 Jun. 2021, ISSN: 19324537. DOI: 10.1109/TNSM.2021.3054752.

[19] D. P. Mulligan, G. Petri, N. Spinale, G. Stockwell, and H. J. M. Vincent, "Confidential computing—a brave new world," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, 2021, pp. 132–138. DOI: 10.1109/SEED51797.2021.00025.

[20] NATO Communications and Information Security Sub-Committee, "AC/322-D(2021)0032-REV1-U: NATO Security Instruction - Cloud Computing," North Atlantic Treaty Organization (NATO), Draft Security Instruction AC/322-D(2021)0032-REV1-U, 2021, Restricted Distribution, retrieved internally for academic purposes. Accessed: May 31, 2025. [Online]. Available: https://www.nato.int/.

[21] H. Won, M. C. Nguyen, M. S. Gil, and Y. S. Moon, "An advanced open data platform for integrated support of data management, distribution, and analysis," *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, pp. 2058–2063, 2021. DOI: 10.1109/BIGDATA52589.2021.9671504.

[22] A. Akram, V. Akella, S. Peisert, and J. Lowe-Power, "SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems," in *International Symposium on Secure and Private Execution Environment Design (SEED)*, 2022, pp. 121–132. DOI: 10.1109/SEED55351.2022.00018.

[23] N. Buchner, "Survey on Trusted Execution Environments," eng, 2022. DOI: 10.2313/NET-2022-07-1_05.

[24] Bundesamt für Sicherheit in der Informationstechnik (BSI), *Referenztabelle C5:2022 zu ISO/IEC 27001:2022*, https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Cloud-Computing/C5/C5.html, Excel-Dokument zur strukturierten Referenzierung zwischen C5:2022 und ISO/IEC 27001:2022, 2022. Accessed: May 31, 2025.

[25] N. Figueira, P. Pochmann, A. Oliveira, and E. P. D. Freitas, "A c4isr application on the swarm drones context in a low infrastructure scenario," *International Conference on Electrical, Computer, and Energy Technologies, ICECET 2022*, 2022. DOI: 10.1109/ICECET55527.2022.9872941.

[26] T. Geppert, S. Deml, D. Sturzenegger, and N. Ebert, "Trusted Execution Environments: Applications and Organizational Challenges," English, *Frontiers in Computer Science*, vol. 4, p. 78, 2022. DOI: 10.3389/fcomp.2022.930741. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fcomp.2022.930741/full.

[27] M. Khanuja and S. Subramanian, *Applied Machine Learning and High-Performance Computing on AWS*. Packt Publishing, 2022, ISBN: 978-1803237015. [Online]. Available: https://www.amazon.com/Applied-Machine-Learning-High-Performance-Computing/dp/1803237015.

[28] P. Silver, *Increase military readiness with aws iot for defense and national security*, Accessed: 2025-04-27, 2022. [Online]. Available: https://aws.amazon.com/blogs/iot/increase-military-readiness-with-aws-iot-for-defense-and-national-security/.

[29] Amazon Web Services, *Understand coordinate systems and sensor fusion*, Accessed: 2025-04-27, 2023. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/sms-point-cloud-sensor-fusion-details.html.

[30] Amazon Web Services, *What is a landing zone? - aws prescriptive guidance*, https://docs.aws.amazon.com/prescriptive-guidance/latest/migration-aws-environment/understanding-landing-zones.html, Accessed: June 2025, 2023.

[31] Amazon Web Services, Inc. "High Performance Computing (HPC). "[Online]. Available: https://aws.amazon.com/hpc/.

[32] AMD. "AMD Secure Encrypted Virtualization (SEV) - AMD. "[Online]. Available: https://developer.amd.com/sev/.

[33] "Azure Confidential Computing – Protect Data In Use. "[Online]. Available: https://azure.microsoft.com/en-us/solutions/confidential-compute/#overview.

[34] K. Chen, "Confidential high-performance computing in the public cloud," *IEEE Internet Computing*, vol. 27, pp. 24–32, 1 Jan. 2023, ISSN: 19410131. DOI: 10.1109/MIC.2022.3226757.

[35] Confidential Computing Consortium. "Confidential Computing Whitepapers. "[Online]. Available: https://www.confidentialcomputing.io/white-papers-reports/.

[36] B. S. Institution, *ISO/IEC 22123-2:2023 - Information technology - Cloud computing - Part 2: Concepts*. London: British Standards Institution, 2023.

[37] Intel. "Confidential Computing." en. [Online]. Available: https://www.intel.com/content/www/us/en/security/confidential-computing.html.

[38] OpenStack. "Open Source Cloud Computing Infrastructure - OpenStack. "[Online]. Available: https://www.openstack.org/.

[39] V. Pfeil, "Seminar work: High performance computing: Trusted execution environments," University of the Bundeswehr, Department of Computer Science, Institute for Software Technology, Tech. Rep., Mar. 2023. [Online]. Available: https://www.unibw.de/inf2.

[40] A. Susanto, A. H. Fathulloh, Nuryasin, and A. Fitriyani, "Comparative analysis of key management service performance on aws, google cloud, and oracle cloud with performance testing," *2023 11th International Conference on Cyber and IT Service Management, CITSM 2023*, 2023. DOI: 10.1109/CITSM60085.2023.10455569.

[41] VMware. "Virtualizing High Performance Computing. "[Online]. Available: https://www.vmware.com/uk/solutions/high-performance-computing.html.

[42] S. Zobaed and M. A. Salehi, "Confidential computing across edge-to-cloud for machine learning: A survey study," Jul. 2023. [Online]. Available: https://arxiv.org/abs/2307.16447v1.

[43] "AC/322-D(2021)0032-REV1: Technical Directive on Cloud-Based Handling of NATO-Classified Information," NATO Consultation, Command and Control Board (C3B), Brussels, Tech. Rep. AC/322-D(2021)0032-REV1, Mar. 2024, Releasable to NATO nations and partners.

[44] Amazon Web Services, *Aws well-architected framework*, https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html, Last accessed May 2025, 2024.

[45] Amazon Web Services, *aws-nitro-enclaves-sdk-c: C SDK for AWS Nitro Enclaves*, https://github.com/aws/aws-nitro-enclaves-sdk-c, Accessed: 2025-06-22, 2024.

[46] M. Brossard et al., "Private delegated computations using strong isolation," *IEEE Transactions on Emerging Topics in Computing*, vol. 12, pp. 386–398, 1 2024, ISSN: 21686750. DOI: 10.1109/TETC.2023.3281738.

[47] Cloud Native Computing Foundation, *Confidential containers project*, CNCF Project Website, https://confidentialcontainers.org/, 2024.

[48] J. Gracewell, R. Santhosh, and V. Sabarish, "Image fusion for improved situational awareness in military operations using machine learning," *2024 2nd International Conference on Advances in Computation, Communication and Information Technology, ICAICCIT 2024*, pp. 733–737, 2024. DOI: 10.1109/ICAICCIT64383.2024.10912129.

[49] H. Hussain et al., "Energy efficient real-time tasks scheduling on high-performance edge-computing systems using genetic algorithm," *IEEE Access*, vol. 12, pp. 54 879–54 892, 2024, ISSN: 21693536. DOI: 10.1109/ACCESS.2024.3388837.

[50] J. R. Kancharla and S. D. M. Kumar, "Advancing data sovereignty in distributed environments: An in-depth exploration of data localization challenges," *2024 International Conference on Computer, Electronics, Electrical Engineering and their Applications, IC2E3 2024*, 2024. DOI: 10.1109/IC2E362166.2024.10827688.

[51] A. Karcher, "Enterprise architecture and it service management (eaitsm) - kapitel 3: The open group architecture framework (togaf)," Institut für Softwaretechnik, Universität der Bundeswehr München, Technical Report, 2024.

[52] A. Karcher, "Enterprise architecture and it service management (eaitsm) - kapitel 5: Archimate," Institut für Softwaretechnik, Universität der Bundeswehr München, Technical Report, 2024.

[53] A. Karcher, "Enterprise architecture and it service management (eaitsm) - kapitel 7: Nato architecture framework (naf)," Institut für Softwaretechnik, Universität der Bundeswehr München, Technical Report, 2024.

[54] H. Li, Z. Wu, F. Yin, C. Yang, and X. Yang, "A study on risk situational awareness of power iot distribution scenarios under cloud-edge collaboration," *2024 4th International Conference on Intelligent Power and Systems, ICIPS 2024*, pp. 824–828, 2024. DOI: 10.1109/ICIPS64173.2024.10900147.

[55] R. Nagarkar, C. Bennie, K. Wang, M. Lam, D. Gonzalez, and M. B. Chaudhari, "Integrating multiple cloud platforms to build a data pipeline for recommendation systems," *Proceedings - 2024 7th International Conference on Data Science and Information Technology, DSIT 2024*, 2024. DOI: 10.1109/DSIT61374.2024.10881634.

[56] V. Pfeil, "Bachelor thesis: Confidential computing via hardware trusted execution environments by an openstack hpc capable cloud," University of the Bundeswehr, Department of Computer Science, Institute for Software Technology, Tech. Rep., Jan. 2024.

[57] V. Pfeil, "Internship report: Optimisation of iav ai cloud infrastructure," University of the Bundeswehr, Department of Computer Science, Institute for Software Technology, Tech. Rep., Oct. 2024.

[58] F. A. Saif, R. Latip, Z. M. Hanapi, K. Shafinah, A. V. S. Kumar, and A. S. Bajaher, "Multi-objectives firefly algorithm for task offloading in the edge-fog-cloud computing," *IEEE Access*, 2024, ISSN: 21693536. DOI: 10.1109/ACCESS.2024.3488032.

[59] I. Satoh, "A general-purpose middleware system for edge-side data processing," *2024 9th International Conference on Fog and Mobile Edge Computing, FMEC 2024*, pp. 190–195, 2024. DOI: 10.1109/FMEC62297.2024.10710251.

[60] J. Thijsman, M. Sebrechts, F. D. Turck, and B. Volckaert, "Trusting the cloud-native edge: Remotely attested kubernetes workers," *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, May 2024, ISSN: 10952055. DOI: 10.1109/ICCCN61486.2024.10637515. [Online]. Available: https://arxiv.org/abs/2405.10131v1.

[61] L. Wilke and G. Scopelliti, "Snpguard: Remote attestation of sev-snp vms using open source tools," *Proceedings - 9th IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2024*, pp. 193–198, 2024. DOI: 10.1109/EUROSPW61312.2024.00026.

[62] X. Zhao, "Design and implementation of campus data governance platform based on big data algorithm," *Proceedings - 2024 Asia-Pacific Conference on Software Engineering, Social Network Analysis and Intelligent Computing, SSAIC 2024*, pp. 391–396, 2024. DOI: 10.1109/SSAIC61213.2024.00080.

[63] Amazon Web Services, *Amd sev-snp for amazon ec2 instances*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html, Accessed: 2025-06-22, 2025.

[64] Amazon Web Services. "Aws european sovereign cloud: Standort brandenburg für erste region in deutschland angekündigt." Accessed: 2025-06-07. [Online]. Available: https://www.aboutamazon.de/news/amazon-web-services/aws-european-sovereign-cloud-brandenburg.

[65] Amazon Web Services. "Introducing the aws european sovereign cloud." Accessed April 2025. [Online]. Available: https://aws.amazon.com/compliance/europe-digital-sovereignty/.

[66] R. Baldoni and G. D. Luna, "Sovereignty in the digital era: The quest for continuous access to dependable technological capabilities," *IEEE Security and Privacy*, vol. 23, pp. 91–96, 1 2025, ISSN: 15584046. DOI: 10.1109/MSEC.2024.3500192.

[67] S. Batewela, M. Liyanage, E. Zeydan, M. Ylianttila, and P. Ranaweera, "Security orchestration in 5g and beyond smart network technologies," *IEEE Open Journal of the Computer Society*, pp. 1–20, 2025, ISSN: 2644-1268. DOI: 10.1109/OJCS.2025.3563619. [Online]. Available: https://ieeexplore.ieee.org/document/10974672/.

[68] Bundeswehr, *Offizielle website der bundeswehr*, https://www.bundeswehr.de, Zugriff am 24.06.2025, 2025. [Online]. Available: https://www.bundeswehr.de.

[69] H. Chen, Y. Tang, A. Tsourdos, and W. Guo, "Contextualized autonomous drone navigation using llms deployed in edge-cloud computing," *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*, pp. 1373–1378, Mar. 2025. DOI: 10.1109/ICMLAS64557.2025.10967934. [Online]. Available: https://ieeexplore.ieee.org/document/10967934/.

[70] Confidential Computing Consortium, *About the confidential computing consortium*, https://confidentialcomputing.io/about/, Last accessed May 2025, 2025.

[71] D. Khan, S. Aslam, and K. Chang, "Vehicle-to-infrastructure multi-sensor fusion (v2i-msf) with reinforcement learning framework for enhancing autonomous vehicle perception," *IEEE Access*, 2025, ISSN: 21693536. DOI: 10.1109/ACCESS.2025.3551367.

[72] S. Mhatre, V. J. Dongre, and S. Mande, "Enhancing edge performance: A comparative analysis of lstm inference using hardware acceleration," *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*, pp. 1569–1574, Mar. 2025. DOI: 10.1109/ICMLAS64557.2025.10968872. [Online]. Available: https://ieeexplore.ieee.org/document/10968872/.

[73] D. Minott and S. S. et. al., "Benchmarking edge ai platforms: Performance analysis of nvidia jetson and raspberry pi 5 with coral tpu," *SoutheastCon 2025*, pp. 1384–1389, Mar. 2025. DOI: 10.1109/SOUTHEASTCON56624.2025.10971592.

[74] NATO Architecture Capability Team, "Archimate modeling guide for the nato architecture framework version 4," NATO Digital Policy Committee, Tech. Rep., 2025, Official NATO Publication.

[75] C. Nilsson and V. Pfeil, *Omniaware pr/faq - strategic concept and platform vision*, PR/FAQ, Capgemini Internal, 2025.

[76] V. Pfeil, *Omniaware use case - contextual image verification system (civs)*, Business Process Model, Capgemini Internal, 2025.

[77] V. Pfeil, *Omniaware use case - contextual image verification system (civs) - capabilities (bpm)*, Business Process Model, Capgemini Internal, 2025.

[78] V. Pfeil, *Omniaware use case - prognostics and health management (phm)*, Business Process Model, Capgemini Internal, 2025.

[79] V. Pfeil, *Omniaware use case - prognostics and health management (phm) - capabilities (bpm)*, Business Process Model, Capgemini Internal, 2025.

[80] J. Salvermoser and V. Pfeil, *Omniaware use case - reference architecture*, Reference Architecture, Capgemini Internal, 2025.

[81] *Guidance for trusted secure enclaves on aws*. [Online]. Available: https://aws.amazon.com/solutions/guidance/trusted-secure-enclaves-on-aws/?nc1=h_ls.

[82] *Nato - nato architecture framework, version 4*. [Online]. Available: https://www.nato.int/cps/en/natohq/topics_157575.htm.

[83] *November 2024 | top500*. [Online]. Available: https://top500.org/lists/top500/2024/11/.

[84] *What is fog computing? - definition from iotagenda*. [Online]. Available: https://www.techtarget.com/iotagenda/definition/fog-computing-fogging.

MODEL DESCRIPTIONS

Table 1: NSV-4: Capability Dependencies

Source	Target	Relation Type	Justification
C1_Cloud Computing Platform	C10_Vehicle Health Analytics	serves	Provides scalable compute resources for health analysis microservices.
C1_Cloud Computing Platform	C16_Tactical Situational Awareness	serves	Hosts scalable dashboard services for PHM tactical visualisation.
C1_Cloud Computing Platform	C20_Streaming Imagery Ingestion	serves	Enables scalable image pipelines in cloud-native CIVS environments.
C1_Cloud Computing Platform	C23_Tactical Situational Awareness Visualisation	serves	Enables scalable and real-time data rendering in visualisation dashboards.
C2_Sensor Data Ingestion	C10_Vehicle Health Analytics	serves	Provides raw telemetry data for vehicle condition monitoring.
C2_Sensor Data Ingestion	C21_Weather Pattern Recognition	serves	Ingests weather-related data from sensor streams.
C3_Data Normalisation/Pre-Processing	C10_Vehicle Health Analytics	serves	Ensures clean, structured input for analysis models.
C3_Data Normalisation/Pre-Processing	C22_Analyst Feedback Loop	serves	Prepares data for effective human-in-the-loop refinement.
C4_Confidential Computing/Data Sovereignty	C13_Health-Based Task Prioritisation	serves	Protects sensitive health data used for decision support.
C4_Confidential Computing/Data Sovereignty	C16_Tactical Situational Awareness	serves	Ensures the secure deployment of dashboards containing classified PHM data.
C4_Confidential Computing/Data Sovereignty	C23_Tactical Situational Awareness	serves	Ensures secure deployment of dashboards in classified ops.
C5_NATO Classification Processing	C15_Occupancy and Crew State Monitoring	serves	Enables compliance with NATO classification for sensitive mission data.
C5_NATO Classification Processing	C24_Mission Impact Prediction	serves	Enforces policy-compliant processing of simulation outputs.
C12_Sensor Fusion	C10_Vehicle Health Analytics	serves	Consolidates sensor signals for accurate health evaluation.
C10_Vehicle Health Analytics	C11_Predictive Maintenance	serves	Provides historical data for forecasting future failures.
C10_Vehicle Health Analytics	C14_Tactical Vehicle Survivability	serves	Supplies live status for survivability prediction.
C10_Vehicle Health Analytics	C16_Tactical Situational Awareness	serves	Provides live vehicle condition data to inform the tactical dashboard.
C15_Occupancy and Crew State Monitoring	C14_Tactical Vehicle Survivability	triggers	Passenger/crew metrics trigger recalculation of survivability under tactical constraints.
C15_Occupancy and Crew State Monitoring	C16_Tactical Situational Awareness	serves	Supplies crew and passenger information to contextualise situational awareness.
C20_Streaming Imagery Ingestion	C23_Tactical Situational Awareness	aggregates	Provides live visual input for dashboard visualisation.
C20_Streaming Imagery Ingestion	C24_Mission Impact Prediction	aggregates	Supplies mission-relevant visual data to simulations.
C21_Weather Pattern Recognition	C23_Tactical Situational Awareness Visualisation	serves	Feeds predictive weather insights into the tactical decision-making interface.
C21_Weather Pattern Recognition	C24_Mission Impact Prediction	serves	Injects weather models for simulating environmental impact.
C22_Analyst Feedback Loop	C21_Weather Pattern Recognition	serves	Analyst insights serve as refined input for weather model improvement.
C22_NATO Classification Processing	C23_Tactical Situational Awareness Visualisation	serves	Provides classified object intelligence to support situational rendering.

Table 2: NSOV-3: Service Functions, Shared Core Services

Service Function	Description
SF1_Confidential Data Ingestion	Encrypted and attested input pipeline for telemetry, imagery and metadata across use cases. Supports multi-source sensor data via IoT protocols (e.g. MQTT, gRPC).
SF2_Confidential Computing Orchestration	Orchestration layer for workload scheduling across TEEs (Nitro Enclaves, AMD SEV-SNP), incl. remote attestation and enclave management.
SF3_Secure Storage and Access Layer	Confidential storage abstraction (e.g. encrypted S3, EBS) with metadata binding to STANAG 4774/4778 classification policies.
SF4_Multi-Level Security API Gateway	Zero-trust compatible API layer supporting multi-domain cross-classification routing and policy enforcement.
SF5_NATO Classification Processing	Core inference pipeline for classification and redaction of mission data, incl. image, telemetry and logs. Uses containerised AI model service.
SF6_Audit/Provenance Service	Cryptographically timestamped event logging across workloads (data access, decisions, model runs), with support for mission forensics.
SF7_Sovereign Policy Enforcement Engine	Rule-based engine that validates all data and service interactions against national and NATO policy bindings.
SF8_Federated Identity Trust Broker	SAML/OpenID-compatible federation hub to mediate identity validation across national entities and mission domains.
SF9_Metadata Management	Distributed metadata management layer to associate mission data with provenance, classification, encryption state and processing policies. Enables data tagging and federation across multi-domain systems.

Table 3: NSOV-3: Service Functions, PHM

Service Function	Description
SF10_Confidential Telemetry Collection	Real-time encrypted capture of vehicle sensor states (engine, chassis, onboard diagnostics).
SF11_Health Analytics Orchestration	Step-based pipeline for scoring and categorising vehicle health anomalies and degradation trends.
SF12_Fault Detection/-Root Cause Analysis	Event-driven anomaly analysis on confidential data using edge-deployed models.
SF13_Insider Presence Monitoring	Detection of seat occupancy and biometric-based driver/passenger identification.
SF14_Vehicle Survivability Estimation	Estimation of operational lifespan under tactical constraints, based on real-time health telemetry.
SF15_Telemetry Provenance Validation	Enforced source integrity verification from data origination point.
SF16_Tactical Health Visualisation	Overlay of health confidence scores onto mission dashboards for mobile or command units.
SF17_Tactical Travel Time Estimator	Estimate of mission-compliant travel duration based on current faults, health trends and personnel status.
SF18_Mission Data Classifier	Classification-aware routing and authorisation logic for telemetry workloads, triggered by enclave attestation and enforced through policy-bound key release.
SF19_Data Object Storage Governance	Ensures secure, policy-driven object storage with KMS-based encryption, compliance tagging and immutable retention policies for classified mission data in S3. Triggers downstream processing events based on data lifecycle transitions and integrity checks.

Table 4: NSOV-3: Service Functions, CIVS

Service Function	Description
SF20_Streaming Imagery Ingestion	Real-time ingestion pipeline for tactical video feeds and satellite stills.
SF21_Sensor Fusion/Spatial Correlation	Integration of visual data with geolocation, time and other contextual metadata.
SF22_Weather Pattern Recognition	AI-driven classification of weather conditions based on image feeds and public datasets (e.g. Copernicus).
SF23_Analyst Feedback Loop	Feedback tagging pipeline to tune model accuracy based on human analyst inputs.
SF24_Contextual Classification Refinement	Context-aware classification refinement service using temporal and spatial priors.
SF25_Tactical Image Overlay Service	Rendering of image overlays for live situational maps and dashboards.
SF26_Mission-Based Tagging Pipeline	Event-tagging of image segments aligned with mission identifiers and classification scope.
SF27_Edge-Compatible Inference Proxy	Local gateway service optimised for low-bandwidth model execution at the tactical edge.

Table 5: NSOV-2: Service Interfaces, PHM

Service Interface	Description
SI1_PHM Ingest Gateway	Secure service ingress interface for mission telemetry, enforcing TLS mutual authentication and classification tag validation.
SI2_Vault Attestation Key Gate	Interface for policy-controlled key release from Vault, gated via SPIRE attestation and workload-bound identity tokens.
SI3_Telemetry Classification API	Interface for binding telemetry metadata to classification levels and mission context before further routing.
SI4_Secure Forwarding Endpoint	Policy-bound forwarding interface for compliant data transfer to internal analytic pipelines.

Table 6: NSOV-2: Service Interfaces, CIVS

Service Interface	Description
SI20_CIVS Image Ingest API	Entry interface for streaming tactical imagery and satellite stills, including pre-ingest validation.
SI21_Classification Overlay Service	API for assigning classification overlays (e.g. NATO Restricted) to image segments.
SI22_Mission Tagging Interface	Interface for mission-aware tagging of visual data, linking image segments with classification and operational metadata.
SI23_Secure Analyst Feedback Channel	Feedback interface enabling analysts to update tagging and classification annotations under audit controls.

DEPLOYMENT - CI/CD-PIPELINE

init_stack.yaml

Defines IAM roles and trust policies to bootstrap StackSet permissions.

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: "This stack contains initial resources e.g. IAM roles, policies, etc. that are required in all target accounts of our CI/CD solution."
Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
    Description: "Name of the application the resources belong to"
  Stage:
    Type: "String"
    Default: "dev"
    Description: "The stage. E.g. 'dev' or 'prod'"
  Prefix:
    Type: "String"
    Default: "omniaware"
    Description: "A prefix used for resource naming. E.g. S3 Bucket prefix."
  # The following parameters are used to specify the source account and role that will assume this role.
  # Please use with caution and ensure that the source account and role are correctly set.
  SourceAccountId:
    Type: "String"
    NoEcho: true
    Description: "The AWS account ID of the account assuming this role."
  SourceAccountCodeBuildRoleName:
    Type: "String"
    Description: "The name of the role in the source account used by CodeBuild."
  StackSetsAdminRoleNameSuffix:
    Type: "String"
    Default: "CustomStackSetsAdminRole"
    Description: "The name of the role in the source account (without application-stage prefix) that will assume this role."

Mappings:
  IAM:
    Roles:
      PermissionsBoundaryPolicy: "GroupIT_SecurityGroupPermissionBoundary"

Resources:
  CrossAccountCloudFormationStackSetsRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: !Sub "${Application}-CustomStackSetsExecutionRole"
      Path: "/"
      PermissionsBoundary: !Join
        - ""
        - - !Sub "arn:aws:iam::${AWS::AccountId}:policy/"
          - !FindInMap [IAM, Roles, PermissionsBoundaryPolicy]
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
```

```
47        Statement:
48          - Effect: "Allow"
49            Principal:
50              AWS: !Sub "arn:aws:iam::${SourceAccountId}:role/${Application}-${StackSe
                ↪  tsAdminRoleNameSuffix}"
51            Action: "sts:AssumeRole"
52    CloudFormationDeploymentPolicy:
53      Type: "AWS::IAM::Policy"
54      Properties:
55        PolicyName: !Sub "${Application}-CustomStackSetsExecutionRolePolicy"
56        Roles:
57          - !Ref CrossAccountCloudFormationStackSetsRole
58        PolicyDocument:
59          Version: "2012-10-17"
60          Statement:
61            - Effect: "Allow"
62              Action:
63                - "cloudformation:CreateStack"
64                - "cloudformation:UpdateStack"
65                - "cloudformation:DeleteStack"
66                - "cloudformation:DescribeStackResources"
67                - "cloudformation:DescribeStacks"
68                - "cloudformation:ListStackResources"
69                - "cloudformation:DescribeStackResource"
70                - "cloudformation:ListStacks"
71                - "cloudformation:DescribeChangeSet"
72                - "cloudformation:DescribeStackSet"
73                - "cloudformation:GetTemplateSummary"
74                - "cloudformation:CreateChangeSet"
75                - "cloudformation:ExecuteChangeSet"
76              Resource: !Sub "arn:aws:cloudformation:${AWS::Region}:${AWS::AccountId}:*"
77    CrossAccountCloudFormationStackRole:
78      Type: "AWS::IAM::Role"
79      Properties:
80        RoleName: !Sub "${Application}-CrossAccountStacksRole"
81        Path: "/"
82        PermissionsBoundary: !Join
83          - ""
84          - - !Sub "arn:aws:iam::${AWS::AccountId}:policy/"
85            - !FindInMap [IAM, Roles, PermissionsBoundaryPolicy]
86        AssumeRolePolicyDocument:
87          Version: "2012-10-17"
88          Statement:
89            - Effect: "Allow"
90              Principal:
91                AWS: !Sub "arn:aws:iam::${SourceAccountId}:role/${Application}-${SourceA
                  ↪  ccountCodeBuildRoleName}"
92              Action: "sts:AssumeRole"
93    CrossAccountCloudFormationStackRolePolicy:
94      Type: "AWS::IAM::Policy"
95      Properties:
96        PolicyName: !Sub "${Application}-CrossAccountStacks-Policy"
97        Roles:
98          - !Ref CrossAccountCloudFormationStackRole
99        PolicyDocument:
100         Version: "2012-10-17"
101         Statement:
102           - Effect: "Allow"
103             Action:
104               - "cloudformation:CreateStack"
105               - "cloudformation:UpdateStack"
```

```
106               - "cloudformation:DeleteStack"
107               - "cloudformation:DescribeStackResources"
108               - "cloudformation:DescribeStacks"
109               - "cloudformation:ListStackResources"
110               - "cloudformation:DescribeStackResource"
111               - "cloudformation:ListStacks"
112               - "cloudformation:DescribeChangeSet"
113               - "cloudformation:GetTemplateSummary"
114               - "cloudformation:CreateChangeSet"
115               - "cloudformation:ExecuteChangeSet"
116             Resource: !Sub "arn:aws:cloudformation:${AWS::Region}:${AWS::AccountId}:*"
117     # This policy allows the role to create and manage resources.
118     # After an initial development phase, this policy should be limited to only the
        ↪  resources that are needed.
119     CrossAccountResourceProvisioningPolicy:
120       Type: "AWS::IAM::Policy"
121       Properties:
122         PolicyName: !Sub "${Application}-CrossAccountResourceProvisioning-Policy"
123         Roles:
124           - !Ref CrossAccountCloudFormationStackRole
125           - !Ref CrossAccountCloudFormationStackSetsRole
126         PolicyDocument:
127           Version: "2012-10-17"
128           Statement:
129             - Effect: "Allow"
130               Action:
131                 - "kms:CancelKeyDeletion"
132                 - "kms:CreateAlias"
133                 - "kms:CreateKey"
134                 - "kms:Decrypt"
135                 - "kms:DeleteAlias"
136                 - "kms:DescribeKey"
137                 - "kms:DisableKeyRotation"
138                 - "kms:EnableKeyRotation"
139                 - "kms:Encrypt"
140                 - "kms:GenerateDataKey*"
141                 - "kms:GetKeyPolicy"
142                 - "kms:ListAliases"
143                 - "kms:ListKeys"
144                 - "kms:ListResourceTags"
145                 - "kms:PutKeyPolicy"
146                 - "kms:ReEncrypt*"
147                 - "kms:ScheduleKeyDeletion"
148                 - "kms:TagResource"
149                 - "kms:UntagResource"
150                 - "kms:UpdateAlias"
151               Resource: !Sub "arn:aws:kms:${AWS::Region}:${AWS::AccountId}:*"
152             - Effect: "Allow"
153               Action:
154                 - "s3:CreateBucket"
155                 - "s3:DeleteBucket"
156                 - "s3:DeleteObject"
157                 - "s3:GetBucketPolicy"
158                 - "s3:GetBucketTagging"
159                 - "s3:GetEncryptionConfiguration"
160                 - "s3:GetObject"
161                 - "s3:ListAllMyBuckets"
162                 - "s3:ListBucket"
163                 - "s3:PutBucketPolicy"
164                 - "s3:PutBucketTagging"
165                 - "s3:PutEncryptionConfiguration"
```

```
166               - "s3:PutObject"
167             Resource: "arn:aws:s3:::*"
168           - Effect: "Allow"
169             Action:
170               - "logs:CreateLogGroup"
171               - "logs:CreateLogStream"
172               - "logs:DeleteLogGroup"
173               - "logs:DeleteLogStream"
174               - "logs:DescribeLogGroups"
175               - "logs:DescribeLogStreams"
176               - "logs:PutLogEvents"
177               - "logs:PutRetentionPolicy"
178               - "logs:TagLogGroup"
179               - "logs:UntagLogGroup"
180             Resource: !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-group:*"
181           - Effect: "Allow"
182             Action:
183               - "lambda:CreateAlias"
184               - "lambda:CreateFunction"
185               - "lambda:DeleteAlias"
186               - "lambda:DeleteFunction"
187               - "lambda:GetFunction"
188               - "lambda:ListFunctions"
189               - "lambda:TagResource"
190               - "lambda:UntagResource"
191               - "lambda:UpdateAlias"
192               - "lambda:UpdateFunctionCode"
193               - "lambda:UpdateFunctionConfiguration"
194             Resource: !Sub "arn:aws:lambda:${AWS::Region}:${AWS::AccountId}:function:*"
195           - Effect: "Allow"
196             Action:
197               - "iam:AttachRolePolicy"
198               - "iam:CreatePolicy"
199               - "iam:CreateRole"
200               - "iam:DeletePolicy"
201               - "iam:DeleteRole"
202               - "iam:DeleteRolePolicy"
203               - "iam:DetachRolePolicy"
204               - "iam:GetPolicy"
205               - "iam:GetRole"
206               - "iam:GetRolePolicy"
207               - "iam:ListAttachedRolePolicies"
208               - "iam:ListRolePolicies"
209               - "iam:PassRole"
210               - "iam:PutRolePolicy"
211               - "iam:TagRole"
212               - "iam:UntagRole"
213               - "iam:UpdateRole"
214             Resource: !Sub "arn:aws:iam::${AWS::AccountId}:*"
215           - Effect: "Allow"
216             Action:
217               - "ec2:AuthorizeSecurityGroupEgress"
218               - "ec2:CreateNetworkInterface"
219               - "ec2:CreateNetworkInterfacePermission"
220               - "ec2:CreateSecurityGroup"
221               - "ec2:DeleteNetworkInterface"
222               - "ec2:DeleteSecurityGroup"
223               - "ec2:DescribeNetworkInterfaces"
224               - "ec2:DescribeSecurityGroups"
225               - "ec2:RevokeSecurityGroupEgress"
226             Resource: "*"
```

```
          - Effect: "Allow"
            Action:
              - "secretsmanager:CreateSecret"
              - "secretsmanager:DeleteSecret"
              - "secretsmanager:DescribeSecret"
              - "secretsmanager:GetSecretValue"
              - "secretsmanager:ListSecrets"
              - "secretsmanager:PutSecretValue"
              - "secretsmanager:TagResource"
              - "secretsmanager:UntagResource"
            Resource: !Sub
            ↪  "arn:aws:secretsmanager:${AWS::Region}:${AWS::AccountId}:secret:*"

Outputs:
  CrossAccountCloudFormationStackRoleArn:
    Description: "The ARN of the cross-account CloudFormation stack role"
    Value: !GetAtt CrossAccountCloudFormationStackRole.Arn
    Export:
      Name: !Sub 'CrossAccountCloudFormationStackRoleArn'
  CrossAccountCloudFormationStackSetsRoleArn:
    Description: "The ARN of the cross-account CloudFormation stack sets role"
    Value: !GetAtt CrossAccountCloudFormationStackSetsRole.Arn
    Export:
      Name: !Sub 'CrossAccountCloudFormationStackSetsRoleArn'
```

1

parameters.json
Provides parameter defaults for the modular deployment structure.

```
[
  {
    "ParameterKey": "SourceAccountId",
    "ParameterValue": "<SourceAccountId>"
  },
  {
    "ParameterKey": "SourceAccountCodeBuildRoleName",
    "ParameterValue": "CodeBuildRole"
  },
  {
    "ParameterKey": "StackSetsAdminRoleName",
    "ParameterValue": "CustomStackSetsAdminRole"
  }
]
```

2 3

1 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.
2 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.
3 Certain elements in the code listing have been anonymised or generalised to preserve confidentiality and align with disclosure requirements.

deploy_stacks_wrapper.sh

Wrapper script to trigger ordered execution of infrastructure scripts.

```
#!/bin/bash

# This script is a wrapper script for ./deploy_stacks.sh.
# It takes a comma separated list of account names as argument and then calls ./deploy_stacks.sh for each account name.

####### IMPORTANT #######
# It is only meant to be used in the CI/CD pipeline and and only executed in the context of the `ingest` account.
# If the currently scoped account is not the same as the target account, only the IAM role of the CI/CD solution of the `ingest` account
# can assume a respective role in the target account.

set -eEuo pipefail

usage() {
  cat <<EOF
Usage: $0
Options:
  --account-names, -a "<account_name>,<account_name>,....,<account_name>": The names of the accounts to deploy to seperated by comma.
EOF
}

validate_inputs() {
  [[ -n "$account_names" ]] || { echo "Account names are required."; usage; exit 1; }
  [[ "$account_names" =~ ^([a-z]+)(,[a-z]+)*$ ]] \
    || { echo "Invalid account names format. Only lowercase words separated by commas are allowed."; exit 1; }
}

# The account names can be provied as environment variable or as command line argument.
account_names="${ACCOUNT_NAMES:-}"

while [[ $# -gt 0 ]]; do
  case "$1" in
    -a|--account-names)
      echo "Account names provided: $2"
      [[ -z "$2" ]] && { echo "Missing argument for $1"; usage; exit 1; }
      account_names="$2"
      shift 2
      ;;
    -h|--help)
      usage
      exit 0
      ;;
    *)
      echo "Unknown option: $1"
      usage
      exit 1
      ;;
  esac
done

validate_inputs

while read -r account; do
  echo "Executing deployment script for: ${account}"
  ./deploy_stacks.sh -a "${account}"
```

```
done < <(tr ',' '\n' <<< ${account_names})
```

4

deploy_stack-sets.sh
Main deployment script for applying stack definitions to all scoped accounts.

```
#!/bin/bash

# This script deploys the StackSets for the OmniAware project.
# It assumes that the AWS CLI is configured and that the user has the necessary permissions.

set -eEuo pipefail

usage() {
  cat <<EOF
Usage: $0
Options:
  --account-names, -a <account_names>: Comma-separated list of account names to deploy the StackSets to.
EOF
}

get_stack_set_operation_status() {
  local stack_set_name="$1"
  local operation_id="$2"
  aws cloudformation describe-stack-set-operation \
    --stack-set-name "$stack_set_name" \
    --operation-id "$operation_id" \
    --query 'StackSetOperation.Status' --output text
}

target_accounts="${TARGET_ACCOUNTS:-}"
region="${AWS_DEFAULT_REGION:-eu-central-1}"
application="${APPLICATION:-OmniAware}"
stage="${STAGE:-dev}"

readonly accounts_config="accounts.json"
[[ -f "$accounts_config" ]] || { echo "File $accounts_config not found"; exit 1; }

# Custom admin and execution role names for stack-sets
# These roles are used to manage stack-sets and stack-instances across accounts.
# The default names are based on the current implementation.
# The admin role must exist in the current account and the execution role must exist in the target accounts.
stack_sets_admin_role_name_default="$application-$stage-CloufFormationStackSets-Custom-Admin-Role"
stack_sets_execution_role_name_default="$application-$stage-CrossAccountStackSets-Role"
stack_sets_admin_role_name="${STACK_SETS_ADMIN_ROLE_NAME:-$stack_sets_admin_role_name_default}"
stack_sets_execution_role_name="${STACK_SETS_EXECUTION_ROLE_NAME:-$stack_sets_execution_role_name_default}"

while [[ $# -gt 0 ]]; do
  case "$1" in
    -a|--account-names)
      echo "Account names provided: $2"
      [[ -z "$2" ]] && { echo "Missing argument for $1"; usage; exit 1; }
```

4 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

```
      target_accounts="$2"
      shift 2
      ;;
    -h|--help)
      usage
      exit 0
      ;;
    *)
      echo "Unknown option: $1"
      usage
      exit 1
      ;;
  esac
done

[[ -n "$target_accounts" ]] || { echo "Target accounts are required."; usage; exit 1; }
[[ "$target_accounts" =~ ^([a-z]+)(,[a-z]+)*$ ]] \
  || { echo "Invalid target accounts format. Only lowercase words separated by commas
  ↪  are allowed."; exit 1; }

echo "Get current account ID"
current_account_id="$(aws sts get-caller-identity --query Account --output text)"

echo "###### Creating and updating stack-sets ######"
readonly stack_sets="$(aws cloudformation list-stack-sets --status 'ACTIVE' --query
↪  'Summaries[].StackSetName' --output text)"
echo "Existing stack-sets: $stack_sets"
while read -r template; do
  echo "Found template: $template"
  file_name="$(basename "$template" .yaml)"
  stack_set_name_in_file_name="$(sed 's/^[0-9]*_//g' <<< "$file_name" | tr '[:lower:]'
  ↪  '[:upper:]')"
  stack_set_name="${application}-${stack_set_name_in_file_name}-StackSet"
  # Check if stack-set already exists
  if [[ ! "$stack_sets" =~ "$stack_set_name" ]]; then
    echo "Creating stack-set: $stack_set_name with template: $template"
    aws cloudformation create-stack-set \
      --stack-set-name "${stack_set_name}" \
      --template-body "file://$template" \
      --capabilities CAPABILITY_NAMED_IAM \
      --administration-role-arn
      ↪  "arn:aws:iam::${current_account_id}:role/${stack_sets_admin_role_name}" \
      --execution-role-name "$stack_sets_execution_role_name" \
      --permission-model "SELF_MANAGED" > /dev/null
  else
    echo "Stack-set: $stack_set_name already exists. Skipping creation."
  fi

  # Create stack-instances for scoped accounts only
  scoped_accounts="$(jq -r '[.[] | select(.deploy_stack_set_instances == true) |
  ↪  .account_id] | join(" ")' $accounts_config)"
  echo "Creating stack-instance for target accounts: $scoped_accounts"
  operation_id="$(aws cloudformation create-stack-instances \
    --stack-set-name "${stack_set_name}" \
    --regions "${region}" \
    --accounts $scoped_accounts \
    --operation-preferences "{\"FailureToleranceCount\":3,\"MaxConcurrentCount\":5}" \
    | jq -r '.OperationId')"

  stack_set_operation_status="$(get_stack_set_operation_status "$stack_set_name"
  ↪  "$operation_id")"
```

```
102
103   # Wait for stack-instances creation to complete
104   while [[ ! $stack_set_operation_status =~ ^(SUCCEEDED|FAILED|STOPPED)$ ]]; do
105     echo "StackSet operation status: ${stack_set_operation_status}. Waiting for
      ↪  stack-instances creation to complete..."
106     sleep 5
107     stack_set_operation_status="$(get_stack_set_operation_status "$stack_set_name"
      ↪  "$operation_id")"
108   done
109
110   if [[ "$stack_set_operation_status" == "FAILED" ]]; then
111     echo "Stack-instances creation for stack-set $stack_set_name failed. Exiting."
112     exit 1
113   else
114     echo "Stack-instances creation for stack-set $stack_set_name completed with status
      ↪  $stack_set_operation_status."
115   fi
116
117   # Update stack-set
118   echo "Updating stack-set $stack_set_name with template: $template"
119   aws cloudformation update-stack-set \
120     --stack-set-name "${stack_set_name}" \
121     --template-body "file://$template" \
122     --capabilities CAPABILITY_NAMED_IAM \
123     --administration-role-arn
      ↪  "arn:aws:iam::${current_account_id}:role/${stack_sets_admin_role_name}" \
124     --execution-role-name "$stack_sets_execution_role_name" \
125     --permission-model "SELF_MANAGED" > /dev/null
126   echo "Updating stack-set $stack_set_name completed."
127
128 done < <(find './shared/stacksets' -name '*.yaml')
```

5

5 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

deploy_stacks-eu-west-1.sh
Region-specific deployment variant for 'eu-west-1'.

```
################################################################################
# Project:        OmniAware - Next-Gen Defence Platform
# Component:      Regional Deployment Script for CI/CD Stacks
# Script Name:    deploy_stacks-eu-west-1.sh
# Description:    Deploys all CloudFormation stacks in the specified directory
#                 for a given AWS account and template, assuming cross-account
#                 roles and supporting dynamic parameter overrides.
#
# Author:         Valentin Pfeil
# Institution:    University of the Bundeswehr Munich (M.Sc. Computer Science)
# Supervision:    Prof. Dr. Wolfgang Hommel / Dr. Karl Fuerlinger
# Date:           2025-06-21
# License:        Research Use Only / Academic Distribution, Subject to Future
↪  Publication
# Format:         Shell Script (Bash)
#
# Tags:           AWS CLI, CI/CD, CloudFormation, DevSecOps, Deployment Script
#
# Notes:
#   - Designed for deployments targeting AWS region eu-west-1 (Ireland)
#   - Supports dynamic template selection and parameter injection
#   - Includes role assumption logic for cross-account stack operations
#   - Integrates with AWS STS and jq for secure session token management
#
# Documentation:
#   - Master Thesis Appendix: CI/CD Pipeline Deployment Scripts
################################################################################

set -eEuo pipefail

usage() {
  cat <<EOF
Usage: $0
Options:
  --account-name, -a <account_name>:   The name of the account to deploy the stack or
  ↪  stacks to.
  --template, -t <template_file_path>: The relative path to a specific CloudFormation
  ↪  template file to deploy.

  NOTE: If no template is specified, all templates in the account's stacks directory
  ↪  will be deployed.
EOF
}

validate_inputs() {
  [[ -n "$account_name" ]] || { echo "Account name is required."; usage; exit 1; }
}

get_account_data() {
  local account_name="$1"
  local accounts_file="accounts.json"
  [[ -f "$accounts_file" ]] || { echo "File $accounts_file not found"; exit 1; }
  jq -c ".[] | select(.name == \"$account_name\")" accounts.json
}

assume_role() {
  local account_id="$1"
  local role_name="$2"
```

```
  local session_name="$3"

  echo "Assuming role $role_name in account $account_id"
  target_role_arn="arn:aws:iam::$account_id:role/$role_name"
  session_credentials="$(aws sts assume-role \
    --role-arn "$target_role_arn" \
    --role-session-name "$session_name" \
    --query 'Credentials.[AccessKeyId,SecretAccessKey,SessionToken]' \
    --output json)"
  export AWS_ACCESS_KEY_ID="$(jq -r '.[0]' <<< "$session_credentials")"
  export AWS_SECRET_ACCESS_KEY="$(jq -r '.[1]' <<< "$session_credentials")"
  export AWS_SESSION_TOKEN="$(jq -r '.[2]' <<< "$session_credentials")"
  echo "Assumed role $role_name in account $account_id"
}

deploy() {
  local template="$1"

  file_name="$(basename "$template" .yaml)"
  path="$(dirname "$template")"
  stack_name_from_file="$(sed 's/^[0-9]*_//g' <<< "$file_name" | tr '[:lower:]'
  ↪  '[:upper:]' | tr '_' '-')"
  stack_name="${application}-${stack_name_from_file}-Stack"

  echo "Deploying stack: $stack_name with template: $template"
  aws cloudformation deploy \
    --region "$region" \
    --template-file "$template" \
    --stack-name "${stack_name}" \
    --parameter-overrides "file://$path/parameters.json" \
    --capabilities CAPABILITY_NAMED_IAM;
}

# Variables and defaults
#OLD: region="${AWS_DEFAULT_REGION:-eu-central-1}"
region="${AWS_DEFAULT_REGION:-eu-west-1}"
application="${APPLICATION:-OmniAware}"
stage="${STAGE:-dev}"
account_name="${ACCOUNT_NAME:-}"
template_file_path="${TEMPLATE_FILE_PATH:-}"
cross_account_role="${CROSS_ACCOUNT_ROLE:-${application}-CrossAccountStacksRole}"

while [[ $# -gt 0 ]]; do
  case "$1" in
    -a|--account-name)
      echo "Account name: $2"
      [[ -z "$2" ]] && { echo "Missing argument for $1"; usage; exit 1; }
      account_name="$2"
      shift 2
      ;;
    -t|--template)
      echo "Template name: $2"
      [[ -z "$2" ]] && { echo "Missing argument for $1"; usage; exit 1; }
      [[ -f "$2" ]] || { echo "Template file $2 not found"; exit 1; }
      template_file_path="$2"
      shift 2
      ;;
    -h|--help)
      usage
      exit 0
      ;;
```

```
    *)
      echo "Unknown option: $1"
      usage
      exit 1
      ;;
  esac
done

account_data="$(get_account_data "$account_name")"
[[ -z "$account_data" ]] && { echo "Account with name $account_name not found in
↪  accounts.json"; exit 1; }

account_id="$(jq -r '.account_id' <<< "$account_data")"
current_account_id="$(aws sts get-caller-identity --query Account --output text)"

# Assume role in the target account if the account ID does not match the current
↪  account ID
if [[ ! "$account_id" =~ "$current_account_id" ]]; then
  echo "Account ID $account_id does not match current account ID $current_account_id.
  ↪  Assuming role in target account."
  assume_role "$account_id" "$cross_account_role" "$account_name"
else
  echo "Account ID $account_id matches current account ID $current_account_id. No role
  ↪  assumption needed."
fi

if [[ -n "$template_file_path" ]]; then
  echo "###### Deploying specific template: $template_file_path ######"
  deploy "$template_file_path"
  echo "###### Deployment of specific template completed ######"
  exit 0
fi

echo "###### Deploying stacks to account $account_name ($account_id) ######"
while read -r template; do
  deploy "$template"
done < <(find ./${account_name}/stacks -type f -name "*.yaml" | sort)
echo "###### Deploying stacks to account $account_name ($account_id) completed ######"
```

6

6 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

CORE INFRASTRUCTURE

00_kmsKeys.yaml

Defines the KMS key and alias for encrypted telemetry and data plane usage.

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: "A stack for KMS keys used in other accounts of the Organization"

Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
    Description: "Name of the application the resources belong to"
  Stage:
    Type: "String"
    Default: "dev"
    Description: "The stage. E.g. 'dev' or 'prod'"
  Prefix:
    Type: "String"
    Default: "omniaware"
    Description: "A prefix used for resource naming. E.g. S3 Bucket prefix."
  Region:
    Type: "String"
    Default: "eu-central-1"
    Description: "The region."
  AuditAccountId:
    Type: "String"
    Description: "AWS Account ID of the central audit account"
  SecurityAccountId:
    Type: "String"
    Description: "AWS Account ID of the central audit account"
  IngestAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  DatalakeAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  ConsumerAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"

Resources:
  AuditS3BucketKMSKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for encrypting audit logs"
      KeyPolicy:
        Id: AuditS3BucketKMSKeyPolicy
        Version: "2012-10-17"
        Statement:
          - Sid: AllowRootAccess
            Effect: Allow
            Principal:
              AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root"
            Action: kms:*
            Resource: "*"
          - Sid: Allow CloudTrail to encrypt logs
            Effect: Allow
            Principal:
              Service: cloudtrail.amazonaws.com
            Action: kms:GenerateDataKey*
```

```
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount":
                  - !Ref AWS::AccountId
                  - !Ref AuditAccountId
                  - !Ref SecurityAccountId
                  - !Ref IngestAccountId
                  - !Ref DatalakeAccountId
                  - !Ref ConsumerAccountId
          - Sid: Allow CloudTrail to describe key
            Effect: Allow
            Principal:
              Service: cloudtrail.amazonaws.com
            Action: kms:DescribeKey*
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount":
                  - !Ref AWS::AccountId
                  - !Ref AuditAccountId
                  - !Ref SecurityAccountId
                  - !Ref IngestAccountId
                  - !Ref DatalakeAccountId
                  - !Ref ConsumerAccountId
          - Sid: Allow cross-account log decryption
            Effect: Allow
            Principal:
              AWS:
                - !Sub "arn:aws:iam::${AWS::AccountId}:root"
                - !Sub "arn:aws:iam::${AuditAccountId}:root"
                - !Sub "arn:aws:iam::${SecurityAccountId}:root"
                - !Sub "arn:aws:iam::${IngestAccountId}:root"
                - !Sub "arn:aws:iam::${DatalakeAccountId}:root"
                - !Sub "arn:aws:iam::${ConsumerAccountId}:root"
            Action:
              - kms:Decrypt
              - kms:ReEncryptFrom
            Resource: "*"
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage
  AccesslogsS3BucketKMSKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for encrypting access logs"
      KeyPolicy:
        Id: AccessS3BucketKMSKeyPolicy
        Version: "2012-10-17"
        Statement:
          - Sid: AllowRootAccess
            Effect: Allow
            Principal:
              AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root"
            Action: kms:*
            Resource: "*"
          - Sid: Allow S3 to encrypt logs
```

```
            Effect: Allow
            Principal:
              Service: s3.amazonaws.com
            Action: kms:GenerateDataKey*
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount":
                  - !Ref AWS::AccountId
                  - !Ref AuditAccountId
                  - !Ref SecurityAccountId
                  - !Ref IngestAccountId
                  - !Ref DatalakeAccountId
                  - !Ref ConsumerAccountId
          - Sid: Allow S3 to describe key
            Effect: Allow
            Principal:
              Service: s3.amazonaws.com
            Action: kms:DescribeKey*
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount":
                  - !Ref AWS::AccountId
                  - !Ref AuditAccountId
                  - !Ref SecurityAccountId
                  - !Ref IngestAccountId
                  - !Ref DatalakeAccountId
                  - !Ref ConsumerAccountId
          - Sid: Allow cross-account log decryption
            Effect: Allow
            Principal:
              AWS:
                - !Sub "arn:aws:iam::${AWS::AccountId}:root"
                - !Sub "arn:aws:iam::${AuditAccountId}:root"
                - !Sub "arn:aws:iam::${SecurityAccountId}:root"
                - !Sub "arn:aws:iam::${IngestAccountId}:root"
                - !Sub "arn:aws:iam::${DatalakeAccountId}:root"
                - !Sub "arn:aws:iam::${ConsumerAccountId}:root"
            Action:
              - kms:Decrypt
              - kms:ReEncryptFrom
            Resource: "*"
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

Outputs:
  AuditS3BucketKMSKey:
    Description: ARN of the KMS key to encrypt S3 audit logs
    Value: !GetAtt AuditS3BucketKMSKey.Arn
    Export:
      Name: "AuditS3BucketKMSKey"
  AccesslogsS3BucketKMSKey:
    Description: ARN of the KMS key to encrypt S3 access logs
    Value: !GetAtt AccesslogsS3BucketKMSKey.Arn
    Export:
```

```
      Name: "AccesslogsS3BucketKMSKey"
```

7

00_kms_ingest.yaml
KMS key definition for ingest-related encryption scopes.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "A stack for KMS keys used in Ingest account"

Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
    Description: "Name of the application the resources belong to"
  Stage:
    Type: "String"
    Default: "dev"
    Description: "The stage. E.g. 'dev' or 'prod'"
  Prefix:
    Type: "String"
    Default: "omniaware"
    Description: "A prefix used for resource naming. E.g. S3 Bucket prefix."
  Region:
    Type: "String"
    Default: "eu-central-1"
    Description: "The region."
  KeyManagementIamRoleName:
    Type: "String"
    Description: "The IAM role name that will be allowed to manage the KMS key"
  IngestAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  IngestAccountAdminRoleName:
    Type: "String"
    Description: "The name of the admin role in the ingest account"

Resources:
  FirehoseDeliveryStreamKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for delivery streams in the Ingest account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
```

7 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

```
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow Create Grant From Ingest Account"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${IngestAccountId}:role/${IngestAccountAdminRoleName}"
            Action: "kms:CreateGrant"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "firehose.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref IngestAccountId
              Bool:
                "kms:GrantIsForAWSResource": true
  FirehoseDeliveryStreamKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/ingest/firehose"
      TargetKeyId: !Ref FirehoseDeliveryStreamKmsKey
  CloudwatchLogsKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for telemetry data logs in the Ingest account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          # Ref: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/encrypt-log-
          ↪  data-kms.html#cmk-permissions
          - Sid: "Allow Key usage for CloudWatch log groups and streams related to
          ↪  Telemetrydata ingestion in the Ingest account"
            Effect: "Allow"
            Principal:
              Service: !Sub "logs.${AWS::Region}.amazonaws.com"
            Action:
              - "kms:Encrypt"
              - "kms:Decrypt"
              - "kms:ReEncrypt*"
              - "kms:GenerateDataKey*"
              - "kms:Describe*"
            Resource: "*"
            Condition:
              ArnLike:
                "kms:EncryptionContext:aws:logs:arn":
                  - !Sub "arn:aws:logs:${Region}:${IngestAccountId}:log-group:/${Appli
                  ↪  cation}/TelemetryData/*"
```

```
                - !Sub "arn:aws:logs:${Region}:${IngestAccountId}:log-group:/${Appli
                ↪  cation}/ImageData/*"
  CloudwatchLogsKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/ingest/logs"
      TargetKeyId: !Ref CloudwatchLogsKmsKey
  LambdaKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for all Lambda functions in the Ingest account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow function usage"
            Effect: "Allow"
            Principal:
              AWS: !Sub "arn:aws:iam::${IngestAccountId}:root"
            Action:
              - "kms:Decrypt"
              - "kms:DescribeKey"
              - "kms:GenerateDataKey"
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref IngestAccountId
              ArnLike:
                "aws:SourceArn": !Sub
                ↪  "arn:aws:lambda:${Region}:${IngestAccountId}:function:*"
          - Sid: "Allow Ingest account admin to create grant"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${IngestAccountId}:role/${IngestAccountAdminRoleName}"
            Action:
              - "kms:CreateGrant"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "lambda.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref IngestAccountId
              Bool:
                "kms:GrantIsForAWSResource": true
          - Sid: "Allow Ingest account admin to encrypt Lambda environment variables"
            Effect: "Allow"
            Principal:
```

```
              AWS: !Sub
              ↪  "arn:aws:iam::${IngestAccountId}:role/${IngestAccountAdminRoleName}"
            Action:
              - "kms:Encrypt"
              - "kms:ReEncrypt*"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "lambda.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref IngestAccountId
  LambdaKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/ingest/lambda"
      TargetKeyId: !Ref LambdaKmsKey
```

8

00_kms_datalake.yaml
Defines storage-layer KMS key for long-term encrypted data at rest.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "A stack for KMS keys used in DataLake account"

Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
    Description: "Name of the application the resources belong to"
  Stage:
    Type: "String"
    Default: "dev"
    Description: "The stage. E.g. 'dev' or 'prod'"
  Prefix:
    Type: "String"
    Default: "omniaware"
    Description: "A prefix used for resource naming. E.g. S3 Bucket prefix."
  Region:
    Type: "String"
    Default: "eu-central-1"
    Description: "The region."
  KeyManagementIamRoleName:
    Type: "String"
    Default: ""
    Description: "The IAM role name that will be allowed to manage the KMS key"
  IngestAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  DataLakeAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  ConsumerAccountId:
    Type: "String"
    Description: "AWS Account ID of the central ingest account"
  DataLakeAccountAdminRoleName:
    Type: "String"
    Description: "The name of the admin role in the data lake account"
  DataLakeAccountProcessTelemetryDataLambdaRoleNameSuffix:
```

8 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

```
38      Type: "String"
39      Description: "The suffix of the name of the Lambda role in the data lake account
        ↪ that handles new raw telemetry data"
40    DataLakeAccountImageClassificationLambdaRoleNameSuffix:
41      Type: "String"
42      Description: "The suffix of the name of the Lambda role in the data lake account
        ↪ that handles image classification"
43    DataLakeAccountImageExifExtractionLambdaRoleNameSuffix:
44      Type: "String"
45      Description: "The suffix of the name of the Lambda role in the data lake account
        ↪ that handles image EXIF extraction"
46    IngestAccountFirehoseIngestRoleNameSuffix:
47      Type: "String"
48      Description: "The name of the ingest role in the ingest account"
49    DataLakeAccountRawTelemetryBucketName:
50      Type: "String"
51      Description: "The name of the S3 bucket in the data lake account for raw telemetry
        ↪ data"
52    DataLakeAccountRawImagesBucketName:
53      Type: "String"
54      Description: "The name of the S3 bucket in the data lake account for raw telemetry
        ↪ data"
55
56  Resources:
57    S3RawDataKmsKey:
58      Type: "AWS::KMS::Key"
59      Properties:
60        Description: "KMS key for S3 Raw Telemetry data stored in the Data Lake account"
61        Enabled: true
62        EnableKeyRotation: true
63        KeySpec: "SYMMETRIC_DEFAULT"
64        KeyUsage: "ENCRYPT_DECRYPT"
65        MultiRegion: false
66        PendingWindowInDays: 30
67        # Overwriting default policy
68        KeyPolicy:
69          Version: "2012-10-17"
70          Statement:
71            # Default Statement. TODO: MUST be restricted. Details to be clarified.
72            - Sid: "Enable IAM User Permissions"
73              Effect: "Allow"
74              Principal:
75                AWS: !Sub
                  ↪ "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
76              Action: "kms:*"
77              Resource: "*"
78            # Allow the DataLake account admin role to use the key.
79            - Sid: "Allow"
80              Effect: "Allow"
81              Principal:
82                AWS: !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdmin
                  ↪ RoleName}"
83              Action:
84                - "kms:Decrypt"
85              Resource: "*"
86            # Allow Kinesis Firehose in the Ingest Account to use the key
87            # when storing data in the respective bucket in the Data Lake account.
88            # https://docs.aws.amazon.com/kms/latest/developerguide/encrypt_context.html
            ↪ #encryption-context-authorization
89            - Sid: "Enable cross account Kinesis Firehose access to the KMS key"
90              Effect: "Allow"
```

```
            Principal:
              # The specific principal role ARN is set in the condition
              # to allow only the Kinesis Firehose role in the Ingest account.
              AWS: !Sub "arn:aws:iam::${IngestAccountId}:root"
            Action:
              - "kms:Encrypt"
              - "kms:Decrypt"
              - "kms:ReEncrypt*"
              - "kms:GenerateDataKey*"
              - "kms:Describe*"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:CallerAccount": !Ref IngestAccountId
              ArnLike:
                "aws:PrincipalArn": !Sub "arn:aws:iam::${IngestAccountId}:role/${Appli
                ↪  cation}-${Stage}-${IngestAccountFirehoseIngestRoleNameSuffix}"
              ArnEquals:
                "kms:EncryptionContext:aws:s3:arn":
                  - !Sub "arn:aws:s3:::${DataLakeAccountRawTelemetryBucketName}/*"
                  - !Sub "arn:aws:s3:::${DataLakeAccountRawImagesBucketName}/*"
          - Sid: "Allow specific to use the key for S3 bucket downloads"
            Effect: "Allow"
            Principal:
              AWS:
                - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                ↪  keAccountProcessTelemetryDataLambdaRoleNameSuffix}"
                - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                ↪  keAccountImageClassificationLambdaRoleNameSuffix}"
                - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                ↪  keAccountImageExifExtractionLambdaRoleNameSuffix}"
            Action:
              - "kms:Decrypt"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "s3.${Region}.amazonaws.com"
              ArnEquals:
                "kms:EncryptionContext:aws:s3:arn":
                  - !Sub "arn:aws:s3:::${DataLakeAccountRawTelemetryBucketName}/*"
  S3RawDataKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/datalake/s3/raw"
      TargetKeyId: !Ref S3RawDataKmsKey
  SnsKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for SNS topics in the Data Lake account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
```

```
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow specific roles of the datalake account to use the key"
            Effect: "Allow"
            Principal:
              AWS:
                - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                ↪  oleName}"
            Action:
              - "kms:DescribeKey"
              - "kms:Decrypt"
              - "kms:GenerateDataKey"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "sns.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref DataLakeAccountId
          - Sid: "Allow SNS to use the key"
            Effect: "Allow"
            Principal:
              Service: "sns.amazonaws.com"
            Action:
              - "kms:Encrypt"
              - "kms:Decrypt"
              - "kms:ReEncrypt*"
              - "kms:GenerateDataKey*"
              - "kms:Describe*"
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref DataLakeAccountId
                "aws:sns:sourceAccount": !Ref DataLakeAccountId
              ArnLike:
                "aws:sns:topicArn": !Sub "arn:aws:sns:${Region}:${DataLakeAccountId}:*"
          # For EventBridge the the condition keys like aws:SourceAccount and
          ↪  aws:SourceArn are not supported.
          # https://docs.aws.amazon.com/sns/latest/dg/sns-key-management.html#sns-what
          ↪  -permissions-for-sse
          - Sid: "Allow EventBridge to use the key"
            Effect: "Allow"
            Principal:
              Service: "events.amazonaws.com"
            Action:
              - "kms:GenerateDataKey*"
              - "kms:Decrypt"
            Resource: "*"
  SnsKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/datalake/sns"
      TargetKeyId: !Ref SnsKmsKey
  SqsKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for SQS queues in the Data Lake account"
      Enabled: true
```

```
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow SQS to use the key"
            Effect: "Allow"
            Principal:
              Service: "sqs.amazonaws.com"
            Action:
              - "kms:Encrypt"
              - "kms:Decrypt"
              - "kms:ReEncrypt*"
              - "kms:GenerateDataKey*"
              - "kms:Describe*"
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref DataLakeAccountId
              ArnLike:
                "aws:SourceArn": !Sub "arn:aws:sqs:${Region}:${DataLakeAccountId}:*"
          - Sid: "Allow specific roles of the datalake account to use the key"
            Effect: "Allow"
            Principal:
              AWS:
                - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                ↪  oleName}"
            Action:
              - "kms:Decrypt"
              - "kms:DescribeKey"
              - "kms:GenerateDataKey"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "sqs.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref DataLakeAccountId
          - Sid: "Allow SNS to use the key"
            Effect: "Allow"
            Principal:
              Service: "sns.amazonaws.com"
            Action:
              - "kms:GenerateDataKey*"
              - "kms:Decrypt"
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref DataLakeAccountId
              ArnLike:
                "aws:SourceArn": !Sub "arn:aws:sns:${Region}:${DataLakeAccountId}:*"
```

```
263           - Sid: "Allow Lambda to use the key"
264             Effect: "Allow"
265             Principal:
266               AWS:
267                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                    ↪  keAccountProcessTelemetryDataLambdaRoleNameSuffix}"
268                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                    ↪  keAccountImageClassificationLambdaRoleNameSuffix}"
269                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                    ↪  keAccountImageExifExtractionLambdaRoleNameSuffix}"
270             Action:
271               - "kms:GenerateDataKey*"
272               - "kms:Decrypt"
273             Resource: "*"
274   SqsKmsKeyAlias:
275     Type: "AWS::KMS::Alias"
276     Properties:
277       AliasName: !Sub "alias/${Prefix}/datalake/sqs"
278       TargetKeyId: !Ref SqsKmsKey
279   EventBusKmsKey:
280     Type: "AWS::KMS::Key"
281     Properties:
282       Description: "KMS key for EventBridge in the Data Lake account"
283       Enabled: true
284       EnableKeyRotation: true
285       KeySpec: "SYMMETRIC_DEFAULT"
286       KeyUsage: "ENCRYPT_DECRYPT"
287       MultiRegion: false
288       PendingWindowInDays: 30
289       # Overwriting default policy
290       KeyPolicy:
291         Version: "2012-10-17"
292         Statement:
293           # Default Statement. TODO: MUST be restricted. Details to be clarified.
294           - Sid: "Enable IAM User Permissions"
295             Effect: "Allow"
296             Principal:
297               AWS: !Sub
                    ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
298             Action: "kms:*"
299             Resource: "*"
300           - Sid: "Allow specific roles of the datalake account to describe the key"
301             Effect: "Allow"
302             Principal:
303               AWS:
304                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                    ↪  oleName}"
305             Action:
306               - "kms:DescribeKey"
307             Resource: "*"
308           - Sid: "Allow specific roles of the datalake account to use the key for
              ↪  EventBridge Bus creation"
309             Effect: "Allow"
310             Principal:
311               AWS:
312                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                    ↪  oleName}"
313             Action:
314               - "kms:GenerateDataKey"
315             Resource: "*"
```

```
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "events.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref DataLakeAccountId
          # Ref: https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-encryption-key-policy.html#eb-encryption-event-bus-confused-deputy
          - Sid: "Allow EventBus to use the key"
            Effect: "Allow"
            Principal:
              Service: "events.amazonaws.com"
            Action:
              - "kms:Decrypt"
              - "kms:GenerateDataKey"
            Resource: "*"
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref DataLakeAccountId
              ArnLike:
                "aws:SourceArn": !Sub "arn:aws:events:${Region}:${DataLakeAccountId}:event-bus/*"
                "kms:EncryptionContext:aws:events:event-bus:arn": !Sub "arn:aws:events:${Region}:${DataLakeAccountId}:event-bus/*"
          # Allow EventBridge to describe the key in order to verify if the key is symmetric.
          # However usage of conidition keys like aws:SourceAccount and aws:SourceArn is not supported.
          # https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-encryption-key-policy.html#eb-encryption-event-bus-confused-deputy
          # https://docs.aws.amazon.com/eventbridge/latest/userguide/encryption-archives.html#encryption-archives-key-policy
          - Sid: "Allow EventBusArchive to use the key"
            Effect: "Allow"
            Principal:
              Service: "events.amazonaws.com"
            Action:
              - "kms:Decrypt"
              - "kms:GenerateDataKey"
              - "kms:ReEncrypt*"
            Resource: "*"
            Condition:
              ArnLike:
                "kms:EncryptionContext:aws:events:event-bus:arn": !Sub "arn:aws:events:${Region}:${DataLakeAccountId}:event-bus/*"
          # Allow EventBridge to describe the key in order to verify if the key is symmetric.
          # However condition key to prevent confused deputy is not supported.
          # https://docs.aws.amazon.com/eventbridge/latest/userguide/encryption-archives.html#encryption-archives-key-policy
          - Sid: "Allow EventBridge Bus to describe the key"
            Effect: "Allow"
            Principal:
              Service: "events.amazonaws.com"
            Action:
              - "kms:DescribeKey"
            Resource: "*"
  EventBusKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/datalake/eventbus"
      TargetKeyId: !Ref EventBusKmsKey
  CloudwatchLogsKmsKey:
```

```
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for CloudWatch logs in the DataLake account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          # Ref: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/encrypt-log-
          ↪  data-kms.html#cmk-permissions
          - Sid: "Allow Key usage for CloudWatch log groups and streams related S3
          ↪  event handling in the Data Lake account"
            Effect: "Allow"
            Principal:
              Service: !Sub "logs.${AWS::Region}.amazonaws.com"
            Action:
              - "kms:Encrypt"
              - "kms:Decrypt"
              - "kms:ReEncrypt*"
              - "kms:GenerateDataKey*"
              - "kms:Describe*"
            Resource: "*"
            Condition:
              ArnLike:
                "kms:EncryptionContext:aws:logs:arn":
                  - !Sub "arn:aws:logs:${Region}:${DataLakeAccountId}:log-group:/${App
                  ↪  lication}/*"
  CloudwatchLogsKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/datalake/logs"
      TargetKeyId: !Ref CloudwatchLogsKmsKey
  LambdaKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for all Lambda functions in the DataLake account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
      # Overwriting default policy
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          # Default Statement. TODO: MUST be restricted. Details to be clarified.
          - Sid: "Enable IAM User Permissions"
```

```
            Effect: "Allow"
            Principal:
              AWS: !Sub
              ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow datalake account admin to create grant"
            Effect: "Allow"
            Principal:
              AWS: !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdmin
              ↪  RoleName}"
            Action:
              - "kms:CreateGrant"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "lambda.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref DataLakeAccountId
              Bool:
                "kms:GrantIsForAWSResource": true
          - Sid: "Allow datalake account admin to encrypt Lambda environment variables"
            Effect: "Allow"
            Principal:
              AWS: !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdmin
              ↪  RoleName}"
            Action:
              - "kms:Encrypt"
              - "kms:ReEncrypt*"
              - "kms:DescribeKey"
            Resource: "*"
            Condition:
              StringEquals:
                "kms:ViaService": !Sub "lambda.${Region}.amazonaws.com"
                "kms:CallerAccount": !Ref DataLakeAccountId
          - Sid: "Allow datalake account admin to view Lambda environment variables"
            Effect: "Allow"
            Principal:
              AWS: !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdmin
              ↪  RoleName}"
            Action:
              - "kms:Decrypt"
            Resource: "*"
            Condition:
              ArnEquals:
                "kms:EncryptionContext:aws:lambda:FunctionArn":
                  - !Sub "arn:aws:lambda:${Region}:${DataLakeAccountId}:function:*"
  LambdaDataLakeKmsKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${Prefix}/datalake/lambda"
      TargetKeyId: !Ref LambdaKmsKey
  DynamoDbKmsKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS key for all dynamo db tables in the DataLake account"
      Enabled: true
      EnableKeyRotation: true
      KeySpec: "SYMMETRIC_DEFAULT"
      KeyUsage: "ENCRYPT_DECRYPT"
      MultiRegion: false
      PendingWindowInDays: 30
```

```
481       # Overwriting default policy
482       KeyPolicy:
483         Version: "2012-10-17"
484         Statement:
485           # Default Statement. TODO: MUST be restricted. Details to be clarified.
486           - Sid: "Enable IAM User Permissions"
487             Effect: "Allow"
488             Principal:
489               AWS: !Sub
                    ↪  "arn:aws:iam::${AWS::AccountId}:role/${KeyManagementIamRoleName}"
490             Action: "kms:*"
491             Resource: "*"
492           # Based on https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
            ↪  encryption.usagenotes.html#dynamodb-kms-authz
493           - Sid: "Allow specific roles of the datalake account to use the key for
            ↪  DynamoDB table creation"
494             Effect: "Allow"
495             Principal:
496               AWS:
497                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                    ↪  oleName}"
498             Action:
499               - "kms:Encrypt"
500               - "kms:ReEncrypt*"
501               - "kms:GenerateDataKey*"
502               - "kms:DescribeKey"
503               - "kms:CreateGrant"
504             Resource: "*"
505             Condition:
506               StringEquals:
507                 "kms:ViaService": !Sub "dynamodb.${Region}.amazonaws.com"
508           - Sid: "Allow specific IAM roles of the datalake account to use the key for
            ↪  DynamoDB table operations"
509             Effect: "Allow"
510             Principal:
511               AWS:
512                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${DataLakeAccountAdminR
                    ↪  oleName}"
513                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                    ↪  keAccountProcessTelemetryDataLambdaRoleNameSuffix}"
514                 - !Sub "arn:aws:iam::${DataLakeAccountId}:role/${Application}-${DataLa
                    ↪  keAccountImageExifExtractionLambdaRoleNameSuffix}"
515             Action:
516               - "kms:Decrypt"
517             Resource: "*"
518             Condition:
519               StringEquals:
520                 "kms:ViaService": !Sub "dynamodb.${Region}.amazonaws.com"
521     DynamoDbKmsKeyAlias:
522       Type: "AWS::KMS::Alias"
523       Properties:
524         AliasName: !Sub "alias/${Prefix}/datalake/dynamodb"
525         TargetKeyId: !Ref DynamoDbKmsKey
```

9

9 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

10_guardrails.yaml
Instantiates detective and preventive controls for all environments.

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: "A stack for all preventive and detective guardrails"

Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
    Description: "Name of the application the resources belong to"
  Stage:
    Type: "String"
    Default: "dev"
    Description: "The stage. E.g. 'dev' or 'prod'"
  Prefix:
    Type: "String"
    Default: "omniaware"
    Description: "A prefix used for resource naming. E.g. S3 Bucket prefix."
  Region:
    Type: "String"
    Default: "eu-central-1"
    Description: "The region."
  AggregatorName:
    Type: String
    Description: Name for the AWS Config Aggregator
    Default: MultiAccountConfigAggregator
  AuditAccountId:
    Type: String
    Description: AWS Account ID of the central audit account
    Default: <AccountId>
  SecurityAccountId:
    Type: String
    Description: AWS Account ID of the central audit account
    Default: <AccountId>
  IngestAccountId:
    Type: String
    Description: AWS Account ID of the central ingest account
    Default: <AccountId>
  DatalakeAccountId:
    Type: String
    Description: AWS Account ID of the central ingest account
    Default: <AccountId>
  ConsumerAccountId:
    Type: String
    Description: AWS Account ID of the central ingest account
    Default: <AccountId>
  ServiceName:
    Type: String
    Default: s3
    Description: 'The name of the AWS service to check for VPC endpoints (e.g., s3,
    ↪  dynamodb, ec2, etc.)'
    AllowedValues:
      - "com.amazonaws.${Region}.s3"
      - "com.amazonaws.${Region}.dynamodb"
      - "com.amazonaws.${Region}.ec2"
      - "com.amazonaws.${Region}.ssm"
      - "com.amazonaws.${Region}.logs"
      - "com.amazonaws.${Region}.monitoring"
      - "com.amazonaws.${Region}.kms"
      - "com.amazonaws.${Region}.secretsmanager"
```

```
      - "com.amazonaws.${Region}.sqs"
      - "com.amazonaws.${Region}.sns"
      - "com.amazonaws.${Region}.lambda"
      - "com.amazonaws.${Region}.kinesis"
      - "com.amazonaws.${Region}.apigateway"
      - "com.amazonaws.${Region}.ecs"
      - "com.amazonaws.${Region}.glue"
      - "com.amazonaws.${Region}.athena"
      - "com.amazonaws.${Region}.cloudtrail"
      - "com.amazonaws.${Region}.cloudwatch"
      - "com.amazonaws.${Region}.sagemaker"
      - "com.amazonaws.${Region}.bedrock"
      - "com.amazonaws.${Region}.transcribe"
      - "com.amazonaws.${Region}.rekognition"
      - "com.amazonaws.${Region}.polly"
      - "com.amazonaws.${Region}.codebuild"
      - "com.amazonaws.${Region}.codepipeline"
  AuthorizedTcpPorts:
    Type: String
    Default: '443'
    Description: 'Comma-separated list of TCP ports authorized to be open to 0.0.0.0/0'
  AuthorizedUdpPorts:
    Type: String
    Default: ''
    Description: 'Comma-separated list of UDP ports authorized to be open to 0.0.0.0/0'
  RestrictedProtocols:
    Type: String
    Default: '6,17'
    Description: 'Comma-separated list of protocols to restrict (e.g., 6=TCP, 17=UDP)'
  RestrictedPorts:
    Type: String
    Default: '443'
    Description: 'Comma-separated list of ports to restrict from the public internet'
  RestrictedPortScope:
    Type: String
    Default: 'public'  # Can be 'public' or 'all'
    AllowedValues:
      - 'all'
    Description: 'Define which security group ingress rules to check, "public" means
    ↪  ingress from 0.0.0.0/0, "all" means all ingress rules'
  ExcludeSecurityGroups:
    Type: String
    Default: ''
    Description: 'Comma-separated list of security group IDs to exclude from the check'
  AuditBucketName:
    Type: String
    Default: !ImportValue AuditBucketName
    Description: 'Name of the audit bucket'
  ConfigBucketName:
    Type: String
    Default: !ImportValue ConfigBucketName
    Description: 'Name of the config bucket'

Resources:
  # ========== Resources for all accounts (Audit and Member accounts) ==========

  # IAM Role for AWS Config
  ConfigServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
```

```
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: config.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWS_ConfigRole
    Policies:
      - PolicyName: ConfigDeliveryPermissions
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - s3:PutObject
              Resource: !Sub "arn:aws:s3:::${ConfigBucketName}-${AuditAccountId}/AWS
              ↪ Logs/${AWS::AccountId}/*"
              Condition:
                StringLike:
                  s3:x-amz-acl: bucket-owner-full-control
            - Effect: Allow
              Action:
                - s3:GetBucketAcl
              Resource: !Sub "arn:aws:s3:::${ConfigBucketName}-${AuditAccountId}"
            - Effect: Allow
              Action:
                - sns:Publish
              Resource: !ImportValue ConfigSNSTopicArn
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

# AWS Config Recorder
ConfigRecorder:
  Type: AWS::Config::ConfigurationRecorder
  Properties:
    Name: default
    RecordingGroup:
      AllSupported: true
      IncludeGlobalResourceTypes: true
    RoleARN: !GetAtt ConfigServiceRole.Arn

# AWS Config Delivery Channel
ConfigDeliveryChannel:
  Type: AWS::Config::DeliveryChannel
  Properties:
    ConfigSnapshotDeliveryProperties:
      DeliveryFrequency: Six_Hours
    S3BucketName: !ImportValue ConfigBucketName
    SnsTopicARN: !ImportValue ConfigSNSTopicArn

# ========== Common Config Rules for all accounts ==========
AccessKeyRotationRule:
  Type: 'AWS::Config::ConfigRule'
  Properties:
    ConfigRuleName: 'access-keys-rotated'
```

```
178       Source:
179         Owner: 'AWS'
180         SourceIdentifier: 'ACCESS_KEYS_ROTATED'
181       Tags:
182         - Key: "Stack"
183           Value: !Ref AWS::StackName
184         - Key: "Application"
185           Value: !Ref Application
186         - Key: "Stage"
187           Value: !Ref Stage
188
189   IAMPasswordPolicyRule:
190     Type: 'AWS::Config::ConfigRule'
191     Properties:
192       ConfigRuleName: 'iam-password-policy'
193       Source:
194         Owner: 'AWS'
195         SourceIdentifier: 'IAM_PASSWORD_POLICY'
196       Tags:
197         - Key: "Stack"
198           Value: !Ref AWS::StackName
199         - Key: "Application"
200           Value: !Ref Application
201         - Key: "Stage"
202           Value: !Ref Stage
203
204   IAMUserMFAEnabled:
205     Type: AWS::Config::ConfigRule
206     Properties:
207       ConfigRuleName: iam-user-mfa-enabled
208       Description: Checks whether AWS Identity and Access Management (IAM) users have
          ↪ multi-factor authentication (MFA) enabled.
209       Source:
210         Owner: AWS
211         SourceIdentifier: IAM_USER_MFA_ENABLED
212       Tags:
213         - Key: "Stack"
214           Value: !Ref AWS::StackName
215         - Key: "Application"
216           Value: !Ref Application
217         - Key: "Stage"
218           Value: !Ref Stage
219
220   IAMUserNoPoliciesCheck:
221     Type: AWS::Config::ConfigRule
222     Properties:
223       ConfigRuleName: iam-user-no-policies-check
224       Description: Checks that none of your IAM users have policies attached. IAM users
          ↪ must inherit permissions from IAM groups or roles.
225       Source:
226         Owner: AWS
227         SourceIdentifier: IAM_USER_NO_POLICIES_CHECK
228       Tags:
229         - Key: "Stack"
230           Value: !Ref AWS::StackName
231         - Key: "Application"
232           Value: !Ref Application
233         - Key: "Stage"
234           Value: !Ref Stage
235
236   S3BucketPublicReadProhibitedRule:
```

```
    Type: 'AWS::Config::ConfigRule'
    Properties:
      ConfigRuleName: 's3-bucket-public-read-prohibited'
      Source:
        Owner: 'AWS'
        SourceIdentifier: 'S3_BUCKET_PUBLIC_READ_PROHIBITED'
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  S3BucketPublicWriteProhibitedRule:
    Type: 'AWS::Config::ConfigRule'
    Properties:
      ConfigRuleName: 's3-bucket-public-write-prohibited'
      Source:
        Owner: 'AWS'
        SourceIdentifier: 'S3_BUCKET_PUBLIC_WRITE_PROHIBITED'
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  S3BucketServerSideEncryptionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: s3-bucket-server-side-encryption-enabled
      Description: Checks that your Amazon S3 bucket either has Amazon S3 default
      ↪  encryption enabled or that the S3 bucket policy explicitly denies put-object
      ↪  requests without server-side encryption.
      Source:
        Owner: AWS
        SourceIdentifier: S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  S3AccessPointInVpcOnly:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: s3-access-point-in-vpc-only
      Description: Checks if Amazon S3 access points are configured to accept requests
      ↪  from a Virtual Private Cloud (VPC) only.
      Source:
        Owner: AWS
        SourceIdentifier: S3_ACCESS_POINT_IN_VPC_ONLY
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
```

```
295         - Key: "Stage"
296           Value: !Ref Stage
297
298   S3AccessPointPublicAccessBlocks:
299     Type: AWS::Config::ConfigRule
300     Properties:
301       ConfigRuleName: s3-access-point-public-access-blocks
302       Description: Evaluates if an S3 access point is configured with a restrictive
          ↪ PublicAccessBlock configuration.
303       Source:
304         Owner: AWS
305         SourceIdentifier: S3_ACCESS_POINT_PUBLIC_ACCESS_BLOCKS
306       Tags:
307         - Key: "Stack"
308           Value: !Ref AWS::StackName
309         - Key: "Application"
310           Value: !Ref Application
311         - Key: "Stage"
312           Value: !Ref Stage
313
314   S3AccountLevelPublicAccessBlocks:
315     Type: AWS::Config::ConfigRule
316     Properties:
317       ConfigRuleName: s3-account-level-public-access-blocks
318       Description: Checks if the required public access block settings are configured
          ↪ from account level.
319       Source:
320         Owner: AWS
321         SourceIdentifier: S3_ACCOUNT_LEVEL_PUBLIC_ACCESS_BLOCKS
322       Tags:
323         - Key: "Stack"
324           Value: !Ref AWS::StackName
325         - Key: "Application"
326           Value: !Ref Application
327         - Key: "Stage"
328           Value: !Ref Stage
329
330   S3BucketAclProhibited:
331     Type: AWS::Config::ConfigRule
332     Properties:
333       ConfigRuleName: s3-bucket-acl-prohibited
334       Description: Checks if Amazon S3 Buckets allow user permissions through access
          ↪ control lists (ACLs).
335       Source:
336         Owner: AWS
337         SourceIdentifier: S3_BUCKET_ACL_PROHIBITED
338       Tags:
339         - Key: "Stack"
340           Value: !Ref AWS::StackName
341         - Key: "Application"
342           Value: !Ref Application
343         - Key: "Stage"
344           Value: !Ref Stage
345
346   S3BucketEncryptionRule:
347     Type: 'AWS::Config::ConfigRule'
348     Properties:
349       ConfigRuleName: 's3-bucket-server-side-encryption-enabled'
350       Scope:
351         ComplianceResourceTypes:
352           - 'AWS::S3::Bucket'
```

```
    Source:
      Owner: 'AWS'
      SourceIdentifier: 'S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED'
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

S3BucketLevelPublicAccessProhibited:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: s3-bucket-level-public-access-prohibited
    Description: Checks if Amazon S3 buckets have public access blocks configured at
    ↪  the bucket level.
    Source:
      Owner: AWS
      SourceIdentifier: S3_BUCKET_LEVEL_PUBLIC_ACCESS_PROHIBITED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

S3BucketDefaultLockEnabled:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: s3-bucket-default-lock-enabled
    Description: Checks if Amazon S3 buckets have lock configurations enabled by
    ↪  default.
    Source:
      Owner: AWS
      SourceIdentifier: S3_BUCKET_DEFAULT_LOCK_ENABLED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

S3BucketMfaDeleteEnabled:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: s3-bucket-mfa-delete-enabled
    Description: Checks if Amazon S3 buckets have MFA delete enabled.
    Source:
      Owner: AWS
      SourceIdentifier: S3_BUCKET_MFA_DELETE_ENABLED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

```

```
412   S3BucketPolicyNotMorePermissive:
413     Type: AWS::Config::ConfigRule
414     Properties:
415       ConfigRuleName: s3-bucket-policy-not-more-permissive
416       Description: Checks if Amazon S3 bucket policies do not allow more permissive
          ↪  actions than the baseline.
417       Source:
418         Owner: AWS
419         SourceIdentifier: S3_BUCKET_POLICY_NOT_MORE_PERMISSIVE
420       Tags:
421         - Key: "Stack"
422           Value: !Ref AWS::StackName
423         - Key: "Application"
424           Value: !Ref Application
425         - Key: "Stage"
426           Value: !Ref Stage
427
428   S3BucketSslRequestsOnly:
429     Type: AWS::Config::ConfigRule
430     Properties:
431       ConfigRuleName: s3-bucket-ssl-requests-only
432       Description: Checks if Amazon S3 buckets have policies that require requests to
          ↪  use Secure Socket Layer (SSL).
433       Source:
434         Owner: AWS
435         SourceIdentifier: S3_BUCKET_SSL_REQUESTS_ONLY
436       Tags:
437         - Key: "Stack"
438           Value: !Ref AWS::StackName
439         - Key: "Application"
440           Value: !Ref Application
441         - Key: "Stage"
442           Value: !Ref Stage
443
444   S3BucketVersioningEnabled:
445     Type: AWS::Config::ConfigRule
446     Properties:
447       ConfigRuleName: s3-bucket-versioning-enabled
448       Description: Checks if versioning is enabled for your S3 buckets.
449       Source:
450         Owner: AWS
451         SourceIdentifier: S3_BUCKET_VERSIONING_ENABLED
452       Tags:
453         - Key: "Stack"
454           Value: !Ref AWS::StackName
455         - Key: "Application"
456           Value: !Ref Application
457         - Key: "Stage"
458           Value: !Ref Stage
459
460   S3DefaultEncryptionKMS:
461     Type: AWS::Config::ConfigRule
462     Properties:
463       ConfigRuleName: s3-default-encryption-kms
464       Description: Checks if the Amazon S3 buckets are encrypted with AWS Key
          ↪  Management Service (AWS KMS).
465       Source:
466         Owner: AWS
467         SourceIdentifier: S3_DEFAULT_ENCRYPTION_KMS
468       Tags:
469         - Key: "Stack"
```

```
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  CloudTrailEncryptionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: cloud-trail-encryption-enabled
      Description: Checks if AWS CloudTrail is configured to use server-side encryption
      ↪ (SSE) with AWS KMS keys (SSE-KMS) encryption.
      Source:
        Owner: AWS
        SourceIdentifier: CLOUD_TRAIL_ENCRYPTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  CloudTrailCloudWatchLogsEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: cloud-trail-cloud-watch-logs-enabled
      Description: Checks if AWS CloudTrail trails are configured to send logs to
      ↪ CloudWatch Logs.
      Source:
        Owner: AWS
        SourceIdentifier: CLOUD_TRAIL_CLOUD_WATCH_LOGS_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  MultiRegionCloudTrailEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: multi-region-cloud-trail-enabled
      Description: Checks if there is at least one multi-region AWS CloudTrail.
      Source:
        Owner: AWS
        SourceIdentifier: MULTI_REGION_CLOUD_TRAIL_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  CloudTrailEnabledRule:
    Type: 'AWS::Config::ConfigRule'
    Properties:
      ConfigRuleName: 'cloudtrail-enabled'
      Source:
```

```
529         Owner: 'AWS'
530         SourceIdentifier: 'CLOUD_TRAIL_ENABLED'
531       Tags:
532         - Key: "Stack"
533           Value: !Ref AWS::StackName
534         - Key: "Application"
535           Value: !Ref Application
536         - Key: "Stage"
537           Value: !Ref Stage
538
539   EBSVolumeEncryptionRule:
540     Type: 'AWS::Config::ConfigRule'
541     Properties:
542       ConfigRuleName: 'ebs-encrypted-volumes'
543       Scope:
544         ComplianceResourceTypes:
545           - 'AWS::EC2::Volume'
546       Source:
547         Owner: 'AWS'
548         SourceIdentifier: 'ENCRYPTED_VOLUMES'
549       Tags:
550         - Key: "Stack"
551           Value: !Ref AWS::StackName
552         - Key: "Application"
553           Value: !Ref Application
554         - Key: "Stage"
555           Value: !Ref Stage
556
557   EC2SecurityGroupRule:
558     Type: 'AWS::Config::ConfigRule'
559     Properties:
560       ConfigRuleName: 'restricted-common-ports'
561       Scope:
562         ComplianceResourceTypes:
563           - 'AWS::EC2::SecurityGroup'
564       Source:
565         Owner: 'AWS'
566         SourceIdentifier: 'INCOMING_SSH_DISABLED'
567       Tags:
568         - Key: "Stack"
569           Value: !Ref AWS::StackName
570         - Key: "Application"
571           Value: !Ref Application
572         - Key: "Stage"
573           Value: !Ref Stage
574
575   VPCFlowLogsEnabledRule:
576     Type: 'AWS::Config::ConfigRule'
577     Properties:
578       ConfigRuleName: 'vpc-flow-logs-enabled'
579       Source:
580         Owner: 'AWS'
581         SourceIdentifier: 'VPC_FLOW_LOGS_ENABLED'
582       Tags:
583         - Key: "Stack"
584           Value: !Ref AWS::StackName
585         - Key: "Application"
586           Value: !Ref Application
587         - Key: "Stage"
588           Value: !Ref Stage
589
```

```
  EC2EbsEncryptionByDefault:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: ec2-ebs-encryption-by-default
      Description: Checks if Amazon Elastic Block Store (EBS) encryption is enabled by
      ↪  default.
      Source:
        Owner: AWS
        SourceIdentifier: EC2_EBS_ENCRYPTION_BY_DEFAULT
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  EC2IMDSv2Check:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: ec2-imdsv2-check
      Description: Checks if Amazon EC2 instances are configured with Instance Metadata
      ↪  Service Version 2 (IMDSv2).
      Source:
        Owner: AWS
        SourceIdentifier: EC2_IMDSV2_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  EC2InstanceDetailedMonitoringEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: ec2-instance-detailed-monitoring-enabled
      Description: Checks if detailed monitoring is enabled for EC2 instances.
      Source:
        Owner: AWS
        SourceIdentifier: EC2_INSTANCE_DETAILED_MONITORING_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  EC2InstanceNoPublicIP:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: ec2-instance-no-public-ip
      Description: Checks if EC2 instances have public IPs assigned.
      Source:
        Owner: AWS
        SourceIdentifier: EC2_INSTANCE_NO_PUBLIC_IP
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
```

```
649         - Key: "Application"
650           Value: !Ref Application
651         - Key: "Stage"
652           Value: !Ref Stage
653
654   EC2NoAmazonKeyPair:
655     Type: AWS::Config::ConfigRule
656     Properties:
657       ConfigRuleName: ec2-no-amazon-key-pair
658       Description: Checks if running Amazon EC2 instances are launched using Amazon EC2
        ↪ key pairs.
659       Source:
660         Owner: AWS
661         SourceIdentifier: EC2_NO_AMAZON_KEY_PAIR
662       Tags:
663         - Key: "Stack"
664           Value: !Ref AWS::StackName
665         - Key: "Application"
666           Value: !Ref Application
667         - Key: "Stage"
668           Value: !Ref Stage
669
670   EC2StoppedInstance:
671     Type: AWS::Config::ConfigRule
672     Properties:
673       ConfigRuleName: ec2-stopped-instance
674       Description: Checks if there are instances stopped for more than the allowed
        ↪ number of days.
675       Source:
676         Owner: AWS
677         SourceIdentifier: EC2_STOPPED_INSTANCE
678       Tags:
679         - Key: "Stack"
680           Value: !Ref AWS::StackName
681         - Key: "Application"
682           Value: !Ref Application
683         - Key: "Stage"
684           Value: !Ref Stage
685
686   EC2VolumeInuseCheck:
687     Type: AWS::Config::ConfigRule
688     Properties:
689       ConfigRuleName: ec2-volume-inuse-check
690       Description: Checks if EBS volumes are attached to EC2 instances.
691       Source:
692         Owner: AWS
693         SourceIdentifier: EC2_VOLUME_INUSE_CHECK
694       Tags:
695         - Key: "Stack"
696           Value: !Ref AWS::StackName
697         - Key: "Application"
698           Value: !Ref Application
699         - Key: "Stage"
700           Value: !Ref Stage
701
702   InstancesInVPC:
703     Type: AWS::Config::ConfigRule
704     Properties:
705       ConfigRuleName: instances-in-vpc
706       Description: Checks if your EC2 instances belong to a virtual private cloud
        ↪ (VPC).
```

```
    Source:
      Owner: AWS
      SourceIdentifier: INSTANCES_IN_VPC
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

# Network Rules
NaclNoUnrestrictedSshRdp:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: nacl-no-unrestricted-ssh-rdp
    Description: Checks if default ports for SSH/RDP ingress traffic in Network ACLs
    ↪ are unrestricted.
    Source:
      Owner: AWS
      SourceIdentifier: NACL_NO_UNRESTRICTED_SSH_RDP
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

NoUnrestrictedRouteToIGW:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: no-unrestricted-route-to-igw
    Description: Checks if there are public routes in the route table to an Internet
    ↪ Gateway (IGW).
    Source:
      Owner: AWS
      SourceIdentifier: NO_UNRESTRICTED_ROUTE_TO_IGW
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

SubnetAutoAssignPublicIpDisabled:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: subnet-auto-assign-public-ip-disabled
    Description: Checks if Amazon Virtual Private Cloud (VPC) subnets are assigned a
    ↪ public IP address.
    Source:
      Owner: AWS
      SourceIdentifier: SUBNET_AUTO_ASSIGN_PUBLIC_IP_DISABLED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
```

```
765           Value: !Ref Stage
766
767   # ELB Rules
768   ELBv2AcmCertificateRequired:
769     Type: AWS::Config::ConfigRule
770     Properties:
771       ConfigRuleName: elbv2-acm-certificate-required
772       Description: Checks if Application Load Balancers and Network Load Balancers have
          ↪ listeners configured to use certificates from AWS Certificate Manager (ACM).
773       Source:
774         Owner: AWS
775         SourceIdentifier: ELBV2_ACM_CERTIFICATE_REQUIRED
776       Tags:
777         - Key: "Stack"
778           Value: !Ref AWS::StackName
779         - Key: "Application"
780           Value: !Ref Application
781         - Key: "Stage"
782           Value: !Ref Stage
783
784   ELBDeletionProtectionEnabled:
785     Type: AWS::Config::ConfigRule
786     Properties:
787       ConfigRuleName: elb-deletion-protection-enabled
788       Description: Checks if Elastic Load Balancing has deletion protection enabled.
789       Source:
790         Owner: AWS
791         SourceIdentifier: ELB_DELETION_PROTECTION_ENABLED
792       Tags:
793         - Key: "Stack"
794           Value: !Ref AWS::StackName
795         - Key: "Application"
796           Value: !Ref Application
797         - Key: "Stage"
798           Value: !Ref Stage
799
800   ELBv2MultipleAZ:
801     Type: AWS::Config::ConfigRule
802     Properties:
803       ConfigRuleName: elbv2-multiple-az
804       Description: Checks if an Elastic Load Balancer V2 (Application, Network or
          ↪ Gateway) has registered instances from multiple Availability Zones.
805       Source:
806         Owner: AWS
807         SourceIdentifier: ELBV2_MULTIPLE_AZ
808       Tags:
809         - Key: "Stack"
810           Value: !Ref AWS::StackName
811         - Key: "Application"
812           Value: !Ref Application
813         - Key: "Stage"
814           Value: !Ref Stage
815
816   # ECS Rules
817   ECSFargateLatestPlatformVersion:
818     Type: AWS::Config::ConfigRule
819     Properties:
820       ConfigRuleName: ecs-fargate-latest-platform-version
821       Description: Checks if Amazon ECS Fargate tasks are using the latest Fargate
          ↪ platform version.
822       Source:
```

```
        Owner: AWS
        SourceIdentifier: ECS_FARGATE_LATEST_PLATFORM_VERSION
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # API Gateway Rules
  APIGWSslEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: api-gw-ssl-enabled
      Description: Checks if Amazon API Gateway stages have SSL certificates enabled.
      Source:
        Owner: AWS
        SourceIdentifier: API_GW_SSL_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  APIGWAssociatedWithWAF:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: api-gw-associated-with-waf
      Description: Checks if an API Gateway stage is using an AWS WAF Web ACL.
      Source:
        Owner: AWS
        SourceIdentifier: API_GW_ASSOCIATED_WITH_WAF
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  APIGWv2AccessLogsEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: api-gwv2-access-logs-enabled
      Description: Checks if Amazon API Gateway V2 stages have access logging enabled.
      Source:
        Owner: AWS
        SourceIdentifier: API_GWV2_ACCESS_LOGS_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Athena Rules
  AthenaWorkgroupEncryptedAtRest:
```

```
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: athena-workgroup-encrypted-at-rest
      Description: Checks if Amazon Athena workgroups are encrypted at rest.
      Source:
        Owner: AWS
        SourceIdentifier: ATHENA_WORKGROUP_ENCRYPTED_AT_REST
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # KMS Rules
  KMSKeyPolicyNoPublicAccess:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: kms-key-policy-no-public-access
      Description: Checks if AWS KMS keys are not publicly accessible.
      Source:
        Owner: AWS
        SourceIdentifier: KMS_KEY_POLICY_NO_PUBLIC_ACCESS
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # GuardDuty Rules
  GuardDutyS3ProtectionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: guardduty-s3-protection-enabled
      Description: Checks if Amazon GuardDuty has S3 Protection enabled.
      Source:
        Owner: AWS
        SourceIdentifier: GUARDDUTY_S3_PROTECTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  GuardDutyEC2ProtectionRuntimeEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: guardduty-ec2-protection-runtime-enabled
      Description: Checks if Amazon GuardDuty has EC2 Protection with runtime
      ↪  monitoring enabled.
      Source:
        Owner: AWS
        SourceIdentifier: GUARDDUTY_EC2_PROTECTION_RUNTIME_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
```

```
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  GuardDutyECSProtectionRuntimeEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: guardduty-ecs-protection-runtime-enabled
      Description: Checks if Amazon GuardDuty has ECS Protection with runtime
      ↪ monitoring enabled.
      Source:
        Owner: AWS
        SourceIdentifier: GUARDDUTY_ECS_PROTECTION_RUNTIME_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  GuardDutyLambdaProtectionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: guardduty-lambda-protection-enabled
      Description: Checks if Amazon GuardDuty has Lambda Protection enabled.
      Source:
        Owner: AWS
        SourceIdentifier: GUARDDUTY_LAMBDA_PROTECTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  GuardDutyMalwareProtectionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: guardduty-malware-protection-enabled
      Description: Checks if Amazon GuardDuty has Malware Protection enabled.
      Source:
        Owner: AWS
        SourceIdentifier: GUARDDUTY_MALWARE_PROTECTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Glue Rules
  GlueMLTransformEncryptedAtRest:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: glue-ml-transform-encrypted-at-rest
      Description: Checks if AWS Glue machine learning transforms are encrypted at
      ↪ rest.
```

```
      Source:
        Owner: AWS
        SourceIdentifier: GLUE_ML_TRANSFORM_ENCRYPTED_AT_REST
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Backup Rules
  BackupRecoveryPointEncrypted:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: backup-recovery-point-encrypted
      Description: Checks if a recovery point is encrypted.
      Source:
        Owner: AWS
        SourceIdentifier: BACKUP_RECOVERY_POINT_ENCRYPTED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # CodeDeploy Rules
  CodeDeployLambdaAllAtOnceTrafficShiftDisabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: codedeploy-lambda-allatonce-traffic-shift-disabled
      Description: Checks if AWS CodeDeploy Lambda deployment group traffic shifting
      ↪  configuration is set to all-at-once.
      Source:
        Owner: AWS
        SourceIdentifier: CODEDEPLOY_LAMBDA_ALLATONCE_TRAFFIC_SHIFT_DISABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # DynamoDB Rules
  DynamoDBPITREnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: dynamodb-pitr-enabled
      Description: Checks if point-in-time recovery (PITR) is enabled for Amazon
      ↪  DynamoDB tables.
      Source:
        Owner: AWS
        SourceIdentifier: DYNAMODB_PITR_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
```

```
        - Key: "Stage"
          Value: !Ref Stage

  DynamoDBTableDeletionProtectionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: dynamodb-table-deletion-protection-enabled
      Description: Checks if Amazon DynamoDB tables have deletion protection enabled.
      Source:
        Owner: AWS
        SourceIdentifier: DYNAMODB_TABLE_DELETION_PROTECTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  DynamoDBTableEncryptedKMS:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: dynamodb-table-encrypted-kms
      Description: Checks if Amazon DynamoDB tables are encrypted with AWS Key
      ↪  Management Service (AWS KMS).
      Source:
        Owner: AWS
        SourceIdentifier: DYNAMODB_TABLE_ENCRYPTED_KMS
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  DynamoDBTableEncryptionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: dynamodb-table-encryption-enabled
      Description: Checks if Amazon DynamoDB tables are encrypted and checks their
      ↪  status.
      Source:
        Owner: AWS
        SourceIdentifier: DYNAMODB_TABLE_ENCRYPTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Kinesis Rules
  KinesisFirehoseDeliveryStreamEncrypted:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: kinesis-firehose-delivery-stream-encrypted
      Description: Checks if Amazon Kinesis Firehose delivery streams are encrypted at
      ↪  rest with AWS Key Management Service (AWS KMS).
      Source:
```

```
        Owner: AWS
        SourceIdentifier: KINESIS_FIREHOSE_DELIVERY_STREAM_ENCRYPTED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Lambda Rules
  LambdaConcurrencyCheck:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: lambda-concurrency-check
      Description: Checks if the Lambda function is configured with function-level
      ↪  concurrent execution limit.
      Source:
        Owner: AWS
        SourceIdentifier: LAMBDA_CONCURRENCY_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  LambdaInsideVPC:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: lambda-inside-vpc
      Description: Checks if AWS Lambda functions are in an Amazon Virtual Private
      ↪  Cloud.
      Source:
        Owner: AWS
        SourceIdentifier: LAMBDA_INSIDE_VPC
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  LambdaVPCMultiAZCheck:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: lambda-vpc-multi-az-check
      Description: Checks if Lambda functions are configured to use more than one
      ↪  Availability Zone.
      Source:
        Owner: AWS
        SourceIdentifier: LAMBDA_VPC_MULTI_AZ_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage
```

```
  LambdaFunctionPublicAccessProhibited:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: lambda-function-public-access-prohibited
      Description: Checks if the Lambda function policy prohibits public access.
      Source:
        Owner: AWS
        SourceIdentifier: LAMBDA_FUNCTION_PUBLIC_ACCESS_PROHIBITED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Network Firewall Rules
  NetFWDeletionProtectionEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: netfw-deletion-protection-enabled
      Description: Checks if an AWS Network Firewall policy has deletion protection
      ↪  enabled.
      Source:
        Owner: AWS
        SourceIdentifier: NETFW_DELETION_PROTECTION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  NetFWStatelessRuleGroupNotEmpty:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: netfw-stateless-rule-group-not-empty
      Description: Checks if an AWS Network Firewall stateless rule group contains
      ↪  rules.
      Source:
        Owner: AWS
        SourceIdentifier: NETFW_STATELESS_RULE_GROUP_NOT_EMPTY
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # OpenSearch Rules
  OpenSearchEncryptedAtRest:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-encrypted-at-rest
      Description: Checks if Amazon OpenSearch Service domains have encryption at rest
      ↪  configuration enabled.
      Source:
        Owner: AWS
```

```
        SourceIdentifier: OPENSEARCH_ENCRYPTED_AT_REST
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchInVPCOnly:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-in-vpc-only
      Description: Checks if Amazon OpenSearch Service domains are within an Amazon
      ↪ Virtual Private Cloud (VPC).
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_IN_VPC_ONLY
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchNodeToNodeEncryptionCheck:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-node-to-node-encryption-check
      Description: Checks if Amazon OpenSearch Service nodes are configured with
      ↪ node-to-node encryption.
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_NODE_TO_NODE_ENCRYPTION_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchHTTPSRequired:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-https-required
      Description: Checks if Amazon OpenSearch domains have HTTPS required for all
      ↪ traffic.
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_HTTPS_REQUIRED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchAccessControlEnabled:
```

```
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-access-control-enabled
      Description: Checks if Amazon OpenSearch domains have fine-grained access control
      ↪ enabled.
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_ACCESS_CONTROL_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchUpdateCheck:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-update-check
      Description: Checks if Amazon OpenSearch Service domains are on the latest
      ↪ service software version.
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_UPDATE_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  OpenSearchAuditLoggingEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: opensearch-audit-logging-enabled
      Description: Checks if Amazon OpenSearch Service domains have audit logging
      ↪ enabled.
      Source:
        Owner: AWS
        SourceIdentifier: OPENSEARCH_AUDIT_LOGGING_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerDomainInVPC:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-domain-in-vpc
      Description: Checks if an Amazon SageMaker domain is configured with VPC Only
      ↪ mode.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_DOMAIN_IN_VPC
      Tags:
        - Key: "Stack"
```

```
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerModelInVPC:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-model-in-vpc
      Description: Checks if an Amazon SageMaker model is configured for a VPC.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_MODEL_IN_VPC
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerEndpointConfigurationKMSKeyConfigured:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-endpoint-configuration-kms-key-configured
      Description: Checks if an AWS KMS key was configured for an Amazon SageMaker
      ↪  endpoint configuration.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_ENDPOINT_CONFIGURATION_KMS_KEY_CONFIGURED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerModelIsolationEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-model-isolation-enabled
      Description: Checks if a SageMaker model is configured with network isolation.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_MODEL_ISOLATION_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerNotebookInstanceInsideVPC:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-notebook-instance-inside-vpc
      Description: Checks if an Amazon SageMaker Notebook instance is launched within a
      ↪  VPC.
```

```
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_NOTEBOOK_INSTANCE_INSIDE_VPC
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerNotebookInstanceKMSKeyConfigured:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-notebook-instance-kms-key-configured
      Description: Checks if an AWS KMS key is configured for an Amazon SageMaker
      ↪ notebook instance.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_NOTEBOOK_INSTANCE_KMS_KEY_CONFIGURED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  SagemakerNotebookNoDirectInternetAccess:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: sagemaker-notebook-no-direct-internet-access
      Description: Checks if direct internet access is disabled for an Amazon SageMaker
      ↪ notebook instance.
      Source:
        Owner: AWS
        SourceIdentifier: SAGEMAKER_NOTEBOOK_NO_DIRECT_INTERNET_ACCESS
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Secrets Manager Rules
  SecretsManagerRotationEnabledCheck:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: secretsmanager-rotation-enabled-check
      Description: Checks if AWS Secrets Manager secrets have rotation enabled.
      Source:
        Owner: AWS
        SourceIdentifier: SECRETSMANAGER_ROTATION_ENABLED_CHECK
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage
```

```
1469
1470    SecretsManagerScheduledRotationSuccessCheck:
1471      Type: AWS::Config::ConfigRule
1472      Properties:
1473        ConfigRuleName: secretsmanager-scheduled-rotation-success-check
1474        Description: Checks if AWS Secrets Manager secrets rotated successfully according
          ↪  to the rotation schedule.
1475        Source:
1476          Owner: AWS
1477          SourceIdentifier: SECRETSMANAGER_SCHEDULED_ROTATION_SUCCESS_CHECK
1478        Tags:
1479          - Key: "Stack"
1480            Value: !Ref AWS::StackName
1481          - Key: "Application"
1482            Value: !Ref Application
1483          - Key: "Stage"
1484            Value: !Ref Stage
1485
1486    # Security Hub Rules
1487    SecurityHubEnabled:
1488      Type: AWS::Config::ConfigRule
1489      Properties:
1490        ConfigRuleName: securityhub-enabled
1491        Description: Checks if AWS Security Hub is enabled for an AWS account.
1492        Source:
1493          Owner: AWS
1494          SourceIdentifier: SECURITYHUB_ENABLED
1495        Tags:
1496          - Key: "Stack"
1497            Value: !Ref AWS::StackName
1498          - Key: "Application"
1499            Value: !Ref Application
1500          - Key: "Stage"
1501            Value: !Ref Stage
1502
1503    # SQS Rules
1504    SQSQueueNoPublicAccess:
1505      Type: AWS::Config::ConfigRule
1506      Properties:
1507        ConfigRuleName: sqs-queue-no-public-access
1508        Description: Checks if Amazon Simple Queue Service (Amazon SQS) queues deny
          ↪  public access.
1509        Source:
1510          Owner: AWS
1511          SourceIdentifier: SQS_QUEUE_NO_PUBLIC_ACCESS
1512        Tags:
1513          - Key: "Stack"
1514            Value: !Ref AWS::StackName
1515          - Key: "Application"
1516            Value: !Ref Application
1517          - Key: "Stage"
1518            Value: !Ref Stage
1519
1520    # VPC Rules
1521    ServiceVpcEndpointEnabledRule:
1522      Type: AWS::Config::ConfigRule
1523      Properties:
1524        ConfigRuleName: service-vpc-endpoint-enabled
1525        Description: !Sub 'Checks whether Service ${ServiceName} VPC endpoint is enabled
          ↪  in all VPCs or specified VPCs'
1526        Source:
```

```
      Owner: AWS
      SourceIdentifier: SERVICE_VPC_ENDPOINT_ENABLED
    InputParameters:
      serviceName: !Ref ServiceName
    Scope:
      ComplianceResourceTypes:
        - AWS::EC2::VPC
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

VPCDefaultSecurityGroupClosed:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: vpc-default-security-group-closed
    Description: Checks if the default security group of any Amazon Virtual Private
    ↪  Cloud (VPC) does not allow inbound or outbound traffic.
    Source:
      Owner: AWS
      SourceIdentifier: VPC_DEFAULT_SECURITY_GROUP_CLOSED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

VPCFlowLogsEnabled:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: vpc-flow-logs-enabled
    Description: Checks if Amazon Virtual Private Cloud flow logs are found and
    ↪  enabled for Amazon VPC.
    Source:
      Owner: AWS
      SourceIdentifier: VPC_FLOW_LOGS_ENABLED
    Tags:
      - Key: "Stack"
        Value: !Ref AWS::StackName
      - Key: "Application"
        Value: !Ref Application
      - Key: "Stage"
        Value: !Ref Stage

VpcSgOpenOnlyToAuthorizedPortsRule:
  Type: AWS::Config::ConfigRule
  Properties:
    ConfigRuleName: vpc-sg-open-only-to-authorized-ports
    Description: 'Checks whether security groups allow unrestricted incoming traffic
    ↪  only for authorized ports'
    Scope:
      ComplianceResourceTypes:
        - 'AWS::EC2::SecurityGroup'
    Source:
      Owner: AWS
      SourceIdentifier: VPC_SG_OPEN_ONLY_TO_AUTHORIZED_PORTS
```

```
      InputParameters:
        authorizedTcpPorts: !Ref AuthorizedTcpPorts
        authorizedUdpPorts: !Ref AuthorizedUdpPorts

  VpcSgPortRestrictionCheckRule:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: vpc-sg-port-restriction-check
      Description: 'Checks if security groups restrict traffic to specified protocols
      ↪ and ports'
      Scope:
        ComplianceResourceTypes:
          - 'AWS::EC2::SecurityGroup'
      Source:
        Owner: AWS
        SourceIdentifier: VPC_SG_PORT_RESTRICTION_CHECK
      InputParameters:
        restrictedProtocols: !Ref RestrictedProtocols
        restrictedPorts: !Ref RestrictedPorts
        restrictedPortScope: !Ref RestrictedPortScope
        excludeSecurityGroups: !Ref ExcludeSecurityGroups

  WAFV2LoggingEnabled:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: wafv2-logging-enabled
      Description: Checks if logging is enabled on AWS WAFv2 regional and global web
      ↪ access control lists (web ACLs).
      Source:
        Owner: AWS
        SourceIdentifier: WAFV2_LOGGING_ENABLED
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  WAFV2RuleGroupNotEmpty:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: wafv2-rulegroup-not-empty
      Description: Checks if AWS WAFv2 rule groups contain rules.
      Source:
        Owner: AWS
        SourceIdentifier: WAFV2_RULEGROUP_NOT_EMPTY
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  WAFGlobalWebACLNotEmpty:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: waf-global-webacl-not-empty
      Description: Checks if a global AWS WAF Web ACL contains any rules or rule
      ↪ groups.
```

```
      Source:
        Owner: AWS
        SourceIdentifier: WAF_GLOBAL_WEBACL_NOT_EMPTY
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Detective Guardrails
  CloudTrailRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: 'CloudTrailRole'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: 'sts:AssumeRole'
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  CloudTrailPolicy:
    Type: 'AWS::IAM::ManagedPolicy'
    Properties:
      ManagedPolicyName: 'CloudTrailPolicy'
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - 's3:PutObject'
              - 's3:GetBucketLocation'
              - 's3:ListBucket'
            Resource:
              - !Sub 'arn:aws:s3:::${AuditBucketName}'
              - !Sub 'arn:aws:s3:::${AuditBucketName}/*'
          - Effect: Allow
            Action:
              - 'kms:GenerateDataKey'
              - 'kms:Decrypt'
            Resource: !ImportValue AuditS3BucketKMSKey
          - Effect: Allow
            Action: sts:AssumeRole
            Resource: !ImportValue CentralAuditRole

  AttachCloudTrailPolicyToRole:
    Type: 'AWS::IAM::RolePolicyAttachment'
    Properties:
      RoleName: !Ref CloudTrailRole
      PolicyArn: !Ref CloudTrailPolicy
```

```
1704
1705  MemberAccountCloudTrail:
1706    Type: AWS::CloudTrail::Trail
1707    Properties:
1708      TrailName: OmniawareAuditTrail
1709      IsLogging: true
1710      # Send logs to the central S3 bucket in the audit account
1711      S3BucketName: AuditBucketName
1712      # Reference the audit account ID where the bucket resides
1713      S3KeyPrefix: 'awslogs/${AWS::AccountId}'
1714      # Enable management event logging
1715      EnableLogFileValidation: true
1716      IncludeGlobalServiceEvents: true
1717      IsMultiRegionTrail: true
1718      # Event selectors for management events
1719      EventSelectors:
1720        - ReadWriteType: All
1721          IncludeManagementEvents: true
1722      # Advanced event selectors for network activity events
1723      AdvancedEventSelectors:
1724        - Name: Log network activity events for specified services
1725          FieldSelectors:
1726            - Field: eventCategory
1727              Equals:
1728                - NetworkActivity
1729            - Field: eventSource
1730              Equals:
1731                - cloudtrail.amazonaws.com
1732                - kms.amazonaws.com
1733                - secretsmanager.amazonaws.com
1734      CloudWatchLogsRoleArn: !GetAtt CloudTrailRole.Arn
1735      Tags:
1736        - Key: "Stack"
1737          Value: !Ref AWS::StackName
1738        - Key: "Application"
1739          Value: !Ref Application
1740        - Key: "Stage"
1741          Value: !Ref Stage
1742
1743  MemberAccountGuardDuty:
1744    Type: AWS::GuardDuty::Detector
1745    Properties:
1746      Enable: true
1747      FindingPublishingFrequency: FIFTEEN_MINUTES
1748      DataSources:
1749        CloudTrail:
1750          Enable: true
1751
1752  # IAM Role for GuardDuty to write to the central S3 bucket
1753  GuardDutyS3WriterRole:
1754    Type: AWS::IAM::Role
1755    Properties:
1756      RoleName: GuardDutyS3WriterRole
1757      AssumeRolePolicyDocument:
1758        Version: '2012-10-17'
1759        Statement:
1760          - Effect: Allow
1761            Principal:
1762              Service: guardduty.amazonaws.com
1763            Action: sts:AssumeRole
1764      ManagedPolicyArns:
```

```
        - !Ref GuardDutyS3BucketAccessPolicy
      Tags:
        - Key: "Stack"
          Value: !Ref AWS::StackName
        - Key: "Application"
          Value: !Ref Application
        - Key: "Stage"
          Value: !Ref Stage

  # Managed policy for GuardDuty S3 bucket access
  GuardDutyS3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Description: Policy for GuardDuty to access the central audit S3 bucket
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action:
              - s3:GetBucketLocation
              - s3:PutObject
              - s3:ListBucket
            Resource:
              - !Sub 'arn:aws:s3:::${AuditBucketName}'
              - !Sub 'arn:aws:s3:::${AuditBucketName}/GuardDuty/${AWS::AccountId}/*'
          - Effect: Allow
            Action: sts:AssumeRole
            Resource: !ImportValue CentralAuditRole

  # Create a publishing destination configuration
  GuardDutyPublishingDestination:
    Type: AWS::GuardDuty::PublishingDestination
    Properties:
      DetectorId: !Ref MemberAccountGuardDuty
      DestinationType: S3
      DestinationProperties:
        DestinationArn: !Sub
        ↪  'arn:aws:s3:::${AuditBucketName}/GuardDuty/${AWS::AccountId}/*'
        KmsKeyArn: !ImportValue AuditS3BucketKMSKey

  # Enable Security Hub in member account
  MemberAccountSecurityHub:
    Type: AWS::SecurityHub::Hub
    Properties:
      # Enable security standards
      EnableDefaultStandards: true
      Tags:
        Environment: Member
        Purpose: SecurityMonitoring
```

10 11

10 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

11 Certain elements in the code listing have been anonymised or generalised to preserve confidentiality and align with disclosure requirements.

Secure Infrastructure Deployment

10_cc-secure-infra-attestation.yaml
CloudFormation stack defining the core Confidential Computing infrastructure for the *Omni-Aware* PoC. Includes all components required for SEV-SNP, Nitro Enclaves, `Vault` and Remote Attestation within a NATO-aligned DevSecOps deployment.

```
##############################################################################
# Project:          OmniAware - Next-Gen Defence Platform
# Component:        Confidential Computing Infrastructure & Remote Attestation
# Stack Name:       10_cc-secure-infra-attestation.yaml
# Description:      Deploys the secure baseline infrastructure for AMD SEV-SNP,
#                   Nitro Enclaves and HashiCorp Vault incl. Transit Engine,
#                   designed for automated Proof of Concept deployments (Dev-Mode)
#
# Author:           Valentin Pfeil
# Institution:      University of the Bundeswehr Munich (M.Sc. Computer Science)
# Supervision:      Prof. Dr. Wolfgang Hommel / Dr. Karl Fuerlinger
# Date:             2025-06-15
# License:          Research Use Only / Academic Distribution, Subject to Future Publication
# Format:           AWS CloudFormation (YAML)
#
# Tags:             Confidential Computing, Vault, SEV-SNP, Nitro Enclaves,
#                   Remote Attestation, DevSecOps, Defence Infrastructure
#
# Notes:
#   - Designed for deployment in AWS eu-west-1 (Ireland)
#   - Includes full stack: VPC, IAM, KMS, Vault, SEV-SNP, Enclaves
#   - Prepared for SSM-based debugging, Cloud-Native bootstrap and secure tagging
#
# Documentation:
#   - Master Thesis Chapter 4.2: Confidential Computing
##############################################################################

# Main Template for the secure infrastructure stack
# Unformal Ingredients: VPC, Subnets, SGs, SEV/Nitro EC2

AWSTemplateFormatVersion: "2010-09-09"
Description: "Remote Attestation Infrastructure (PoC) - AMD SEV-SNP + Nitro Enclaves + HashiCorp Vault"

Parameters:
  ProjectName:
    Type: "String"
    Default: "omniaware-cc"
    Description: "Project name for resource naming"
  Environment:
    Type: "String"
    Default: "dev"
    Description: "Environment name"

Mappings:
  RegionMap:
    eu-west-1:
      UbuntuAMI: "ami-01f23391a59163da9"           # Ubuntu 24.04 LTS, AMI Catalog, Quick Start AMI
      AmazonLinuxAMI: "ami-015b1e8e2a6899bdb"      # Amazon Linux 2023 (Nitro Enclaves-ready)
```

```
49
50  Resources:
51    # ====================================
52    # VPC Infrastructure, 1 /16 VPC, 1 Public Subnet, 1 Private Subnet
53    # ====================================
54    VPC:
55      Type: "AWS::EC2::VPC"
56      Properties:
57        CidrBlock: "10.0.0.0/16"
58        EnableDnsHostnames: true
59        EnableDnsSupport: true
60        Tags:
61          - Key: "Name"
62            Value: !Sub "${ProjectName}-${Environment}-vpc"
63
64    # Public Subnet for NAT Gateway
65    PublicSubnet:
66      Type: "AWS::EC2::Subnet"
67      Properties:
68        VpcId: !Ref VPC
69        CidrBlock: "10.0.1.0/24"
70        AvailabilityZone: !Select [0, !GetAZs ""]
71        MapPublicIpOnLaunch: true
72        Tags:
73          - Key: "Name"
74            Value: !Sub "${ProjectName}-${Environment}-public-subnet"
75
76    # Private Subnet for EC2 Instanzen
77    PrivateSubnet:
78      Type: "AWS::EC2::Subnet"
79      Properties:
80        VpcId: !Ref VPC
81        CidrBlock: "10.0.2.0/24"
82        AvailabilityZone: !Select [0, !GetAZs ""]
83        Tags:
84          - Key: "Name"
85            Value: !Sub "${ProjectName}-${Environment}-private-subnet"
86
87    # Internet Gateway for Outbound-Traffic (incl. NAT Gateway)
88    # The Internet Gateway allows instances in the public subnet to access the internet
89    # and is attached to the VPC. It is used by the NAT Gateway to route outbound traffic
90    # from private instances to the internet.
91    InternetGateway:
92      Type: "AWS::EC2::InternetGateway"
93      Properties:
94        Tags:
95          - Key: "Name"
96            Value: !Sub "${ProjectName}-${Environment}-igw"
97
98    AttachGateway:
99      Type: "AWS::EC2::VPCGatewayAttachment"
100     Properties:
101       VpcId: !Ref VPC
102       InternetGatewayId: !Ref InternetGateway
103
104   # NAT Gateway for Outbound Internet (incl. Internet Gateway)
105   # NAT Gateway needs one Elastic IP (EIP) to function properly
106   # This EIP is created in the same region as the VPC
107   # The NAT Gateway is used to allow private instances to access the internet for
      ↪  updates and other outbound traffic
108   # without exposing them to the public internet directly
```

```
109   # The NAT Gateway is created in the public subnet to allow it to route traffic
      ↪  through the Internet Gateway
110   NATGatewayEIP:
111     Type: "AWS::EC2::EIP"
112     DependsOn: AttachGateway
113     Properties:
114       Domain: vpc
115
116   NATGateway:
117     Type: "AWS::EC2::NatGateway"
118     Properties:
119       AllocationId: !GetAtt NATGatewayEIP.AllocationId
120       SubnetId: !Ref PublicSubnet
121       Tags:
122         - Key: "Name"
123           Value: !Sub "${ProjectName}-${Environment}-nat"
124
125   # Route Tables
126   # Public Route Table for Internet Gateway
127   # Private Route Table for NAT Gateway
128   # The public route table is associated with the public subnet and routes all outbound
      ↪  traffic to the Internet Gateway
129   # The private route table is associated with the private subnet and routes all
      ↪  outbound traffic to the NAT Gateway
130   # This allows instances in the private subnet to access the internet for updates and
      ↪  other outbound traffic
131   # without exposing them to the public internet directly
132   PublicRouteTable:
133     Type: "AWS::EC2::RouteTable"
134     Properties:
135       VpcId: !Ref VPC
136       Tags:
137         - Key: "Name"
138           Value: !Sub "${ProjectName}-${Environment}-public-rt"
139
140   PrivateRouteTable:
141     Type: "AWS::EC2::RouteTable"
142     Properties:
143       VpcId: !Ref VPC
144       Tags:
145         - Key: "Name"
146           Value: !Sub "${ProjectName}-${Environment}-private-rt"
147
148   PublicRoute:
149     Type: "AWS::EC2::Route"
150     DependsOn: AttachGateway
151     Properties:
152       RouteTableId: !Ref PublicRouteTable
153       DestinationCidrBlock: "0.0.0.0/0"
154       GatewayId: !Ref InternetGateway
155
156   PrivateRoute:
157     Type: "AWS::EC2::Route"
158     Properties:
159       RouteTableId: !Ref PrivateRouteTable
160       DestinationCidrBlock: "0.0.0.0/0"
161       NatGatewayId: !Ref NATGateway
162
163   PublicSubnetRouteTableAssociation:
164     Type: "AWS::EC2::SubnetRouteTableAssociation"
165     Properties:
```

```
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTable

  PrivateSubnetRouteTableAssociation:
    Type: "AWS::EC2::SubnetRouteTableAssociation"
    Properties:
      SubnetId: !Ref PrivateSubnet
      RouteTableId: !Ref PrivateRouteTable

  # ====================================
  # KMS Key for Test Secrets
  # ====================================
  # KMS Key for Remote Attestation Test Secrets
  # This KMS key is used to encrypt and decrypt test secrets for remote attestation
  # It is created with a key policy that allows the EC2 service to use it for
  ↪  decryption and data key generation
  # The key policy also allows the root user of the AWS account to manage the key
  # The key alias is created to provide a friendly name for the key
  # The key alias is used in the EC2 role policy to allow access to the key for
  ↪  attestation purposes
  # The key is used by the HashiCorp Vault server to encrypt and decrypt secrets
  # The key is also used by the EC2 instances to encrypt and decrypt attestation data
  # The key is created in the same region as the VPC and is used by the EC2 instances
  ↪  in the private subnet
  AttestationKMSKey:
    Type: "AWS::KMS::Key"
    Properties:
      Description: "KMS Key for Remote Attestation Test Secrets"
      KeyPolicy:
        Version: "2012-10-17"
        Statement:
          - Sid: "Enable IAM User Permissions"
            Effect: "Allow"
            Principal:
              AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root"
            Action: "kms:*"
            Resource: "*"
          - Sid: "Allow EC2 Service"
            Effect: "Allow"
            Principal:
              Service: "ec2.amazonaws.com"
            Action:
              - "kms:Decrypt"
              - "kms:GenerateDataKey"
              - "kms:CreateGrant"
            Resource: "*"

  AttestationKMSKeyAlias:
    Type: "AWS::KMS::Alias"
    Properties:
      AliasName: !Sub "alias/${ProjectName}-${Environment}-attestation"
      TargetKeyId: !Ref AttestationKMSKey

  # ====================================
  # IAM Roles for Session Manager
  # ====================================
  # IAM Role for EC2 Instances to allow Session Manager access
  # This role allows EC2 instances to be managed via AWS Systems Manager Session
  ↪  Manager
  # It includes the AmazonSSMManagedInstanceCore managed policy which provides the
  ↪  necessary permissions
```

```
222   # to communicate with the Systems Manager service
223   # The role is assumed by EC2 instances in the private subnet to allow them to be
      ↪  managed without direct SSH access
224   # The role is created with a trust policy that allows the EC2 service to assume the
      ↪  role
225   EC2Role:
226     Type: "AWS::IAM::Role"
227     Properties:
228       RoleName: !Sub "${ProjectName}-${Environment}-ec2-role"
229       AssumeRolePolicyDocument:
230         Version: "2012-10-17"
231         Statement:
232           - Effect: "Allow"
233             Principal:
234               Service: "ec2.amazonaws.com"
235             Action: "sts:AssumeRole"
236       ManagedPolicyArns:
237         - "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
238       Policies:
239         - PolicyName: "AttestationPermissions"
240           PolicyDocument:
241             Version: "2012-10-17"
242             Statement:
243               - Effect: "Allow"
244                 Action:
245                   - "kms:Decrypt"
246                   - "kms:GenerateDataKey"
247                   - "kms:CreateGrant"
248                 Resource: !GetAtt AttestationKMSKey.Arn
249               - Effect: "Allow"
250                 Action:
251                   - "ec2:CreateTags"
252                 Resource: !Sub
                  ↪  "arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:instance/*" # Allows
                  ↪  tagging of EC2 instances
253   EC2InstanceProfile:
254     Type: "AWS::IAM::InstanceProfile"
255     Properties:
256       Roles:
257         - !Ref EC2Role
258
259   # ====================================
260   # Security Groups
261   # ====================================
262   # Security Group for Internal Communication between Attestation Components
263   # This security group allows internal communication between the EC2 instances and the
      ↪  Vault server
264   # It allows inbound traffic on the SSH port (22) for management access
265   # It allows inbound traffic on the Vault API port (8200) for communication with the
      ↪  Vault server
266   # It allows inbound traffic on the attestation service ports (9000-9100) for
      ↪  communication between attestation components
267   # It allows outbound traffic to the internet for updates and other outbound traffic
268   # The security group is created in the same VPC as the EC2 instances and the Vault
      ↪  server
269   InternalSecurityGroup:
270     Type: "AWS::EC2::SecurityGroup"
271     Properties:
272       GroupName: !Sub "${ProjectName}-${Environment}-internal-sg"
273       GroupDescription: "Internal communication between attestation components"
274       VpcId: !Ref VPC
```

```
      SecurityGroupEgress:
        # Outbound Internet for Updates
        - IpProtocol: "-1"
          CidrIp: "0.0.0.0/0"
          Description: "Outbound Internet"
      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-internal-sg"

  # Self-referencing rules needs to be created after the security group is created
  InternalSSHRule:
    Type: "AWS::EC2::SecurityGroupIngress"
    Properties:
      GroupId: !Ref InternalSecurityGroup
      IpProtocol: tcp
      FromPort: 22
      ToPort: 22
      SourceSecurityGroupId: !Ref InternalSecurityGroup
      Description: "Internal SSH"

  InternalVaultRule:
    Type: "AWS::EC2::SecurityGroupIngress"
    Properties:
      GroupId: !Ref InternalSecurityGroup
      IpProtocol: tcp
      FromPort: 8200
      ToPort: 8200
      SourceSecurityGroupId: !Ref InternalSecurityGroup
      Description: "Vault API"

  InternalAttestationRule:
    Type: "AWS::EC2::SecurityGroupIngress"
    Properties:
      GroupId: !Ref InternalSecurityGroup
      IpProtocol: tcp
      FromPort: 9000
      ToPort: 9100
      SourceSecurityGroupId: !Ref InternalSecurityGroup
      Description: "Attestation Services"

  # ====================================
  # EC2 Key Pair
  # ====================================
  # EC2 Key Pair for SSH Access
  # This key pair is used to access the EC2 instances via SSH
  # It is created with a key name that includes the project name and environment
  # The key pair is created in the same region as the VPC and is used by the EC2
  ↪  instances in the private subnet

  EC2KeyPair:
    Type: "AWS::EC2::KeyPair"
    Properties:
      KeyName: !Sub "${ProjectName}-${Environment}-keypair"
      KeyType: "rsa"
      KeyFormat: "pem"

  # ====================================
  # EC2 Instances
  # ====================================
  # AMD SEV-SNP Instance
  # This instance is used for remote attestation using AMD SEV-SNP technology
```

```
335   # It is created with an instance type that supports SEV-SNP (c6a.large)
336   # The instance is created in the private subnet and is associated with the internal
      ↪  security group
337   # The instance is configured with user data to install the necessary tools for
      ↪  SEV-SNP attestation
338   # The instance is also configured with a user data script to install the SEV-SNP
      ↪  attestation tools
339   # The instance is created with a key pair for SSH access
340   # The instance is created with an IAM instance profile that allows it to access the
      ↪  KMS key for attestation
341   # The instance is tagged with the project name, environment and role for easy
      ↪  identification
342
343   # Amazon Linux 2023 - 1st AMD SEV-SNP Instance
344   SEVSNPLaunchTemplate:
345     Type: "AWS::EC2::LaunchTemplate"
346     Properties:
347       LaunchTemplateData:
348         InstanceType: "c6a.large"
349         ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AmazonLinuxAMI]
350         IamInstanceProfile:
351           Name: !Ref EC2InstanceProfile
352         KeyName: !Ref EC2KeyPair
353         SecurityGroupIds:
354           - !Ref InternalSecurityGroup
355         CpuOptions:
356           AmdSevSnp: "enabled"
357         UserData:
358           Fn::Base64: !Sub |
359             #!/bin/bash
360             set -e
361
362             # Set hostname
363             hostnamectl set-hostname OmniAware-EC2-SEV-SNP
364             echo '127.0.0.1 OmniAware-EC2-SEV-SNP' >> /etc/hosts
365
366             # Install development tools and dependencies
367             dnf groupinstall -y "Development Tools"
368             dnf install -y cmake git wget jq openssl-devel \
369               protobuf-compiler libtool autoconf automake \
370               kernel-headers kernel-devel awscli
371             # rust
372
373             # Install Rust (if not available globally)
374             rm -rf ~/.cargo ~/.rustup
375             curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
376             source $HOME/.cargo/env
377             rustup install stable
378             rustup default stable
379
380             # Fetch IMDSv2 token
381             TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" \
382               -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
383
384             # Retrieve instance metadata
385             INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
386               http://169.254.169.254/latest/meta-data/instance-id)
387             REGION=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
388               http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r
                  ↪  .region)
389
```

```
            # Tag EC2 instance with hostname
            aws ec2 create-tags --resources "$INSTANCE_ID" \
              --tags Key=Hostname,Value=OmniAware-EC2-SEV-SNP \
              --region "$REGION"

            # Set prompt for SSM
            echo 'export PS1="\u@\h:\w$ "' >> /etc/profile.d/ssm_prompt.sh
            chmod +x /etc/profile.d/ssm_prompt.sh

            # Install snpguest
            cd /opt
            git clone https://github.com/virtee/snpguest.git
            cd snpguest
            cargo build --release
            cp target/release/snpguest /usr/local/bin/

            # Install sevctl
            cd /opt
            git clone https://github.com/virtee/sevctl.git
            cd sevctl
            cargo build --release
            cp target/release/sevctl /usr/local/bin/

            # Optional: verify tools
            snpguest --help || echo "[x] snpguest not properly installed"
            sevctl --help || echo "[x] sevctl not installed"
            sevctl ok || echo "[x] SEVCTL OK check failed"

            # Vault CLI installation
            apt-get update -y
            apt-get install -y gnupg software-properties-common curl unzip
            curl -fsSL https://apt.releases.hashicorp.com/gpg | gpg --dearmor -o
            ↪ /usr/share/keyrings/hashicorp-archive-keyring.gpg

            echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
            ↪ https://apt.releases.hashicorp.com $(lsb_release -cs) main" | tee
            ↪ /etc/apt/sources.list.d/hashicorp.list

            # Vault Dependencies
            apt-get install -y python3-pip
            apt-get install -y python3-full


            # Set environment variable for Vault address
            echo 'export VAULT_ADDR="http://<!Ref VaultInstancePrivateIP>"' >>
            ↪ ~/.bashrc
    SEVSNPInstance:
      Type: "AWS::EC2::Instance"
      Properties:
        SubnetId: !Ref PrivateSubnet
        LaunchTemplate:
          LaunchTemplateId: !Ref SEVSNPLaunchTemplate
          Version: !GetAtt SEVSNPLaunchTemplate.LatestVersionNumber
        Tags:
          - Key: "Name"
            Value: !Sub "${ProjectName}-${Environment}-sev-snp"
          - Key: "Role"
            Value: "SEV-SNP-Attester"

    # Ubuntu 24.04 LTS - 2nd AMD SEV-SNP Instance
    SEVSNPUbuntuLaunchTemplate:
```

```
    Type: "AWS::EC2::LaunchTemplate"
    Properties:
      LaunchTemplateData:
        InstanceType: "c6a.large"
        ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", UbuntuAMI]
        IamInstanceProfile:
          Name: !Ref EC2InstanceProfile
        KeyName: !Ref EC2KeyPair
        SecurityGroupIds:
          - !Ref InternalSecurityGroup
        CpuOptions:
          AmdSevSnp: "enabled"
        UserData:
          Fn::Base64: !Sub |
            #!/bin/bash
            set -e

            # Set hostname - Done
            hostnamectl set-hostname OmniAware-EC2-SEV-SNP-Ubuntu
            echo '127.0.0.1 OmniAware-EC2-SEV-SNP-Ubuntu' >> /etc/hosts

            # Install tools & dependencies - Done
            snap install aws-cli --classic
            apt-get update && apt-get install -y \
              jq curl wget git cmake build-essential net-tools \
              linux-headers-$(uname -r) libssl-dev pkg-config \
              autoconf automake libtool protobuf-compiler libprotobuf-dev

            # Retrieve IMDSv2-Token, EC2 Metadata - Done
            TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" \
              -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

            INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
              http://169.254.169.254/latest/meta-data/instance-id)

            REGION=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
              http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r
              ↪  .region)

            # Tag the instance - Done
            aws ec2 create-tags --region "$REGION" \
              --resources "$INSTANCE_ID" \
              --tags Key=Hostname,Value=OmniAware-EC2-SEV-SNP-Ubuntu || echo "[x]
              ↪  Tagging failed"

            # Configure SSM prompt; Bash-Var for Prompt-Style
            # This sets the prompt for SSM sessions to show user, host and current
            ↪  directory
            echo 'export PS1="\u@\h:\w$ "' >> /etc/profile.d/ssm_prompt.sh
            chmod +x /etc/profile.d/ssm_prompt.sh


            # Install Rust using rustup - Done
            curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
            source $HOME/.cargo/env

            # Ensure Rust is available globally
            rustup install stable
            rustup default stable

            # Here, we install the necessary tools for SEV-SNP attestation
```

```
            # Machines crash irregularly while installing snpguest and sevctl
            # Install snpguest
            cd /opt
            git clone https://github.com/virtee/snpguest.git
            cd snpguest
            cargo build --release
            cp target/release/snpguest /usr/local/bin

            # Install sevctl
            cd /opt
            git clone https://github.com/virtee/sevctl.git
            cd sevctl
            cargo build --release
            cp target/release/sevctl /usr/local/bin

            # Optional: verify tools
            snpguest --help || echo "^^e2^^9d^^8c snpguest not properly installed"
            sevctl --help || echo "^^e2^^9d^^8c sevctl not installed"
            sevctl ok || echo "^^e2^^9d^^8c SEVCTL OK check failed"

            # Vault CLI installation
            apt-get update -y
            apt-get install -y gnupg software-properties-common curl unzip
            curl -fsSL https://apt.releases.hashicorp.com/gpg | gpg --dearmor -o
            ↪  /usr/share/keyrings/hashicorp-archive-keyring.gpg

            echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
            ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main" | tee
            ↪  /etc/apt/sources.list.d/hashicorp.list

            # Vault Dependencies
            apt-get install -y python3-pip
            apt-get install -y python3-full
  SEVSNPInstance2:
    Type: "AWS::EC2::Instance"
    Properties:
      SubnetId: !Ref PrivateSubnet
      LaunchTemplate:
        LaunchTemplateId: !Ref SEVSNPUbuntuLaunchTemplate
        Version: !GetAtt SEVSNPUbuntuLaunchTemplate.LatestVersionNumber
      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-sev-snp-ubuntu"
        - Key: "Role"
          Value: "SEV-SNP-Ubuntu-Attester"

  # Nitro Enclave Instance
  # This instance is used for remote attestation using Nitro Enclaves technology
  # It is created with an instance type that supports Nitro Enclaves (c5.xlarge)
  # The instance is created in the private subnet and is associated with the internal
  ↪  security group
  # The instance is configured with user data to install the necessary tools for Nitro
  ↪  Enclaves attestation
  # The instance is also configured with a user data script to install the Nitro
  ↪  Enclaves CLI and configure the enclave options
  # The instance is created with a key pair for SSH access
  # The instance is created with an IAM instance profile that allows it to access the
  ↪  KMS key for attestation
  # The instance is tagged with the project name, environment and role for easy
  ↪  identification
  NitroEnclaveInstance:
```

```
    Type: "AWS::EC2::Instance"
    Properties:
      InstanceType: "c5.2xlarge"  # Nitro Enclaves ben^^c3^^b6tigen mindestens .xlarge!
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", AmazonLinuxAMI]
      KeyName: !Ref EC2KeyPair
      SubnetId: !Ref PrivateSubnet
      SecurityGroupIds:
        - !Ref InternalSecurityGroup
      IamInstanceProfile: !Ref EC2InstanceProfile
      EnclaveOptions:
        Enabled: true
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          set -e

          # Set persistent hostname
          hostnamectl set-hostname OmniAware-EC2-Nitro-Enclave
          echo '127.0.0.1 OmniAware-EC2-Nitro-Enclave' >> /etc/hosts

          # Install dev tools
          dnf groupinstall -y "Development Tools"
          dnf install -y cmake openssl-devel git wget
          dnf remove curl-minimal -y
          dnf install curl -y
          dnf install git -y
          # Tag EC2 instance with its hostname
          dnf install -y awscli jq

          # Fetch IMDSv2 token
          TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" \
            -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

          # Retrieve instance metadata
          INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
            http://169.254.169.254/latest/meta-data/instance-id)
          REGION=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
            http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r
            ↪  .region)

          # Tag EC2 instance with hostname
          aws ec2 create-tags --resources "$INSTANCE_ID" \
            --tags Key=Hostname,Value=OmniAware-EC2-SEV-SNP \
            --region "$REGION"

          # Set prompt for SSM
          echo 'export PS1="\u@\h:\w$ "' >> /etc/profile.d/ssm_prompt.sh
          chmod +x /etc/profile.d/ssm_prompt.sh

          # Set prompt for SSM sessions
          echo 'export PS1="\u@\h:\w$ "' >> /etc/profile.d/ssm_prompt.sh
          chmod +x /etc/profile.d/ssm_prompt.sh

          # Install Nitro Enclaves CLI and dependencies
          dnf install -y aws-nitro-enclaves-cli aws-nitro-enclaves-cli-devel
          # Enable Nitro Enclaves for ec2-user
          usermod -aG ne ec2-user

          # Configuration of Nitro Enclaves Allocator and Enclaves
          # Create allocator config - Not needed for Nitro Enclaves, default is
          ↪  sufficient
```

```
          mkdir -p /etc/nitro_enclaves
          echo "memory_mib: 1024" > /etc/nitro_enclaves/allocator.yaml
          echo "cpu_count: 2" >> /etc/nitro_enclaves/allocator.yaml

          # Enable and start Nitro Enclaves allocator
          systemctl start nitro-enclaves-allocator.service
          systemctl enable nitro-enclaves-allocator.service

          # Install Docker
          systemctl enable docker
          systemctl start docker

          # Create Enclave Image
          docker build /usr/share/nitro_enclaves/examples/hello -t hello
          docker image ls
          nitro-cli build-enclave --docker-uri hello:latest --output-file hello.eif

          # Start Enclave
          ENCLAVE_ID=$(nitro-cli run-enclave \
            --eif-path example.eif \
            --memory 1024 \
            --cpu-count 2 \
            --debug-mode \
            --enclave-cid 16 \
            | jq -r '.EnclaveID')

          # Output Enclave ID
          echo "Enclave started with ID: $ENCLAVE_ID"

          # Check Enclave-Status
          nitro-cli describe-enclaves

          # Get Attestation Document - Not possible, because Nitro SDK is not
          ↪  implemented
          # nitro-cli get-attestation-document --enclave-id "$ENCLAVE_ID"
      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-nitro-enclave"
        - Key: "Role"
          Value: "Nitro-Enclave-Attester"
  # ====================================
  # Outputs
  # ====================================
  # These outputs provide information about the created resources
  # They can be used to reference the resources in other stacks or for management
  ↪  purposes
  # The outputs include VPC ID, Subnet IDs, Instance IDs, KMS Key ID and EC2 Key Pair
  ↪  Name
  # The outputs are also exported for use in other stacks
  # The outputs include commands to connect to the instances via Session Manager
  # The commands can be used to connect to the AMD SEV-SNP instance, Nitro Enclave
  ↪  instance and Vault instance
  # The outputs are tagged with the project name and environment for easy
  ↪  identification
  # The outputs are exported with a name that includes the stack name for easy
  ↪  reference
  # The outputs can be used in other stacks or for management purposes
  # The outputs include the ARN of the IAM role and instance profile created for the
  ↪  Vault instance
  # The outputs also include the security group ID for the Vault instance
  # AWS::StackName is own stack name, used for exporting outputs
```

```
671
672 Outputs:
673   VPCId:
674     Description: "VPC ID"
675     Value: !Ref VPC
676     Export:
677       Name: !Sub "${AWS::StackName}-VPC-ID"
678
679   PrivateSubnetId:
680     Description: "Private Subnet ID"
681     Value: !Ref PrivateSubnet
682     Export:
683       Name: !Sub "${AWS::StackName}-PrivateSubnet-ID"
684
685   InternalSecurityGroupId:
686     Description: "Security Group ID for internal Vault communication"
687     Value: !Ref InternalSecurityGroup
688     Export:
689       Name: !Sub "${AWS::StackName}-Internal-Security-Group-ID"
690
691   SEVSNPInstanceId:
692     Description: "AMD SEV-SNP Instance ID"
693     Value: !Ref SEVSNPInstance
694     Export:
695       Name: !Sub "${AWS::StackName}-SEV-SNP-Instance-ID"
696
697   SEVSNPInstanceId2:
698     Description: "AMD SEV-SNP Instance ID"
699     Value: !Ref SEVSNPInstance2
700     Export:
701       Name: !Sub "${AWS::StackName}-SEV-SNP-Instance-ID2"
702
703   NitroEnclaveInstanceId:
704     Description: "Nitro Enclave Instance ID"
705     Value: !Ref NitroEnclaveInstance
706     Export:
707       Name: !Sub "${AWS::StackName}-Nitro-Enclave-Instance-ID"
708
709   KMSKeyId:
710     Description: "KMS Key ID for Attestation"
711     Value: !Ref AttestationKMSKey
712     Export:
713       Name: !Sub "${AWS::StackName}-KMS-Key-ID"
714
715   EC2KeyPairName:
716     Description: "EC2 Key Pair Name"
717     Value: !Ref EC2KeyPair
718     Export:
719       Name: !Sub "${AWS::StackName}-KeyPair-Name"
720
721   EC2InstanceProfileName:
722     Description: "EC2 Instance Profile Name"
723     Value: !Ref EC2InstanceProfile
724     Export:
725       Name: !Sub "${AWS::StackName}-InstanceProfile-Name"
726
727   SessionManagerConnectCommands:
728     Description: "Commands to connect via Session Manager"
729     Value: !Sub |
730       # AMD SEV-SNP Instance:
731       aws ssm start-session --target ${SEVSNPInstance} --region ${AWS::Region}
```

```
      # AMD SEV-SNP-Ubuntu Instance:
      aws ssm start-session --target ${SEVSNPInstance2} --region ${AWS::Region}

      # Nitro Enclave Instance:
      aws ssm start-session --target ${NitroEnclaveInstance} --region ${AWS::Region}
```

12

Vault Deployment

15_cc-vault-poc.yaml
CloudFormation template for deploying a standalone `Vault` PoC instance. Includes configuration for JWT authentication, Transit Secret Engine and policy enforcement to support SEV-SNP and Nitro Enclave attestation workflows.

```
################################################################################
# Project:        OmniAware - Next-Gen Defence Platform
# Component:      Vault Key Management & Attestation Token Handling
# Stack Name:     15_cc-vault-poc.yaml
# Description:    Deploys the HashiCorp Vault Proof of Concept environment
#                 including JWT validation, Transit Secret Engine and policies
#                 to support remote attestation workflows.
# Author:         Valentin Pfeil
# Institution:    University of the Bundeswehr Munich (M.Sc. Computer Science)
# Supervision:    Prof. Dr. Wolfgang Hommel / Dr. Karl Fuerlinger
# Date:           2025-06-15
# License:        Research Use Only / Academic Distribution, Subject to Future
↪ Publication
# Format:         AWS CloudFormation (YAML)
################################################################################

AWSTemplateFormatVersion: "2010-09-09"
Description: "Vault Deployment PoC Stack - Integrates with Confidential Infrastructure
  provisioned by '10_cc-secure-infra-attestation.yaml' (via exported outputs)"

Parameters:
  InfraStackName:
    Type: "String"
    Default: "OmniAware-CC-SECURE-INFRA-ATTESTATION-Stack"
  ProjectName:
    Type: "String"
    Default: "omniaware-cc"
    Description: "Project name for resource naming"
  Environment:
    Type: "String"
    Default: "dev"
    Description: "Environment name"

Mappings:
  RegionMap:
    eu-west-1:
      UbuntuAMI: "ami-01f23391a59163da9"

Resources:
  VaultInstance:
    Type: AWS::EC2::Instance
    Properties:
```

12 This infrastructure template enables enclave-based remote attestation using AMD SEV-SNP and AWS Nitro Enclaves. It was designed and validated in a controlled PoC environment.

```
      InstanceType: t3.micro
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", UbuntuAMI]
      KeyName:
        !ImportValue
          Fn::Sub: "${InfraStackName}-KeyPair-Name"
      SubnetId:
        !ImportValue
          Fn::Sub: "${InfraStackName}-PrivateSubnet-ID"
      SecurityGroupIds:
        - !ImportValue
            Fn::Sub: "${InfraStackName}-Internal-Security-Group-ID"
      IamInstanceProfile:
        !ImportValue
          Fn::Sub: "${InfraStackName}-InstanceProfile-Name"
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          set -e

          hostnamectl set-hostname OmniAware-EC2-Vault
          echo '127.0.0.1 OmniAware-EC2-Vault' >> /etc/hosts

          snap install aws-cli --classic
          apt-get update && apt-get install -y jq curl wget git cmake build-essential \
            linux-headers-$(uname -r) libssl-dev pkg-config autoconf automake libtool \
            protobuf-compiler libprotobuf-dev gnupg software-properties-common unzip

          curl -fsSL https://apt.releases.hashicorp.com/gpg | gpg --dearmor -o
          ↪  /usr/share/keyrings/hashicorp-archive-keyring.gpg
          echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
          ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main" | tee
          ↪  /etc/apt/sources.list.d/hashicorp.list
          apt-get update && apt-get install -y vault net-tools

          useradd --system --home /etc/vault.d --shell /usr/sbin/nologin vault
          mkdir -p /opt/vault/data /etc/vault.d
          chown -R vault:vault /opt/vault /etc/vault.d

          # Write Vault Config
          cat <<VAULTCFGEOF > /etc/vault.d/vault.hcl
          storage "file" {
            path = "/opt/vault/data"
          }

          listener "tcp" {
            address     = "0.0.0.0:8200"
            tls_disable = true
          }

          api_addr = "http://127.0.0.1:8200"
          cluster_addr = "https://127.0.0.1:8201"
          ui = true
          VAULTCFGEOF

          # Write Systemd Unit File
          cat <<VAULTUNITEOF > /etc/systemd/system/vault.service
          [Unit]
          Description=HashiCorp Vault - A tool for managing secrets
          Documentation=https://www.vaultproject.io/docs/
          Requires=network-online.target
          After=network-online.target
```

```
          ConditionFileNotEmpty=/etc/vault.d/vault.hcl

          [Service]
          User=vault
          Group=vault
          ExecStart=/usr/bin/vault server -config=/etc/vault.d/vault.hcl
          Restart=on-failure

          [Install]
          WantedBy=multi-user.target
          VAULTUNITEOF

          systemctl daemon-reload
          systemctl enable vault
          systemctl start vault
          sleep 10

          export VAULT_ADDR="http://127.0.0.1:8200"
          vault operator init -key-shares=1 -key-threshold=1 >
          ↪  /home/ubuntu/vault-keys.txt
          UNSEAL_KEY=$(grep 'Unseal Key 1' /home/ubuntu/vault-keys.txt | awk '{print
          ↪  $NF}')
          ROOT_TOKEN=$(grep 'Initial Root Token' /home/ubuntu/vault-keys.txt | awk
          ↪  '{print $NF}')
          vault operator unseal "$UNSEAL_KEY"
          vault login "$ROOT_TOKEN"
          vault secrets enable transit
          vault write -f transit/keys/attestation-test

          # Create Transit Key for Attestation
          cat <<POLICY > /tmp/attestation-policy.hcl
          path "transit/encrypt/attestation-test" {
            capabilities = ["update"]
          }
          path "transit/decrypt/attestation-test" {
            capabilities = ["update"]
          }
          path "transit/keys/attestation-test" {
            capabilities = ["read"]
          }
          POLICY

          vault policy write attestation-policy /tmp/attestation-policy.hcl
          echo "export VAULT_ADDR=http://127.0.0.1:8200" >> /home/ubuntu/.bashrc
          echo "export VAULT_TOKEN=$ROOT_TOKEN" >> /home/ubuntu/.bashrc
          chown ubuntu:ubuntu /home/ubuntu/vault-keys.txt
          chmod 600 /home/ubuntu/vault-keys.txt

          # JWT Validation Setup
          vault auth enable jwt

          # Structure Setup
          mkdir -p /etc/vault.d/jwt
      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-vault"
        - Key: "Role"
          Value: "Vault-Server"

Outputs:
  VaultInstanceId:
```

```
158     Description: "Vault Instance ID"
159     Value: !Ref VaultInstance
160     Export:
161       Name: !Sub "${AWS::StackName}-Vault-Instance-ID"
162
163   VaultInstancePrivateIP:
164     Description: "Vault Instance Private IP"
165     Value: !GetAtt VaultInstance.PrivateIp
166     Export:
167       Name: !Sub "${AWS::StackName}-Vault-Private-IP"
```

Miscellaneous

PyJWT.py
SEV-SNP JWT Python Script for Custom Claim Injection and Token Generation. Demonstrates enclave-signed claim generation and token issuance for `Vault` JWT attestation workflows.

```
################################################################################
# Project:        OmniAware - Next-Gen Defence Platform
# Component:      SEV-SNP Remote Attestation - JWT Claim Generation
# Script Name:    PyJWT.py
# Description:    Generates a signed JSON Web Token (JWT) embedding attestation
#                 metadata for SEV-SNP workloads. The token includes a nonce,
#                 time-bound validity and a base64-encoded attestation report.
#                 This token is submitted to Vault for secure policy evaluation.
# Author:         Valentin Pfeil
# Institution:    University of the Bundeswehr Munich (M.Sc. Computer Science)
# Supervision:    Prof. Dr. Wolfgang Hommel / Dr. Karl Fuerlinger
# Date:           2025-06-15
# License:        Research Use Only / Academic Distribution, Subject to Future
↪  Publication
# Requirements:   PyJWT, Python 3.10+, valid SEV-SNP attestation report
↪  (guest_report.b64)
################################################################################

import jwt
from datetime import datetime, timedelta, timezone

# Load private key used for RS256 signing (e.g. from Nitro Enclave, HSM or secure
↪  store)
private_key = open("private.key", "r").read()

# Prepare payload including attestation metadata
payload = {
    "sub": "attester-001",           # Subject identifier of the enclave
    "aud": "vault",                  # Intended audience, e.g. Vault verifier
    "iss": "sev-snp",                # Issuer of the attestation (SEV-SNP runtime)
    "nonce": "abc123",               # Optionally SHA256(nonce) for replay protection
    "iat": datetime.now(timezone.utc),                          # Issued at timestamp
    "exp": datetime.now(timezone.utc) + timedelta(minutes=5),  # Expiry time
    "report": open("/tmp/guest_report.b64", "rb").read().hex() # SEV-SNP attestation
    ↪  report (base64)
}

# Encode the JWT using RS256 and output token
token = jwt.encode(payload, private_key, algorithm="RS256")
print(token)
```

13

13 This attestation report was generated by a prototype AMD SEV-SNP instance and has been minimally redacted.

INTERFACES - API GATEWAYS

Secure Ingest Gateway Deployment

20_secure-ingest-api.yaml
CloudFormation template for the Secure Ingest API. Provides a telemetry ingestion interface and demonstrates Zero Trust design principles within a NATO-aligned architecture.

```
################################################################################
# Project:        OmniAware - Next-Gen Defence Platform
# Component:      Sensor Ingest API (Secure Gateway Design)
# Stack Name:     20_ingest-secure-api.yaml
# Description:    Defines a secure and extensible API structure for ingesting
#                 telemetry and sensor data. Demonstrates interoperability and
#                 extensibility in a NATO-compliant (NAFv4) deployment model.
#
# Author:         Valentin Pfeil
# Institution:    University of the Bundeswehr Munich (M.Sc. Computer Science)
# Supervision:    Prof. Dr. Wolfgang Hommel / Dr. Karl Fuerlinger
# Date:           2025-06-15
# License:        Research Use Only / Academic Distribution, Subject to Future
↪  Publication
# Format:         OpenAPI 3.0 / YAML
#
# Tags:           Ingest API, Defence Data Platform, NAFv4, Interoperability,
#                 Sensor Data, NATO Compliance, API Gateway, Zero Trust
#
# Notes:
#   - Implements a secure, extensible ingest interface for telemetry and image data.
#   - Includes support policy-enforced access.
#   - Intended for deployment in attested environments (e.g. SEV-SNP / Nitro Enclave).
#
# Documentation:
#   - Refer to Master Thesis Chapter 4.3: Interfaces
#   - Use Case Context: Platform Health Monitoring (PHM)
#
# Referenced Dependencies:
#   - Vault Token Injection (for JWT verification layer)
#   - Security Group (Ingress/Logging Layer for Gateway Lambda)
################################################################################



AWSTemplateFormatVersion: "2010-09-09"
Description:
  "Secure Ingest API Stack for AWS Guild Account - based on GroupIT version,
  reduced to essential components for Proof of Concept and experimental Confidential
  ↪  Computing setup."

Parameters:
  Application:
    Type: "String"
    Default: "OmniAware"
  Stage:
    Type: "String"
    Default: "dev"
  Prefix:
    Type: "String"
    Default: "omniaware"
  Region:
```

```
    Type: "String"
    Default: "eu-west-1"
  VpcId:
    Type: "String"
    Description: "The ID of the VPC to deploy the API into"
  AdminRoleName:
    Type: "String"
    Default: "GuildAdminRole"
  SecurityAccountId:
    Type: "String"
    Default: "000000000000"
  CloudWatchKmsKeyId:
    Type: "String"
    Default: "dummy-key-id"

Resources:
  ApiExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub "${Application}-${Stage}-ApiGatewayRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: apigateway.amazonaws.com
            Action: sts:AssumeRole

  ApiExecutionPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: !Sub "${Application}-${Stage}-ApiGatewayPolicy"
      Roles:
        - !Ref ApiExecutionRole
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Resource: "*"

  ApiGatewayAccount:
    DependsOn: ApiExecutionPolicy
    Type: AWS::ApiGateway::Account
    Properties:
      CloudWatchRoleArn: !GetAtt ApiExecutionRole.Arn

  ApiLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/${Application}/${Stage}/IngestApi"
      RetentionInDays: 30
      KmsKeyId: !Sub
      ↪  "arn:aws:kms:${Region}:${SecurityAccountId}:key/${CloudWatchKmsKeyId}"

  IngestRestApi:
    Type: AWS::ApiGateway::RestApi
    Properties:
```

```
      Name: !Sub "${Application}-${Stage}-IngestApi"
      Description: "Private API for telemetry data ingestion (PoC)"
      FailOnWarnings: true
      BinaryMediaTypes:
        - "application/json"
      EndpointConfiguration:
        Types:
          - "REGIONAL"

  RootResource:
    Type: AWS::ApiGateway::Resource
    Properties:
      RestApiId: !Ref IngestRestApi
      ParentId: !GetAtt IngestRestApi.RootResourceId
      PathPart: v1

  TelemetryResource:
    Type: AWS::ApiGateway::Resource
    Properties:
      RestApiId: !Ref IngestRestApi
      ParentId: !Ref RootResource
      PathPart: telemetry

  TelemetryModel:
    Type: AWS::ApiGateway::Model
    Properties:
      RestApiId: !Ref IngestRestApi
      ContentType: "application/json"
      Name: "TelemetryDataModel"
      Description: "Schema model for ingest data"
      Schema:
        "$schema": "http://json-schema.org/draft-04/schema#"
        type: "object"
        properties:
          timestamp:
            type: string
          payload:
            type: object

  TelemetryValidator:
    Type: AWS::ApiGateway::RequestValidator
    Properties:
      RestApiId: !Ref IngestRestApi
      Name: "TelemetryValidator"
      ValidateRequestBody: true
      ValidateRequestParameters: false

  TelemetryMethod:
    Type: AWS::ApiGateway::Method
    Properties:
      RestApiId: !Ref IngestRestApi
      ResourceId: !Ref TelemetryResource
      HttpMethod: POST
      AuthorizationType: "AWS_IAM"
      RequestModels:
        "application/json": !Ref TelemetryModel
      RequestValidatorId: !Ref TelemetryValidator
      Integration:
        Type: MOCK
        IntegrationResponses:
          - StatusCode: 200
```

```
        RequestTemplates:
          "application/json": "{\"statusCode\": 200}"
      MethodResponses:
        - StatusCode: 200

  IngestDeployment:
    DependsOn: TelemetryMethod
    Type: AWS::ApiGateway::Deployment
    Properties:
      RestApiId: !Ref IngestRestApi
      Description: "Initial deployment"

  IngestStage:
    DependsOn: ApiGatewayAccount
    Type: AWS::ApiGateway::Stage
    Properties:
      StageName: !Ref Stage
      RestApiId: !Ref IngestRestApi
      DeploymentId: !Ref IngestDeployment
      Description: "PoC stage"
      AccessLogSetting:
        DestinationArn: !GetAtt ApiLogGroup.Arn
      MethodSettings:
        - HttpMethod: POST
          ResourcePath: /v1/telemetry
          LoggingLevel: INFO
          MetricsEnabled: true

      Tags:
        - Key: "Name"
          Value: !Sub "${ProjectName}-${Environment}-secure-ingest-api"
        - Key: "Role"
          Value: "Secure-Ingest-API"

Outputs:
  IngestApiInvokeUrl:
    Description: "Invoke URL for the ingest API"
    Value: !Sub "https://${IngestRestApi}.execute-api.${AWS::Region}.amazonaws.com/${S
    ↪  tage}/v1/telemetry"
    Export:
      Name: !Sub "${Application}-${Stage}-IngestApiInvokeUrl"
```

14

14 The implementation of this module was based on code contributions by colleagues in the *OmniAware* project and has been integrated with their explicit permission.

Secure Ingest Gateway Validation

Telemetry Ingest API

NGVA - Sample JSON Data Model, simplified
The following JSON structure represents a minimal, schema-aligned telemetry payload adhering to the NGVA data model for secure ingestion into the pipeline.

```
{
  "DateTime": {},
  "Vehicle_Configuration": {
    "Actual_Configured_Vehicle": {
      "vehicleId": {}
    }
  }
}
```

NGVA - Sample JSON Data Model
This extended and partially simplified JSON schema implements a representative data payload for secure telemetry ingestion based on the NGVA specification. It was developed to emulate realistic conditions and includes structural fields for vehicle configuration, usage and condition monitoring, threshold definitions and publication metadata.

```
{
  "DateTime": {},
  "Vehicle_Configuration": {
    "Actual_Configured_Vehicle": {
      "vehicleId": {},
      "battleOverride": {},
      "equipmentPowerLevel": {},
      "operatingMode": {}
    }
  },
  "Usage_And_Condition_Monitoring": {
    "Monitored_Characteristic": {
      "engine_temperature": {
        "value": {}
      }
    },
    "Monitored_Characteristic_Specification": {
      "engine_temperature_specification": {
        "unit": {},
        "descriptor": {},
        "publishingIntervalInSeconds": {},
        "characteristicKind": {}
      }
    },
    "Threshold": {
      "engine_temperature_threshold_min": {
        "name": {},
        "value": {},
        "type": {},
        "maxDuration": {},
        "maxNumberOfRepetitions": {}
      },
      "engine_temperature_threshold_max": {
        "name": {},
        "value": {},
        "type": {},
        "maxDuration": {},
```

```
        "maxNumberOfRepetitions": {}
      }
    },
    "Threshold_Specification": {
      "engine_temperature_min_threshold_specification": {
        "valueThresholdSupported": {},
        "durationThresholdSupported": {},
        "repetitionsThresholdSupported": {}
      },
      "engine_temperature_max_threshold_specification": {
        "valueThresholdSupported": {},
        "durationThresholdSupported": {},
        "repetitionsThresholdSupported": {}
      }
    }
  },
  "Navigation_Reference": {
    "Position": {
      "currentPosition": {}
    },
    "Position_Uncertainty": {
      "value": {},
      "type": {}
    },
    "Speedometer": {
      "measuredSpeed": {}
    }
  }
}
```

JSON Schema Draft-04 - Sample Telemetry Schema for Test Purposes

This JSON schema was developed for testing purposes to emulate structured telemetry payloads with JSON-style message formats. It serves as a prototype to validate basic schema conformance, field structure and processing logic within the ingestion pipeline under controlled test conditions [3].

```
{
    "id": "http://json-schema.org/draft-04/schema#",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Core schema meta-schema",
    "definitions": {
        "schemaArray": {
            "type": "array",
            "minItems": 1,
            "items": { "$ref": "#" }
        },
        "positiveInteger": {
            "type": "integer",
            "minimum": 0
        },
        "positiveIntegerDefault0": {
            "allOf": [ { "$ref": "#/definitions/positiveInteger" }, { "default": 0 } ]
        },
        "simpleTypes": {
            "enum": [ "array", "boolean", "integer", "null", "number", "object",
            ↪  "string" ]
        },
        "stringArray": {
            "type": "array",
            "items": { "type": "string" },
            "minItems": 1,
```

```
            "uniqueItems": true
        }
    },
    "type": "object",
    "properties": {
        "id": {
            "type": "string"
        },
        "$schema": {
            "type": "string"
        },
        "title": {
            "type": "string"
        },
        "description": {
            "type": "string"
        },
        "default": {},
        "multipleOf": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        },
        "maximum": {
            "type": "number"
        },
        "exclusiveMaximum": {
            "type": "boolean",
            "default": false
        },
        "minimum": {
            "type": "number"
        },
        "exclusiveMinimum": {
            "type": "boolean",
            "default": false
        },
        "maxLength": { "$ref": "#/definitions/positiveInteger" },
        "minLength": { "$ref": "#/definitions/positiveIntegerDefault0" },
        "pattern": {
            "type": "string",
            "format": "regex"
        },
        "additionalItems": {
            "anyOf": [
                { "type": "boolean" },
                { "$ref": "#" }
            ],
            "default": {}
        },
        "items": {
            "anyOf": [
                { "$ref": "#" },
                { "$ref": "#/definitions/schemaArray" }
            ],
            "default": {}
        },
        "maxItems": { "$ref": "#/definitions/positiveInteger" },
        "minItems": { "$ref": "#/definitions/positiveIntegerDefault0" },
        "uniqueItems": {
            "type": "boolean",
```

```
            "default": false
        },
        "maxProperties": { "$ref": "#/definitions/positiveInteger" },
        "minProperties": { "$ref": "#/definitions/positiveIntegerDefault0" },
        "required": { "$ref": "#/definitions/stringArray" },
        "additionalProperties": {
            "anyOf": [
                { "type": "boolean" },
                { "$ref": "#" }
            ],
            "default": {}
        },
        "definitions": {
            "type": "object",
            "additionalProperties": { "$ref": "#" },
            "default": {}
        },
        "properties": {
            "type": "object",
            "additionalProperties": { "$ref": "#" },
            "default": {}
        },
        "patternProperties": {
            "type": "object",
            "additionalProperties": { "$ref": "#" },
            "default": {}
        },
        "dependencies": {
            "type": "object",
            "additionalProperties": {
                "anyOf": [
                    { "$ref": "#" },
                    { "$ref": "#/definitions/stringArray" }
                ]
            }
        },
        "enum": {
            "type": "array",
            "minItems": 1,
            "uniqueItems": true
        },
        "type": {
            "anyOf": [
                { "$ref": "#/definitions/simpleTypes" },
                {
                    "type": "array",
                    "items": { "$ref": "#/definitions/simpleTypes" },
                    "minItems": 1,
                    "uniqueItems": true
                }
            ]
        },
        "format": { "type": "string" },
        "allOf": { "$ref": "#/definitions/schemaArray" },
        "anyOf": { "$ref": "#/definitions/schemaArray" },
        "oneOf": { "$ref": "#/definitions/schemaArray" },
        "not": { "$ref": "#" }
    },
    "dependencies": {
        "exclusiveMaximum": [ "maximum" ],
        "exclusiveMinimum": [ "minimum" ]
```

```
    },
    "default": {}
}
```

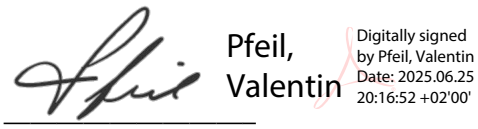
Image Ingest API

`Image to Base64 - Encoding`
Illustrates the Base64 encoding of binary image data for transmission via the Secure Ingest API.

```
cat <Insert Image Path> | base64 > <Insert Base64 File Path>.txt
```

Hereby, I assure that the attached work has been created autonomously and without any support. The quotations have been done properly and I do not have used any sources beyond the scope of my bibliography.

Furthermore, I have taken the instructions for the academic and final thesis into account and granted my University of the Bundeswehr Munich the right of use.



Signature