SANTA CATARINA STATE UNIVERSITY - UDESC
COLLEGE OF TECHNOLOGICAL SCIENCE - CCT
GRADUATE PROGRAM IN APPLIED COMPUTING - PPGCAP

HENRIQUE ZANELA COCHAK

CONTAINER ORCHESTRATION IMPACT ON SPIFFE IDENTITY ARTIFACTS: A PERFORMANCE ANALYSIS OF DOCKER VS KUBERNETES

JOINVILLE
2025

HENRIQUE ZANELA COCHAK

CONTAINER ORCHESTRATION IMPACT ON SPIFFE IDENTITY ARTIFACTS: A PERFORMANCE ANALYSIS OF DOCKER VS KUBERNETES

Master thesis presented to the Graduate Program in Applied Computing of the College of Technological Science from the Santa Catarina State University, as a partial requisite for receiving the Master's degree in Applied Computing.

Supervisor: Dr. Charles Christian Miers

JOINVILLE

2025

Ficha catalográfica elaborada pelo programa de geração automática da Biblioteca Universitária Udesc,

com os dados fornecidos pelo(a) autor(a)

Cochak, Henrique
Container Orchestration Impact on SPIFFE Identity Artifacts: A Performance Analysis of Docker vs Kubernetes / Henrique Cochak. -- 2025.
80 p.

Orientador: Charles Christian Miers
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2025.

1. Cloud Computing. 2. Microservices. 3. Security. 4. Benchmarking. I. Christian Miers, Charles. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

Cochak, Henrique Zanela

Container Orchestration Impact on SPIFFE Identity Artifacts: A Performance Analysis of Docker vs Kubernetes/ Henrique Zanela Cochak. – 2025   81 p.

Supervisor: Dr. Charles Christian Miers

Thesis (Master's degree) – Santa Catarina State University, College of Technological Science, Graduate Program in Applied Computing, Joinville, 2025.

1. Cloud Computing. 2. Microservices. 3. Security. 4. Benchmarking. I. Miers, Charles Christian. II. Santa Catarina State University, College of Technological Science, Graduate Program in Applied Computing. III. Title.

Henrique Zanela Cochak

Container Orchestration Impact on SPIFFE Identity Artifacts: A Performance Analysis of Docker vs Kubernetes

Master thesis presented to the Graduate Program in Applied Computing of the College of Technological Science from the Santa Catarina State University, as a partial requisite for receiving the **Master's degree in Applied Computing**.

Joinville, 25th November 2025

I dedicate this work to my family, friends, and professors who guided me to accomplish this.

ACKNOWLEDGMENTS

I dedicate this work and the whole effort required to it. First, I am grateful to my parents for all their support and patience in helping me reach this step. I want to give a special thanks to Prof. Charles for all the opportunities he has offered me so far. One of them is meeting such a great group of people in the LARC-USP, specifically Prof. Marcos Simplicio. Another person I believe has a great influence on this work is Marco Antonio Marques. He is a person with tenacity and resilience to reach his goals. Today, I find him as a mentor between collaborations, and I thank him for all the help provided so far. Furthermore, I would like to thank the other professors at Graduate Program in Applied Computing (PPGCAP) and collaborators at Laboratório de Processamento Paralelo e Distribuído (LabP2D), who directly or indirectly participated in this process of my evolution as an academic.

ABSTRACT

This thesis addresses gaps in the SPIFFE-IdT project by consolidating four independent security artifact implementations into a unified architecture and evaluating their performance in container-orchestrated environments. Comprehensive benchmarking across Docker Compose and Kubernetes reveals that Kubernetes environment demonstrate 15-35% lower CPU utilization while requiring 15-30% higher memory allocation. Network analysis identifies connection establishment as the primary source of orchestration overhead, with non-pooled operations experiencing up to 22.6% higher latency in Kubernetes. Connection pooling effectively neutralizes this overhead, reducing performance impact to below 5% for modes implementing connection reuse. LSVID consistently demonstrates the lowest resource footprint, validating its lightweight design. The findings reveal that connection operations dominate performance behavior in security-focused deployments. In particular, frequent mTLS and JWKS exchanges impose substantial overheads, which can be effectively mitigated through connection pooling. This underscores that optimizing connection reuse is key to achieving efficient and scalable identity management.

Keywords: Cloud computing, Microservices, Security, Benchmarking.

RESUMO

Esta tese aborda lacunas no projeto SPIFFE-IdT ao consolidar quatro implementações independentes de artefatos de segurança em uma arquitetura unificada e avaliar seu desempenho em ambientes orquestrados por contêineres. Análises de desempenho entre Docker Compose e Kubernetes revelam que o ambiente Kubernetes demonstra uma utilização menor de CPU (15-35%) menor enquanto requer uma maior alocação de memória (15-30%). A análise de rede identifica o estabelecimento de conexões como a principal fonte de sobrecarga de orquestração, com operações sem pooling experimentando até 22,6% maior latência no Kubernetes. O pooling de conexões mitiga essa sobrecarga, reduzindo o impacto no desempenho para abaixo de 5% nos modos que implementam reutilização de conexões. O LSVID demonstra consistentemente o menor consumo de recursos, validando seu design leve. Os resultados revelam que operações de conexão dominam o comportamento de desempenho em implantações focadas em segurança. Em particular, trocas frequentes de mTLS e JWKS impõem sobrecargas substanciais, que podem ser efetivamente mitigadas através do pooling de conexões. Isso ressalta que otimizar a reutilização de conexões é fundamental para alcançar gerenciamento de identidade eficiente e escalável.

Palavras-chaves: Computação em nuvem, Microsserviços, Segurança, Análise de desempenho.

LIST OF FIGURES

LIST OF TABLES

API Application Programming Interface

CISA U.S. Department of Homeland Security, Cybersecurity and Infrastructure Security Agency

CNI Container Network Interface

DNS Domain Name System

CaaS Containers as a Service

CNCF Cloud Native Computing Foundation

DVID Delegated Assertion SVID

gRPC gRPC Remote Procedure Call

HPE Hewlett Packard Enterprise

IaaS Infrastructure as a Service

IAM Identity and Access Management

ICAM Identity, Credential, and Access Management

IDM Identity Management

IdP Identity Provider

IFC Instituto Federal Catarinense

IMS Identity Management System

JWS JSON Web Signature

JSON JavaScript Object Notation

JWT JSON Web Token

LabP2D Laboratório de Processamento Paralelo e Distribuído

LSVID Lightweight SVID

MSA Microservices Architecture

mTLS mutual Transport Layer Security

NIST National Institute of Standards and Technology

PaaS Platform as a Service

PoC	Proof of Concept
PPGCAP	Graduate Program in Applied Computing
REST	Representational State Transfer
RSA	Rivest-Shamir-Adleman
SaaS	Software as a Service
SPIFFE	Secure Production Identity Framework for Everyone
SPIRE	SPIFFE Runtime Environment
SVID	SPIFFE Verifiable Identity Document
TLS	Transport Layer Security
TTP	Trusted Third Party
UCaaS	Unified Communications as a Service
UDESC	Universidade do Estado de Santa Catarina
URI	Uniform Resource Identifier
URI	Uniform Resource Locator
USP	Universidade de São Paulo
ZKP	Zero Knowledge Proof
IPC	Inter-Process Communication
JWKS	JSON Web Key Sets
TSDB	Time Series Database

CONTENTS

1 INTRODUCTION

Today, cloud computing is one of the leading technologies enabling the dynamic provisioning of processing, storage, and networking resources. Cloud providers may employ different virtualization technologies, including virtual machines and containers, to abstract the physical infrastructure supplying those resources. However, the benefits of cloud computing come with administrative challenges regarding the security and privacy of provided services (GONZALEZ et al., 2012; KUMAR; GOYAL, 2019; ALLIANCE, 2023).

In particular, Secure Production Identity Framework for Everyone (SPIFFE) provides a set of open-source standards for secure workload identification and authentication, specifically designed for dynamic, heterogeneous, and federated cloud environments (FELDMAN et al., 2020). Its reference implementation, SPIRE, is widely available to the academic community and the software industry, further expanding the toolset available for secure identity management. SPIFFE stands out due to its unique focus on securing the identities of workloads, making SPIFFE particularly well-suited for cloud computing environments, in which distributed workloads, often executing with microservice architecture, need to authenticate each other to establish secure connections mutually. However, even SPIFFE has its solution scope well defined, and the context covered by its main security document, SVIDs, is limited to the identity of the workload directly sending or receiving a message.

This led to the development of the SPIFFE-IdT project, aiming to enhance and extend the SPIFFE / SPIRE frameworks, focusing on security, performance, and functionality. The project was divided into three phases (Spanning from 2021 to 2023), each developing new security documents or models to fulfill predefined use cases in the project. Several papers derived from it, including three papers written as a byproduct of benchmarking tests to validate the efficiency and gather tangible evidence of the artifacts created in each phase. To further build upon the foundations established by the PoC solutions, the research highlights the necessity of a richer and more complex operational environment closely aligned with the real-world applications of the SPIFFE framework. While the initial tests provided valuable baseline insights, they represent only a fraction of the intricate scenarios encountered in dynamic, federated environments. A more thriving environment would include diverse workloads operating in interconnected domains, handling high volumes of interactions, and leveraging various levels of trust and authentication. This type of setup better reflects the challenges and opportunities of modern cloud-native architectures, which demand robust and scalable identity management solutions.

Therefore, this master thesis proposes continuing the SPIFFE-IdT project, with divergent goals originally from it, to enhance SPIFFE capabilities with new security documents and models in mind. The first goal is to ensure that each artifact is scalable and can effectively handle the demands of modern, dynamic environments. Scalability is a critical aspect of cloud-native architectures, in which workloads can rapidly grow in volume and complexity. The existing codebase will undergo a careful review and refactoring process to achieve this. This will focus on supporting efficient scaling, both in terms of performance and resource management. As part of this process, each artifact's current design will be revisited to identify potential bottlenecks or limitations that could hinder scalability. Addressing these issues early aims to optimize the system's ability to cope better with increasing workload demands and ensure smooth, scalable performance across diverse environments.

Subsequently, each artifact will be tested within an orchestrated environment using the Kubernetes platform, a key tool in the Cloud Native Computing Foundation (CNCF) ecosystem. This will be a controlled setting to evaluate how the refactored artifacts perform closer to real-world, cloud-native conditions. Following these tests, a new series of benchmarking will be conducted to assess the updated artifacts' practicality and effectiveness. These benchmarks will provide valuable insights into the artifacts' scalability, resource consumption, and overall performance in an operational environment, helping to validate the improvements made during the refactoring process. For both goals, the observability tool Prometheus, another graduated CNCF tool, is employed to gather performance and resource usage data. However, each goal will leverage Prometheus differently. Thus, for the scenario without orchestration, Prometheus will monitor the system's resource consumption, including CPU, memory, and network usage, to identify bottlenecks and assess the efficiency of the refactored artifacts under varying load conditions. During the benchmarking phase in the Kubernetes environment, Prometheus will track predefined metrics, which will later be analyzed.

The primary objective of this thesis is refine and extend the work conducted under the SPIFFE-IdT project by reassessing the PoCs artifacts, and adapting them for container-orchestrated environments, specifically Kubernetes. Unlike the original project, which focused on developing security documents and models, this research shifts its focus toward evaluating their feasibility and performance in the real world.

A major contribution of this work is the benchmarking and performance evaluation of the refactored artifacts within containerized environments. This involves systematically measuring how container orchestration affects resource utilization, computational overhead, and system responsiveness under different workloads. By leveraging Prometheus, a CNCF observability tool, this research will collect detailed metrics, providing empirical data to validate the proposed enhancements in the SPIFFE framework.

This research follows an applied methodology, systematically evaluating the scalability and performance of PoCs artifacts in a containerized environment. The methodology is structured around a three-phase test plan, ensuring a thorough assessment before advancing to the next stage. The first phase of the test plan involves the reevaluation of artifacts created. This step includes analyzing their current implementation, identifying dependencies, and assessing potential scalability challenges. The second phase focuses on benchmarking the reevaluated artifacts in a controlled environment with a group of predefined metrics, creating a baseline measurement. This baseline will later serve as a reference point when evaluating the impact of container orchestration. The third phase involves the adaptation and integration of the new artifacts into a Kubernetes environment, followed by another round of benchmarking. This step evaluates the feasibility of running these artifacts in cloud-native architectures and investigates whether container orchestration introduces significant performance tradeoffs. Kubernetes-specific metrics, collected using Prometheus, will provide empirical insights into resource efficiency, scaling behavior, and overall system responsiveness.

The work is organized as follows. Chapter 2 addresses the fundamental concepts of cloud computing, identity, and access systems. Furthermore, it explores how the SPIFFE-IdT Project was developed and provides the requirements explored in related works and the motivation for this master thesis. Chapter 3 proposes the research plan, defining the metrics to be measured, how they will be collected, and the testbed environment for the unification and evaluation of the artifacts. Chapter 4 presents the implementation of the unified architecture, the experimental results, and a comprehensive data analysis of the performance characteristics across both Docker Compose and Kubernetes deployments. Chapter 5 discusses key findings, implications, and outlines directions for future research in SPIFFE-based identity management for container-orchestrated environments.

2 FUNDAMENTAL CONCEPTS

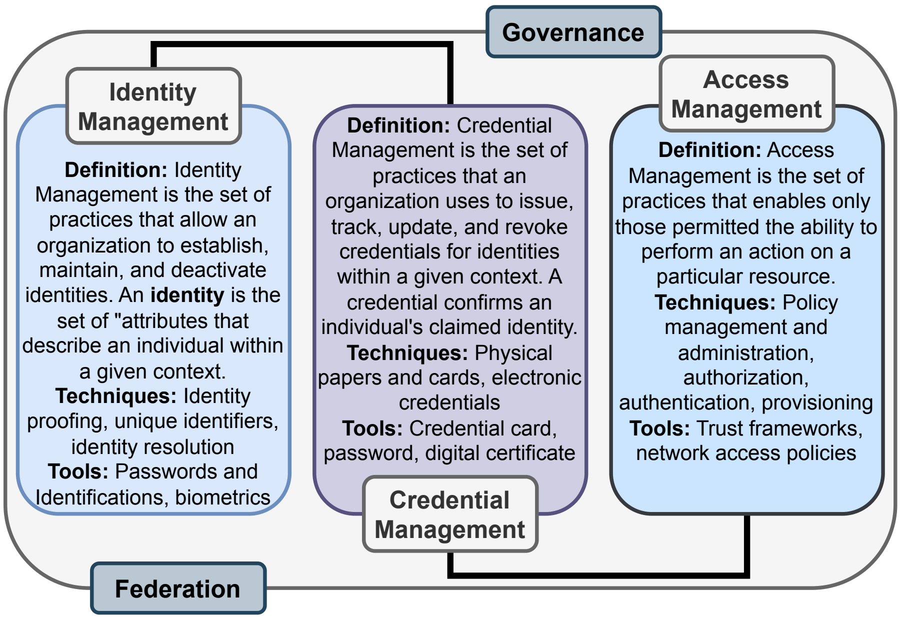
According to the US government agency National Institute of Standards and Technology (NIST), cloud computing is defined as a model for enabling ubiquitous and convenient on-demand network access to a shared pool of computing resources, e.g., networks, servers, storage, applications, services, that can be rapidly provisioned and launched with minimal management effort or interaction from the cloud provider (MELL; GRANCE, 2011). Cloud computing has emerged as one of the fastest-growing technologies in computer science, playing a crucial role in transforming personal, governmental, and professional activities. Some of the latest features include performance improvements related to better management of cloud provider resources and increased security since providers also offer a broad set of policies and control technologies, strengthening the security of the organization's data, applications, and infrastructure (MICROSOFT, 2024a).

There are several cloud computing service models, each providing a set of resources to deliver more specialized services. The most widely accepted model is that of NIST, classifying cloud computing into three service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (MELL; GRANCE, 2011). However, there are more recent models, considered to be more niched cases, such as Containers as a Service (CaaS) (REDHAT, 2024) and Unified Communications as a Service (UCaaS) (MICROSOFT, 2024b). In addition to the service model, the same document (Mell e Grance (2011)) also classifies clouds into deployment models: private, community, public, and hybrid. With the expansion of these cloud computing models, a similar growth in coding architecture models that can benefit from it is seen. Here, Microservices Architecture (MSA) advocates implementing small-scale and independently distributed services, decomposing the application into a set of small services and making them communicate with each other through light weight mechanisms (e.g., Representational State Transfer (REST)ful Application Programming Interface (API) or stream-based communications) (FOWLER; LEWIS, 2014).

While MSA offers flexibility and scalability, it also introduces new security challenges. Microservice architecture does not simplify an application; it only distributes the application logic into multiple smaller components, resulting in a much more complex network interaction model between components (SUN; NANDA; JAEGER, 2015). Each microservice must be individually secured, authenticated, and authorized, increasing the complexity of managing identity, access control, and data protection across numerous interconnected components. This distributed model also broadens the attack surface, as each service endpoint represents a potential entry point for unauthorized access. Consequently, robust security practices, such as mutual authentication, encryption, and fine-grained access control, are critical to maintaining a secure microservices environment.

To address the security complexities introduced by MSA, organizations utilize ICAM frameworks as a foundation, along with other security strategies, to ensure comprehensive protection. ICAM, more broadly known as Identity and Access Management (IAM), provides a structured approach for defining, enforcing, and managing access policies across distributed systems. Only authorized users and services can access specific resources and functionalities. According to U.S. Department of Homeland Security, Cybersecurity and Infrastructure Security Agency (CISA) (SECURITY; (CISA), 2023a), ICAMs are essential for securing dynamic environments (e.g., microservices, federations) by offering identity verification, credential management, and access controls across both human and non-human entities. CISA provides an exceptional briefing, illustrated in Figure 1.

Figure 1 – ICAM definition.


Governance
Identity Management
Definition: Identity Management is the set of practices that allow an organization to establish, maintain, and deactivate identities. An identity is the set of "attributes that describe an individual within a given context.
Techniques: Identity proofing, unique identifiers, identity resolution
Tools: Passwords and Identifications, biometrics
Definition: Credential Management is the set of practices that an organization uses to issue, track, update, and revoke credentials for identities within a given context. A credential confirms an individual's claimed identity.
Techniques: Physical papers and cards, electronic credentials
Tools: Credential card, password, digital certificate
Credential Management
Access Management
Definition: Access Management is the set of practices that enables only those permitted the ability to perform an action on a particular resource.
Techniques: Policy management and administration, authorization, authentication, provisioning
Tools: Trust frameworks, network access policies
Federation

Adapted from (SECURITY; (CISA), 2023b).

To meet the security demands of microservices, ICAM frameworks go beyond basic access control, incorporating granular policies that adapt to complex, distributed architectures. Given the diversity of users, devices, and services interacting within microservice environments, ICAM offers an essential layer of control, allowing organizations to dynamically adjust access privileges and apply consistent identity management practices across multiple domains, also known as federations. A federated approach promotes mutual trust and interoperability between clouds and communities of interest, having the autonomy to set agreed-upon rules for establishing trust and conditions for sharing information (SECURITY; (CISA), 2023b), defining what is referenced as governance: the complete framework of rules, policies, and processes that the issuing authority follows to manage and oversee the issuing process, ensuring that the authority maintains high standards of accuracy, security, and accountability when verifying identities.

ICAM frameworks are the foundational processes of digital identity management, authentication, and authorization. According to the Digital Identity Guidelines published by NIST (GRASSI; GARCIA; FENTON, 2017), a digital identity is defined as the unique, verifiable representation of a subject engaged in an online transaction. This identity must be unique within the context of a specific digital service, although it does not have to identify the subject uniquely in all contexts. Managing these digital identities encompasses a range of policies, tools, and mechanisms designed to oversee their lifecycle, which may include individuals, software, or hardware components. This management is essential for establishing trust within the system, enabling secure interactions among different entities.

Authentication plays a crucial role in this framework, as it verifies a claimed identity based on a pre-existing label from a mutually recognized namespace (ABOBA; WOOD, 2003). Digital signatures and public key certificates are commonly employed to facilitate authentication, which are issued by a Trusted Third Party (TTP) to ensure the correct association of a public key with a specific entity. In conjunction with authentication, authorization determines whether a particular right, such as access to specific resources, can be granted to the holder of a credential (ABOBA; WOOD, 2003). These processes, digital identity management, authentication, and authorization, create a robust framework that supports secure access and interaction within modern digital environments.

Due to its distributed and scalable nature, implementing strict and well-defined security measures is a common requirement, especially in cloud computing. Robust authentication and authorization protocols are critical to protect data and applications from unauthorized access and breaches. Among the available solutions for securing cloud environments, the SPIFFE framework (FELDMAN et al., 2020) stands out as a significant example of an Identity Management System (IMS), a specialized subset within IAM solution. As its classification suggests, SPIFFE is explicitly dedicated to identity management rather than broader access control tasks, defining an open framework and standards for identifying workloads and securing their communications. To reduce the risk of breaches through credential compromise, SPIFFE provides a strongly attested identity for authentication across the entire infrastructure, a key rotation mechanism, and addresses security needs by enabling, among other applications, verifiable mTLS connections between services. This ensures secure communication between workloads, no matter where they are deployed (FELDMAN et al., 2020), facilitating the establishment of a zero-trust architecture (ROSE et al., 2020), one of SPIFFE's core principles.

2.1 SPIFFE / SPIRE

SPIFFE is an open-source set of standards designed to establish interoperable software identity across diverse platforms and organizations, providing a uniform identity control plane across modern and heterogeneous infrastructure. It facilitates the automated issuance and validation of cryptographic identities for non-human entities, such as servers and services, enabling secure communication over networks (FELDMAN et al., 2020). The SPIFFE standard operates under the zero trust security model (STAFFORD, 2020), a cybersecurity paradigm focused on resource protection and the premise that trust is never granted implicitly but must be continually evaluated. Under this model, every entity must undergo strict identity verification and authorization before accessing resources, whether a user, device, or application. This ensures trust is established explicitly and securely, regardless of the entity's location or network boundaries. While SPIFFE manages the lifecycle of the identities it emits, it does not engage directly with the identities it provides, leaving implementation for authentication and access control to the respective services. Importantly, SPIFFE does not address use cases involving human identities, focusing solely on automated identification for machines and services.

A service is a distinct piece of software deployed with a specific configuration to fulfill a particular purpose. This can involve multiple instances performing the same function, such as a cluster of servers handling web requests or a background worker program processing tasks from a queue. In some cases, a service might represent a collection of interconnected systems working together, like a web application that depends on a database backend for storing and retrieving data. In the context of SPIFFE, it is useful to further narrow the definition by considering workloads at a more granular level. Workloads can refer to individual processes running on a node, especially in container orchestration platforms, in which multiple isolated workloads share the same infrastructure (FELDMAN et al., 2020). This granular view is significant because it aligns with how SPIFFE operates, focusing on assigning unique identities to each workload, no matter how small, ensuring that even the smallest components within a system are securely identifiable, supporting robust security practices in scenarios where workloads are frequently spun up, moved, or terminated.

Summarizing, SPIFFE creates its standard of workload identification followed by a cryptographically verifiable document employed with mutual authentication between peers. It happens with three major elements: standardization of an identity namespace, imposes how an issued identity may be presented and verified, and specifies an API through which identity may be retrieved and/or issued by peers. These components are known as **(I)** the SPIFFE ID; **(II)** the SVID; and **(III)** the Workload API (FELDMAN et al., 2020):

(I) **SPIFFE ID:** is a string represented under the Uniform Resource Identifier (URI) format (BERNERS-LEE; FIELDING; MASINTER, 2005) that uniquely and specifically identifies a workload (SPIFFE, 2024b). This represents the core component that starts the foundation of the SPIFFE framework. It comprises a "trust domain name" followed by a workload identifier. The trust domain name is the authority component, identifying the system in which a given identity is issued. The basic scheme of a SPIFFE ID is as follows: ***spiffe://trust domain/workload identifier***. The trust domain corresponds to a system's trust root and could represent an individual, organization, environment, or department running its own independent SPIFFE infrastructure.

Each trust domain maintains its own cryptographic keys, which serve as the "root of trust" for the domain, forming the cryptographic basis for verifying all identities issued within that domain. These keys are distributed in a standardized format called a "SPIFFE bundle", which allows other systems and trust domains to validate identities and establish trust relationships across domain boundaries. This isolation model is crucial for maintaining security, as the compromise of one trust domain does not affect others, enabling organizations to securely manage identities across different environments or organizational units with independent security boundaries.

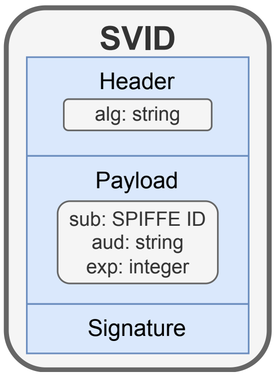
It is highly recommended that trust domains be kept distinct for different environments, such as staging and production, or for different organizational units, to better manage security policies, limit the blast radius of potential security incidents, and reduce risks associated with key compromise. For interoperability considerations, SPIFFE states that implementations must support SPIFFE IDs up to 2048 bytes in length, while it should not generate SPIFFE IDs of length greater than 2048 bytes (SPIFFE, 2024b).

(II) **SVID:** is defined as the mechanism by which a workload proves its identity to a resource or caller, much like a document (e.g., *Carteira de Identidade Nacional*) that carries the identity of its presenter (such as a *Cadastro de Pessoa Física* in Brazil or a Social Security Number in the United States). It must be resistant to forgery and linked to the individual or entity presenting it. To achieve this, an SVID incorporates cryptographic properties that verify its authenticity and confirm that it belongs to the claimed presenter. An SVID is considered valid if an authority has signed it within a SPIFFE trust domain, ensuring that only trusted entities within the domain can issue or validate these documents. SVIDs define the essential properties needed for identity verification and specify how the identity information can be encoded and validated across different existing document formats rather than being a specific document itself (SPIFFE, 2024b).

SVIDs support two main document types for identity representation: X.509 certificates (BOEYEN et al., 2008) and JWT tokens (JONES; BRADLEY; SAKIMURA, 2015b). Regardless of the format, issuing SVIDs implies an attestation process, ensuring that only previously authorized workloads can acquire valid SVIDs through designated API calls. Workloads use these identity documents for mutual authentication, such as establishing an authenticated mutual Transport Layer Security (mTLS) connection with X.509 SVIDs or exchanging signed JWT SVIDs within secure communication channels (SPIFFE, 2024b). Given the focus of this work, we will exclusively examine the JWT SVIDs, leaving the details and use cases related to X.509 certificates outside the scope of this discussion.

Each SVID carries at least a single SPIFFE ID declared in the *subject* claim and a valid signature. The objective of the SVID is to represent the identity of the service presenting it but also has space for an optional public key in the payload section. The SPIFFE ID and the public key (if present) must be included in a portion of the signed payload. If a public key is included, then the corresponding private key is retained by the entity to which the SVID has been issued, and is used to prove ownership of the SVID itself (SPIFFE, 2024b). For a JWT SVID, Figure 2 illustrates the schema containing the mandatory claims:

Figure 2 – JWT SVID mandatory claims scheme.



Source: The author.

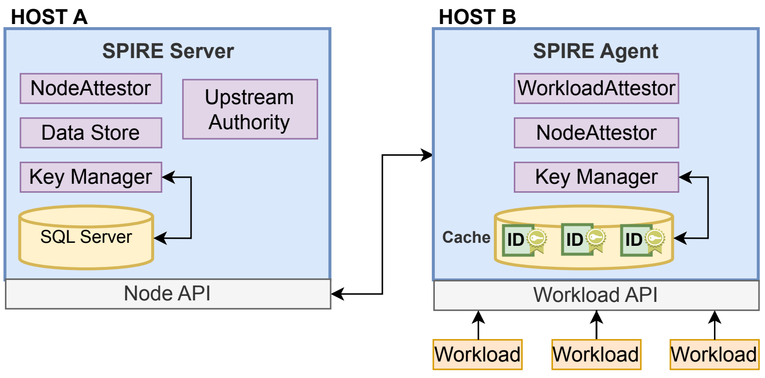
JWT SVIDs are JSON Web Signature (JWS) data structures utilizing JWS compact serialization, described as Uniform Resource Locator (URI) safe strings (JONES; BRADLEY; SAKIMURA, 2015a). They follow the standard JWT token with a handful of restrictions applied regarding the claim specifications (JONES; BRADLEY; SAKIMURA, 2015b). The key claims define important identity attributes and are essential for ensuring secure communication and proper validation. Each claim serves a specific purpose, helping to verify the workload's identity and control how the token is used within the system. *Alg*, or "algorithm" claim, defines which types of algorithms are allowed. It follows all algorithms in (JONES, 2015), specifically sections 3.3, 3.4, and 3.5. The *sub* claim, or "subject", must be the SPIFFE ID of the workload to which it is issued, and here it is the primary claim against which workload identity is asserted. *Aud*, or "audience" claim, ensures that only certain authorized services or systems are intended to read or process the identity information carried by the SVID. Lastly, *exp*, or "expiration time" claim, indicates the exact time when the token becomes invalid.

(III) **Workload API**: provides information and services that enable workloads to leverage SPIFFE identities and SPIFFE-based authentication systems. It is served by the SPIFFE Workload Endpoint, and comprises a number of services in the format of profiles. The JWT SVID profile provides a set of gRPC Remote Procedure Call (gRPC) methods which workloads can use to retrieve JWT SVIDs and their related trust bundles (SPIFFE, 2024b). In order to minimize exposure from a key being leaked or compromised, all private keys are short-lived, rotated frequently, and automatically requested upon the original SPIRE Server issuer.

In the SPIFFE framework, each workload is assigned a unique SPIFFE ID embedded within an SVID to represent the workload's identity securely. SPIRE, as an implementation of SPIFFE, ensures that these identities are properly managed, verified, and distributed. To accomplish secure identity management, SPIRE relies on a coordinated architecture that integrates identity issuance, verification, and distribution at various levels. By enforcing SPIFFE standards, SPIRE enables workloads to dynamically obtain and renew their SVIDs, allowing for continuous authentication within complex, multi-cloud, or containerized environments. The high-level view of this architecture contains two main components: the SPIRE Server and the SPIRE Agent. The Server acts as a signing authority for identities, handles registration entries, performs workload attestation through various plugins, and ensures secure distribution of SVIDs to SPIRE Agents (FELDMAN et al., 2020). It also provides APIs, illustrated in Figure 3, for managing identities and supports integration with external systems for scalable and secure identity management.

The Figure 3 begins with the *SPIRE Server* starting up. If no *UpstreamAuthority* plugin is configured by the user, the *server* generates a self-signed certificate, which is signed using its own private key. This certificate is used to sign SVIDs for all

Figure 3 – High level view of the SPIRE architecture.



Adapted from (FELDMAN et al., 2020).

workloads within the *server* 's trust domain. During its first startup, the *server* automatically creates a trust bundle and stores its contents in an SQL datastore specified in the configuration.

Next, the *SPIRE Agent* starts up on the node where the workload is running, initiating node attestation to verify the identity of its host node to the server. Following this, the *server* performs additional attestation steps to gather more information about the node, updating the node's registration entries and issues an SVID to the *agent*, representing the *agent* 's identity. Using its SVID as a Transport Layer Security (TLS) client certificate, the *agent* contacts the *server* to retrieve the registration entries it is authorized for. The *server* verifies the *agent*'s identity using the provided SVID and completes an mTLS handshake, while the *agent* authenticates the *server* using the bootstrap bundle. Lastly, when a workload requests an SVID via the Workload API, the *agent* starts the workload attestation process by calling its workload attestors and providing the process ID of the workload. These attestors use a combination of kernel and user-space calls to collect additional details about the workload. The collected information, known as workload selectors, is returned to the *agent*. By comparing the discovered selectors with the registered entries, the *agent* determines the workload's identity and returns the appropriate SVID from its cache.

Having established the foundational concepts of ICAM frameworks and the SPIFFE/SPIRE identity control plane for workloads, it becomes clear that while these systems provide robust mechanisms for workload authentication and authorization, they operate within distinct scopes. SPIFFE establishes specifications for distributed workloads to send and receive messages while authenticating identity and asserting message integrity, reasoning about workload identity and authentication in distributed systems. However, the authentication context is currently limited to the workload's identity directly sending or receiving a message, lacking methods and procedures to work with end-users. This limitation presents challenges in scenarios where both user and workload identities must be considered together, particularly in modern cloud-native applications where user requests trigger workload operations across distributed microservices architectures. Understanding this gap between user-centric identity management (typically handled by external Identity Providers (IdPs) using protocols like OAuth 2.0 and OpenID Connect) and workload-centric identity control planes (like SPIFFE) is essential for appreciating the motivation behind extending SPIFFE's capabilities to encompass richer authentication contexts that bridge these two domains.

2.2 MOTIVATION AND PROBLEM DEFINITION

The emergence of cloud-native architectures and microservice-based systems has fundamentally transformed the landscape of distributed computing, introducing unprecedented challenges in identity management and secure inter-service communication. Identity Management (IDM) systems facilitate the creation, verification, and revocation of credentials and the attributes associated with the identity they represent, and for this reason, they are commonly employed in federated environments composed of different administrative domains that attribute different degrees of trust to each other (MALER; REED, 2008). Traditional identity frameworks, originally designed for monolithic applications and relatively static infrastructure, struggle to address the dynamic nature of containerized workloads that are continuously created, migrated, and terminated across heterogeneous computing environments. Taking advantage of the prevalence of federated models, new solutions based on similar credential concepts and formats have been emerging in federated cloud environments. In particular, there has been growing interest in investigating identity management systems suitable for dealing with the high dynamism of virtualization architectures based on computing containers.

The identity control plane solution that is of particular interest in this project is SPIFFE (SPIFFE, 2025b). As discussed in Section 2.1, SPIFFE's solution is particularly interesting for its use in cloud computing environments, in which multiple workloads must be mutually authenticated before establishing a secure connection and exchanging services. A distinctive feature of SPIFFE is that it only issues credentials for workloads that have been attested (e.g., by agents positioned near the compute nodes on which these workloads are executed), guaranteeing the workload's provenance and integrity. This approach reduces the need to pre-issue long-term static credentials for processing workloads, thus avoiding the risk of security incidents due to the exposure of such credentials to capture by attackers.

A core component of SPIFFE is the SVID, a short-lived identity document that cryptographically binds a workload to a unique identifier inside a trusted domain (SPIFFE, 2024b). Even though SPIFFE provides security and flexibility in cloud environments, the context covered by SVIDs is limited to the identity of the workload directly sending or receiving a message. In several applications, however, a richer authentication context is desirable. This limitation, while maintaining architectural simplicity, restricts SPIFFE's applicability in scenarios where workloads must act on behalf of authenticated users or where fine-grained authorization decisions require knowledge of the complete request chain. Recognizing these constraints, the SPIFFE-IdT research project was initiated to systematically investigate extensions to the SPIFFE framework that could address these gaps without compromising its foundational security guarantees.

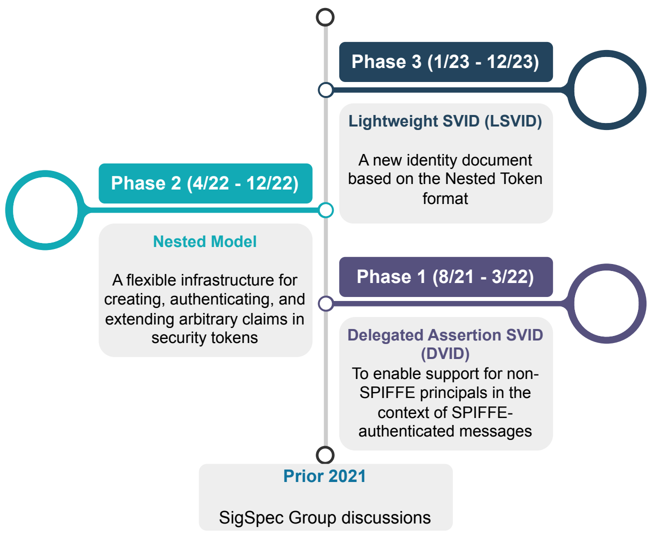
This section contextualizes the present research within this broader initiative by examining two complementary perspectives. Subsection 2.2.1 provides a comprehensive overview of the SPIFFE-IdT project, documenting its three-year evolution through distinct research phases that progressively expanded SPIFFE's capabilities. It details the collaborative engagement with the SPIFFE community, the creation of the "SPIFFE - Assertions and Tokens Workgroup", and the development of Delegated Assertion SVID (DVID) for user identity delegation, the Nested Token model enabling extensible authentication contexts, and the Lightweight SVID (LSVID) as a unified identity document format. This historical account establishes the technical foundation upon which this thesis builds, demonstrating both the significant achievements of the project and the methodological approach employed throughout its execution.

Subsection 2.2.2 then transitions from accomplishments to challenges, explicitly identifying the critical gaps that emerged from the project's implementation strategy. While each research phase successfully demonstrated technical feasibility through independent PoCs, this approach resulted in fragmented codebases with substantial duplication, inconsistent deployment procedures that hinder reproducibility, and a complete absence of evaluation under container orchestration—despite SPIFFE's inherent design for cloud-native environments. These shortcomings, though understandable within the exploratory context of the original project, represent significant barriers to practical adoption and rigorous comparative analysis. This subsection articulates these problems explicitly and formulates the specific research objectives that guide this thesis: consolidating the disparate implementations into a unified, maintainable architecture and systematically evaluating their operational feasibility and performance characteristics under Kubernetes orchestration. Together, these subsections establish both the substantial groundwork provided by the SPIFFE-IdT project and the precise technical challenges that remain unresolved, thereby justifying the research direction undertaken in this work.

2.2.1 Motivation: The SPIFFE-IdT Project

The research project entitled **Gerenciamento seguro de identidades federadas: aprimorando e estendendo a arquitetura SPIFFE** (SPIFFE-IdT), conducted in collaboration with Universidade do Estado de Santa Catarina (UDESC), Universidade de São Paulo (USP), Instituto Federal Catarinense (IFC), and Hewlett Packard Enterprise (HPE), is an exploratory study aimed at identifying opportunities to enhance SPIFFE / SPIRE processes and technologies from multiple perspectives, particularly in terms of security, performance, and functionality. The origins of the project are tied to the efforts of the Special Interest Group SPIFFE Specification (SIG-Spec), an initiative within the SPIFFE community focused on advancing the framework's capabilities and expanding its technical horizons (SPIFFE, 2024a). Meetings and discussions eventually led to its official start in 2021. Spanning three years (2021–2023), the project involved the participants listed in Table 1 and was structured into annual phases. Each year focused on addressing specific challenges and use cases while proposing innovative solutions to enhance the framework's applicability and functionality. Key milestones of the project are illustrated in Figure 4, highlighting its initiation in August 2021, the annual phases of work, and its conclusion in December 2023.

Figure 4 – SPIFFE-IdT Project timeline.


Phase 3 (1/23 - 12/23)
Lightweight SVID (LSVID)
A new identity document based on the Nested Token format
Phase 2 (4/22 - 12/22)
Nested Model
A flexible infrastructure for creating, authenticating, and extending arbitrary claims in security tokens
Phase 1 (8/21 - 3/22)
Delegated Assertion SVID (DVID)
To enable support for non-SPIFFE principals in the context of SPIFFE-authenticated messages
Prior 2021
SigSpec Group discussions

Source: The author.

Phase 1 (Subsection 2.2.1.1) main use case is the concept of authenticating requests based on end-user identity. This work designed a new security document named DVID, which produced two key outcomes for the author: **(I)** the author's undergraduate thesis titled "**Análise de desempenho do Secure Production Identity Framework for Everyone (SPIFFE)**" at UDESC; and **(II)** the published paper "**DVID: Adding Delegated Authentication to SPIFFE Trusted Domains**" (JESSUP et al.,

2024) presented at the Advanced Information Networking and Applications (AINA) conference.

Phase 2 (Subsection 2.2.1.2) builds upon this foundational research by addressing use cases that fall outside the original SPIFFE scope. This phase focuses on enhancing security measures by investigating a novel token model that incorporates a nesting concept called *Nested Token*. The findings from this investigation resulted in the published paper **"Enhancing SPIFFE/SPIRE Environment with a Nested Security Token Model."** (COCHAK. et al., 2024) at the International Conference on Cloud Computing and Services Science (CLOSER).

Phase 3 (Subsection 2.2.1.3) converges all previously obtained results to create a new security document named LSVID, which is utilized in a modified version of SPIRE. This work resulted in the published paper **"Lightweight SPIFFE Verifiable Identity Document (LSVID): A Nested Token Approach for Enhanced Security and Flexibility in SPIFFE"** at the International Conference on Cloud Computing Technology and Science (CLOUDCOM) (COCHAK et al., 2024).

Table 1 – Project list participants by year.

Participants	**Category / Title**	**Participation Date**
Charles Christian Miers	Researcher	2021, 2022, 2023
Gabriel Dias Tambelli	Undergraduate Student	2022, 2023
Henrique Zanela Cochak ***	Undergraduate Student / Master Student	2021, 2022, 2023
Lucas Rodrigues Cupertino Cardoso	Undergraduate Student	2022, 2023
Luís Henrique de Almeida Fernandes	Undergraduate Student	2021
Marco Antonio Marques *	Doctorate Student	2021, 2022, 2023
Marco Antonio Torrez Rojas	Researcher	2021, 2022, 2023
Marcos Antonio Simplicio Junior **	Project Coordinator / Researcher	2021, 2022, 2023
Milton Pedro Pagliuso Neto ***	Undergraduate Student / Master Student	2022, 2023
Pedro Henrique Barcha Correia	Master Student	2022, 2023

Source: The author.

Note: * The project was conducted as part of Marco Antonio Marques's doctorate thesis.
** Marcos Antonio Simplicio Junior served as both the project coordinator and researcher of the project.
*** Students who began as undergraduates and continued their academic journey into a Master's program within the project.

A central focus of this research is to evaluate mechanisms that enable the identification of users making requests to workloads operating within its environment. In this context, the project envisions that these identities, along with their associated attributes, can be delegated (e.g., via a token) across different stages of request processing, introducing the concept of transitive identity. Transitive identities are authentications from trusted domains, in which these domains can again provide access to other domains recursively (nowadays chosen by the industry as Federated Identity (OKTA, 2024)).

Each participant in the project (Table 1) had key objectives to study or jobs specified to carry out. My function in it was to code, debug, and most of all, discuss, create, and generate a group of benchmarks of each phase, which again, resulted in the published papers ((JESSUP et al., 2024), (COCHAK. et al., 2024) and (COCHAK et al., 2024)). The baselines provided aim to generate a dataset that will serve as a comparative line for developers aiming to implement the SPIFFE standard in their applications, as well as for future applications that extend or adapt the project. As detailed in Section 2.1, SPIFFE establishes specifications for distributed workloads to authenticate identity and assert message integrity in distributed systems, but the authentication context is currently limited to the workload's identity directly sending or receiving a message, lacking methods to work with end-users. The SPIFFE-IdT project builds upon the foundational concepts of SPIFFE, addressing its limitations concerning user-centric scenarios and expanding its scope beyond workload authentication. The first result of this effort is the DVID.

2.2.1.1 SPIFFE IdT Project - Phase I - DVID

The goal of DVID is to enable support for non-SPIFFE principals in the context of SPIFFE authenticated messages. The proposed framework allows a SPIFFE workload W, called *Subject Workload*, to obtain a DVID in exchange for a credential issued by an IdP related to an end user U. The resulting JWT based document, issued by an *Asserting Workload*, validates the end user token and binds together W's and U's identities, asserting that W is entitled to act on behalf of U for a given period.

Inside a SPIFFE trust domain, the Asserting Workload may be considered a TTP. However, when dealing with multiple security domains, the framework includes mechanisms to prove token validity without revealing its contents using a Zero Knowledge Proof (ZKP) (GOLDWASSER; MICALI; RACKOFF, 1989). This Rivest-Shamir-Adleman (RSA)-based ZKP prevents token replay attacks and unauthorized privilege escalation across trust domain boundaries.

The PoC benchmarks evaluated two key operations: minting (generating a DVID) and validation (verifying its integrity and authenticity), both with and without optional RSA-ZKP proofs. Performance metrics included CPU consumption, memory consumption, and execution time, measured using Docker containers with constrained resources (1 CPU core, 128MB RAM) and collected via Prometheus at 50-millisecond intervals. The parameters are detailed in Table 2.

Table 2 – Phase 1 parameters.

Parameter	**Description**
CPU Consumption	Percentage of utilized CPU resources
Execution Time	Time taken to complete key operations
Memory Consumption	Percentage of utilized RAM resource

Source: (JESSUP et al., 2024).

The baseline measurements demonstrated that RSA-ZKP introduces significant computational overhead for both minting and validation operations, with minimal impact on CPU and memory consumption but substantial payload size increases that could affect network performance in high-throughput environments. Detailed results are presented in (JESSUP et al., 2024), providing a comparative framework for developers implementing DVID and highlighting opportunities for optimization through alternative cryptographic approaches.

2.2.1.2 SPIFFE IdT Project - Phase II - Nested Token Model

Building upon the DVID research, the Nested Token model was developed to support diverse identity approaches and token extension capabilities, enabling delegation, attenuation, and token sealing for more adaptable and fine-grained access control. The recursive construction allows any workload to create or extend an existing token to include new signed claims, enabling use cases where information integrity and tracking are desirable while detecting potential tampering along the chain.

The solution supports two signature schemes: **(I)** *ID-Mode*, which leverages an existing IdP where workloads obtain valid identity documents associated with their signature keys, allowing non-repudiable token creation and extension using their private keys; and **(II)** *Anonymous Mode*, which uses an identity-based signature scheme with Schnorr signature concatenation, where each workload extracts the aggregation key from the previous signature to sign the new token, avoiding dependencies on external authentication entities and reducing token size by maintaining only partial signatures for all tokens except the last.

Benchmarks evaluated the same metrics as Phase 1, with the addition of *Token Size Growth* to assess how token size varies with each concatenation (Table 3).

Table 3 – Phase 2 parameters.

Parameter	**Description**
CPU Consumption	Percentage of utilized CPU resources
Execution Time	Time taken to complete key operations
Token Size Growth	Size in bytes of each concatenated token
Memory Consumption	Percentage of utilized RAM resource

Source: (COCHAK. et al., 2024).

Results demonstrated that Anonymous Mode achieved smaller token sizes due to its signature concatenation mechanism, while ID-Mode exhibited lower validation times at the cost of higher memory consumption from certificate storage and validation. Comprehensive analysis and detailed results are presented in (COCHAK. et al., 2024).

2.2.1.3 SPIFFE IdT Project - Phase III - LSVID

The final phase focuses on developing the LSVID, an identity document building upon the nested token model to enhance the SPIFFE framework's existing SVID.

The LSVID extends the nested token architecture, creating an identity document that can be used as an extensible token for fine-grained access control mechanisms. Implementation required significant modifications to the SPIRE architecture, including new endpoints for minting and validating the document, marking the first official integration of this type within the framework.

Phase 3 centers on the *ID-Mode*, where SPIRE functions as a TTP, issuing identity documents with valid signatures. In this mode, workloads obtain identity documents associated with their signature keys, enabling non-repudiable token creation and extension. The LSVID structure comprises payload, signature, and nested components, following a recursive model where tokens can be extended by appending new signed claims. This enables use cases including delegation, attenuation, and path tracing while maintaining cryptographic verification throughout the chain.

Benchmark evaluation focused on performance metrics detailed in Table 4, selected to validate design goals through empirical data.

Table 4 – Phase 3 parameters.

Parameter	**Description**
Execution Time	Time taken to complete key operations
Token Size Growth	Size in bytes of each concatenated token

Source: (COCHAK et al., 2024).

Results demonstrated that LSVID operations remain efficient for authentication systems requiring low latency. Token size growth follows a predictable pattern with each extension, influenced by the signature scheme and payload content. The document format using JavaScript Object Notation (JSON) and base64 encoding provides flexibility while maintaining compact representation compared to alternatives requiring multiple independent tokens. Comprehensive analysis and detailed results are presented in (COCHAK et al., 2024).

2.2.2 Problem Definition

The SPIFFE-IdT project successfully demonstrated that SPIFFE's scope can be extended beyond workload-to-workload authentication to encompass richer authentication contexts, including user identity delegation, extensible token chains, and fine-grained authorization mechanisms. Each implementation proved technically feasible through independent PoCs, delivering published research contributions and establishing baseline performance metrics. However, this exploratory development approach, while appropriate for validating novel cryptographic constructions and security properties, left questions about how the security documents would behave in environments closer to actual deployment scenarios.

Each PoC was developed as a complete, standalone microservice application comprising four architectural tiers: Subject-WL (entry point receiving external OAuth tokens and front-end), Asserting-WL (security document minting and validation service), one or more M-Tier components (acting as intermediary hops), and Target-WL (backend resource server such as a database). While this modular architecture facilitated independent development during the project timeline, it resulted in four separate codebases implementing fundamentally similar functionality with significant code duplication. Common components, including workload communication protocols, token validation workflows, benchmark instrumentation, and Docker containerization configurations, were reimplemented independently for each security document type. This fragmentation complicates testing and maintenance: each PoC runs as an independent Docker Compose deployment with its own configuration, making it difficult to ensure consistent behavior across implementations despite their structural similarities.

The independent PoCs lack standardized deployment procedures, environmental configurations, and dependency management strategies. Each implementation employs slightly different Docker Compose configurations, resource allocation policies, and network topologies, making it challenging for external researchers to reproduce experimental results accurately. Furthermore, deployment documentation varies in completeness across the four implementations, creating barriers to validation by the broader research community. This inconsistency undermines one of the project's stated objectives: establishing baseline performance metrics that can serve as comparative references for developers implementing SPIFFE extensions. Without reproducible deployment procedures and consistent environmental conditions, empirical comparisons between security document types become unreliable.

All baseline measurements were conducted using Docker Compose for bare container deployment, which provides direct container-to-container networking without the service mesh abstractions, dynamic scheduling, and resource management policies characteristic of production Kubernetes environments. This represents a significant gap in understanding real world deployment feasibility. Container orchestrators introduce additional layers of abstraction, including service discovery mechanisms, load balancing, health checks, and network policies that may affect performance characteristics in ways not captured by bare-container benchmarks. Moreover, orchestrated environments enable horizontal scaling and dynamic workload distribution, capabilities that remain unexplored in the current PoC implementations.

This thesis addresses these identified gaps through two complementary research objectives. First, the architectural consolidation objective unifies the four independent PoCs into a single, maintainable prototype system that eliminates code duplication through shared service abstractions and modular security document handlers while preserving the original functionality and cryptographic properties of all four implementations. The unified system establishes standardized deployment procedures, environmental configurations, and dependency management to ensure reproducibility, and implements consistent benchmark instrumentation across all security document modes, enabling rigorous comparative analysis under identical infrastructure conditions.

Second, the orchestrated deployment evaluation objective investigates the deployment of the unified prototype within Kubernetes environments, evaluating operational feasibility and performance impact. The operational feasibility investigation examines the extent to which SPIFFE-based security mechanisms integrate with container orchestration abstractions (service discovery, configuration management), and identifies what architectural or configurational adjustments are required for reliable operation across all security document modes. The performance impact analysis measures the computational overhead introduced by orchestration layers, examining how these abstractions affect critical performance metrics including latency, system throughput, and resource consumption patterns. A comprehensive benchmarking methodology systematically measures performance across both bare-container and Kubernetes-orchestrated deployments, providing empirical evidence of the trade-offs inherent in adopting container orchestration for SPIFFE-based identity management systems.

2.3 RELATED WORKS

The deployment of security microservices in orchestrated environments requires understanding both the performance characteristics of orchestration platforms themselves and the additional overhead introduced by security mechanisms integrated into the service communication path. While extensive research exists on container orchestration performance comparison, a significant gap remains in understanding how **security document validation systems**, particularly cryptographic identity, behave when deployed under the abstractions and constraints of Kubernetes orchestration. This section reviews the existing literature through two complementary lenses: (1) orchestration platform performance characteristics that establish baseline expectations for deployment overhead, and (2) the operational implications of deploying security enhanced microservices in orchestrated environments.

2.3.1 Search Methodology

To conduct a comprehensive review of container orchestration performance research, a systematic search approach was employed using Google Scholar as the primary search engine. The search was configured to span publications from 2018 to 2025, maintaining consistency with the timeframe established in previous work and ensuring coverage of the period following SPIFFE's acceptance into the CNCF in March 2018 (SPIFFE, 2025a). Two pre-configuration settings were applied: a custom date range filter and the exclusion of citation only entries. The following search query was formulated to identify relevant studies on container orchestration performance comparison:

```
("container orchestration" AND "performance") AND
("comparison" OR "evaluation" OR "benchmark") AND
(
    ("k3s" AND "kubernetes") OR
    ("microk8s" AND "kubernetes") OR
    ("docker" AND "kubernetes") OR
    ("k3s" AND "microk8s")
) AND
("monitoring" OR "workload" OR "resource")
```

This query was designed to capture studies that evaluate container orchestration platforms from a performance perspective. The term *“container orchestration"* ensures focus on the deployment and management layer rather than individual container technologies. The inclusion of *“performance"* combined with *“comparison"*, *“evaluation"*, or *“benchmark"* targets studies that provide quantitative assessments rather than purely architectural descriptions. The platform-specific terms focus the search on comparative studies involving Kubernetes and its lightweight distributions (K3s, MicroK8s) as well as Docker-based orchestration. Finally, the terms *“monitoring"*, *“workload"*, and *“resource"* ensure the inclusion of studies that measure observable system behavior through instrumentation and metrics collection.

The search results were then filtered according to inclusion and exclusion criteria to identify the most relevant studies for this research, as presented in Table 5.

Table 5 – Inclusion and exclusion criteria for related works.

Inclusion criteria	**Exclusion criteria**
Scholarly literature such as articles, papers, and theses	Books, slides, or project proposals
Documents written in English	Documents written before 2018
Studies comparing container orchestration platforms	Studies focusing solely on container runtime performance
Performance evaluation with quantitative metrics	Purely theoretical or architectural discussions

Source: The author.

After applying these criteria, several key studies were identified that provide relevant context for understanding container orchestration performance evaluation. Table 6 presents a comprehensive overview of these studies, highlighting their approach to platform comparison, security considerations, dependency handling, and workload characteristics.

Table 6 – Comprehensive comparison of container orchestration performance and security studies.

Study	**Security Focus**	**Platform Comparison**	**External Deps.**	**Monitoring Approach**	**Real Workloads**
(KOZIOLEK; ESKANDANI, 2023)	None	MicroK8s vs K3s vs K0s vs MicroShift	None	k-bench + netdata	No
(AQASIZADE; ATAIE; BASTAM, 2025)	None	Kubeadm vs K3s vs MicroK8s vs K0s	Xen, Docker/-Containerd	IOzone + Sysbench + K6	Yes (MySQL, OpenFaaS)
(BÖHM; WIRTZ, 2021)	None	Kubernetes vs MicroK8s vs K3s	None	CPU, memory, disk monitoring	Yes (nginx)
(YAKUBOV; HÄSTBACKA, 2025a)	None	K0s vs K3s vs KubeEdge vs OpenYurt vs K8s	None	CPU, memory, disk, throughput	Mixed
(YAKUBOV; HÄSTBACKA, 2025b)	**Security compliance**	K0s vs K3s vs KubeEdge vs OpenYurt vs K8s	Network outage sim.	kube-bench security assessment	Yes
(RAMADAN et al., 2025)	None	K3s vs K3d vs Kind vs MicroK8s vs Minikube vs K8s	None	kube-burner stress testing	No
(KOUKIS et al., 2024)	None	CNI plugins on K3s, K0s, MicroK8s	Multiple CNI	Network throughput, latency	Yes
(ČILIć et al., 2023)	None	K8s vs K3s vs KubeEdge vs ioFog	None	Startup/migration times	Yes (edge)
(FAYOS-JORDAN et al., 2020)	None	Docker Swarm vs Kubernetes	None	SBC monitoring	Yes (IoT)
(BRAUN; HOFFMANN; MÖRSEBURG, 2019)	None	Microservices vs. Monolithic	None	Response time, throughput	Yes (Web ARS)
(DIN et al., 2022)	Network security	NDN-enabled vehicular nodes	Yes (Edge and Cloud)	Bandwidth and latency	Yes (IoV microservices)
(BARLETTA et al., 2025)	None	Priority-based orchestration	Yes (5G Networks)	Orchestration latency	Yes (Network functions)
(PEDNEKAR et al., 2024)	None	Kubernetes vs OpenShift	Yes (Different hardware)	Workload performance	Yes (Containerized apps)
(PAPADOPOULOS, 2025)	Yes (API security)	K3s vs K0s vs MicroK8s vs K8s	None	Resource consumption, performance	Yes (Edge computing)
(ASCENSÃO et al., 2024)	Yes (Security compliance)	K8s vs K3s vs K0s	None	Performance and security metrics	Yes
(KJORVEZIROSKI; FILIPOSKA, 2022)	None	K8s vs K3s vs MicroK8s	Yes (OpenFaaS)	Response time, startup times	Yes (Serverless functions)
(TELENYK et al., 2021)	None	K8s vs MicroK8s vs K3s	None	Resource utilization, startup speed	Yes
SPIFFE-IdT Project Contributions					
(JESSUP et al., 2024)	**SPIFFE ext.**	No (Docker Compose)	Yes (Okta)	Prometheus	Yes
(COCHAK. et al., 2024)	**SPIFFE ext.**	No (Docker Compose)	Yes (Okta)	Prometheus	Yes
(COCHAK et al., 2024)	**SPIFFE ext.**	No (Docker Compose)	Yes (Okta)	Host-level	Yes
This work	**SPIFFE unified**	**Docker vs K8s**	**Yes (Google)**	**Prometheus**	**Yes**

Source: The author.

(KOZIOLEK; ESKANDANI, 2023) conducted one of the most comprehensive analyses of lightweight Kubernetes distributions, comparing MicroK8s, K3s, K0s, and MicroShift. Their measurements revealed that K3s and K0s achieved the highest control plane throughput, with pod creation latencies ranging from 200ms to 470ms depending on the distribution. These baseline latency measurements are critical for understanding orchestration overhead, particularly for security systems where identity validation occurs on every service interaction. (BÖHM; WIRTZ, 2021) complemented this work by profiling platforms across complete lifecycle events, demonstrating that MicroK8s exhibited higher resource utilization and longer latencies for cluster operations that become critical when considering certificate rotation and workload attestation cycles in SPIFFE-based systems.

(BRAUN; HOFFMANN; MÖRSEBURG, 2019) compared microservice and monolithic architectures for web-based audience response systems, providing important baseline performance differences between architectural patterns. (TELENYK et al., 2021) analyzed performance metrics for orchestration actions in Kubernetes, MicroK8s, and K3s, finding that standard Kubernetes outperformed lightweight alternatives in most tests while K3s demonstrated better disk utilization. (PEDNEKAR et al., 2024) conducted a comparative analysis of Kubernetes and OpenShift based on workloads using different hardware architectures, offering insights into platform selection based on infrastructure constraints.

(AQASIZADE; ATAIE; BASTAM, 2025) investigated the impact of underlying infrastructure by comparing virtualization modes and container runtimes, finding that Docker outperformed Containerd in both disk and CPU intensive workloads. This is particularly relevant for security document systems that perform frequent cryptographic operations and certificate storage operations. (YAKUBOV; HÄSTBACKA, 2025a) extended the analysis to edge computing scenarios, demonstrating that K3s exhibited the lowest resource consumption while K0s and standard Kubernetes excelled in data plane throughput. (PAPADOPOULOS, 2025) further validated these findings by conducting a comparative analysis of K3s, K0s, MicroK8s and K8s in edge computing environments, emphasizing K3s and MicroK8s suitability for resourceconstrained deployments.

(ČILIć et al., 2023) compared four orchestration tools (K8S, K3s, KubeEdge, and ioFog) for edge computing environments, measuring container startup times, service migration latencies, and memory consumption. Their results showed that Kubernetes achieved the best startup performance, while ioFog demonstrated significantly higher startup times of over 34 seconds. (FAYOS-JORDAN et al., 2020) conducted a direct performance comparison between Docker Swarm and Kubernetes on single-board computers for fog computing scenarios, concluding that Docker Swarm outperformed Kubernetes in resource-constrained environments. (KJORVEZIROSKI; FILIPOSKA, 2022) evaluated serverless computing performance using three different Kubernetes distributions (Kubernetes, K3s, and MicroK8s) with the OpenFaaS platform, providing valuable insights into edge deployment patterns for security-critical services.

(BARLETTA et al., 2025) investigated priority-based orchestration for 5G network functions, measuring orchestration latency for real-world workloads with stringent timing requirements. (KOUKIS et al., 2024) evaluated Container Network Interface (CNI) plugins across lightweight distributions, revealing that plugin deployment does not necessarily improve resource utilization. (RAMADAN et al., 2025) advanced benchmarking methodology using kube-burner for stress testing. Din et al. (DIN et al., 2022) implemented a microservices in-network computing framework for Information-Centric IoVs, focusing on reducing latency and bandwidth consumption.

From a security perspective, only a small fraction of studies explicitly addressed security concerns. (YAKUBOV; HÄSTBACKA, 2025b) evaluated security compliance using kube-bench and resilience through network outage simulation, revealing that lightweight distributions (K3s, K0s) offer superior performance but lower security compliance compared to standard Kubernetes and specialized edge distributions. (ASCENSÃO et al., 2024) reinforced this finding by demonstrating that K0s achieved best performance but exhibited security vulnerabilities comparable to standard Kubernetes. These findings are significant for SPIFFE deployments, which implement application-layer security, raising the question of whether platform-level security deficiencies are adequately compensated by robust identity management.

SPIFFE-Based Artifact: The SPIFFE-IdT project has contributions (JESSUP et al., 2024; COCHAK. et al., 2024; COCHAK et al., 2024) investigating SPIFFE capabilities for enhanced authentication and authorization scenarios. These studies, which are direct results of the SPIFFE-IdT project, explored SPIFFE extensions in containerized environments using Docker Compose deployments exclusively. The investigations focused on understanding the behavior and performance characteristics of novel SPIFFE security artifacts, including delegated authentication documents, nested token models, and lightweight encoding formats. Each project study conducted benchmark measurements to evaluate computational overhead, network latency, and resource consumption patterns introduced by these security enhancements during credential generation, transmission, and verification processes.

However, these previous SPIFFE-IdT project investigations shared a common limitation: all experiments were conducted exclusively on Docker Compose without comparative analysis across orchestration technologies. The primary objective of these project results was to validate the feasibility of the proposed SPIFFE extensions and quantify their performance impact in isolation, demonstrating that enhanced security mechanisms could operate within acceptable performance bounds. The SPIFFE-IdT studies did not explore how different container orchestration platforms affect the performance of these security-focused workloads, leaving a critical gap in understanding orchestration-induced overhead on cryptographic identity validation systems.

2.4 CHAPTER CONSIDERATIONS

This chapter established the foundational concepts of cloud computing, microservices architecture, and the fundamentals of the SPIFFE ecosystem. Cloud computing evolution has led to the widespread adoption of microservices architecture, which introduces significant security challenges by distributing application logic across multiple components, broadening the attack surface and increasing complexity in managing identity and access control. To address these challenges, ICAM frameworks provide structured approaches for managing digital identities across distributed systems. Within this context, SPIFFE emerges as a specialized IMS solution designed specifically for workloads in dynamic cloud environments. The SPIFFE framework creates a standardized workload identification system through three core components: SPIFFE ID (a URI-formatted unique identifier), SVID (a cryptographically verifiable identity document), and the Workload API (for retrieving and validating credentials).

The SPIFFE-IdT project, initiated in 2021 as a collaboration between UDESC, USP, IFC, and HPE, systematically extended SPIFFE's capabilities through three annual phases. Phase 1 introduced DVID to enable user identity delegation, allowing workloads to act on behalf of authenticated users. Phase 2 developed the Nested Token model to support more flexible identity approaches with delegation and attenuation capabilities. Phase 3 culminated in LSVID, a unified identity document format that integrated the advances from previous phases into a modified version of SPIRE. These extensions addresses SPIFFE's fundamental limitation, its restriction to workload-to-workload authentication without support for end-user identity contexts. Through these innovations, the project bridges the gap between workload-centric identity control planes and user-centric identity management systems.

The SPIFFE-IdT project successfully demonstrated that SPIFFE's scope can be extended beyond workload-to-workload authentication to encompass richer authentication contexts, including user identity delegation, extensible token chains, and fine-grained authorization mechanisms. While each implementation proved technically feasible through independent PoCs deployments, the exploratory development approach resulted in fragmented codebases with significant code duplication, with a lack of evaluation under container orchestration environments, despite SPIFFE's inherent design for cloud-native applications. By unifying the independent PoCs implementations into a single architecture, this work eliminated code duplication, standardized deployment procedures, and enabled a consistent benchmarking across all security document types, preserving the distinct cryptographic properties of each implementation while creating a foundation for a rigorous and easier comparative analysis under identical infrastructure conditions.

Through empirical measurement across both Docker Compose and Kubernetes deployments, this research has quantified the orchestration overhead for security document operations from the project. The findings provide evidence-based insights into the trade-offs involved when deploying SPIFFE-based IMSs in orchestrated environments. These measurements establish a new basis for understanding how the abstractions, introduced by Kubernetes with Minikube, may affect metrics such as latency, and resource consumption.

3 PROPOSAL

This thesis proposes to investigate the operational feasibility and performance characteristics of SPIFFE-based artifacts when deployed in container orchestrated environments. Specifically, this research examines how the security artifacts developed under the **SPIFFE-IdT project**, originally implemented as four independent PoCs, behave when consolidated into a unified prototype and subjected to the abstractions and overhead inherent in Kubernetes orchestration.

The central research questions driving this proposal are:

- **Operational Feasibility:** To what extent does the unified prototype integrate with container orchestrators, and what architectural or configurational adjustments are required to ensure reliable operation across all security document modes within orchestrated environments?
- **Performance Impact of Orchestration Abstraction:** What measurable computational overhead does container orchestration introduce, and how do orchestration-layer abstractions affect critical performance metrics including scalability, resource utilization, inter-component latency, and overall system throughput?

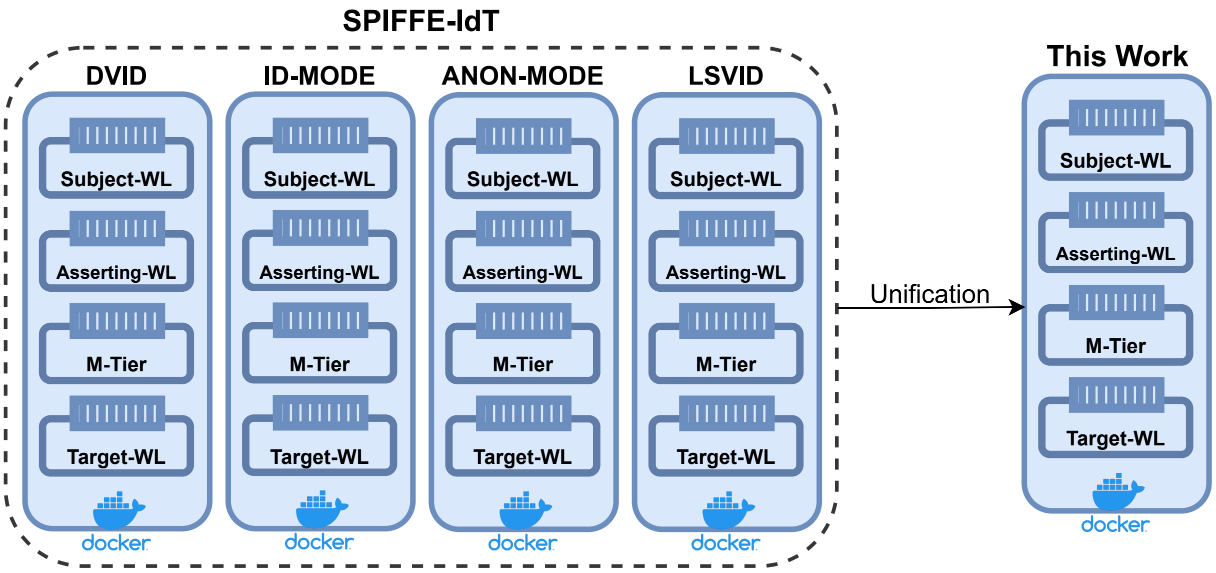
Answering these questions requires two preliminary contributions: first, the architectural consolidation of the four independent PoCs into a single coherent prototype; and second, the deployment of this unified system within both baseline (Docker Compose) and orchestrated (Kubernetes) environments to enable systematic comparative analysis.

3.1 UNIFICATION OF SPIFFE-IDT MODELS

This master's thesis represents a continuation of the research conducted under the project **Gerenciamento seguro de identidades federadas: aprimorando e estendendo a arquitetura SPIFFE (SPIFFE-IdT)**, discussed in Subsection 2.2.1, though with a fundamentally distinct research objective. Upon the advancements achieved throughout the project's three year duration, this work establishes a new direction focused on consolidation, systematic evaluation, and real-world deployment feasibility of the developed security artifacts.

One of the core contributions enabling this investigation is the architectural unification of four independent PoCs into a single, coherent prototype system. As depicted in Figure 5, each original PoC implemented a complete microservice architecture consisting of four or more tiers: Subject-WL, Asserting-WL (security token service), one or more M-Tier components (intermediaries), and Target-WL (backend resource server). In this architecture, Subject-WL acts as the entry point, receiving external OAuth tokens from third-party identity providers. Asserting-WL serves as the internal identity authority, minting all SPIFFE identity artifacts and participating in validation procedures according to the specific security document mode. The M-Tier component(s) perform mode-specific validation logic and forward authorized requests to Target-WL, which executes operations after conducting final authorization checks.

Figure 5 – Architectural Unification: Consolidation of Four SPIFFE-IdT PoCs into a Single Integrated Prototype.



Source: The author.

While this modular architecture facilitated independent development and validation of distinct security document types, it resulted in significant code duplication, inconsistent deployment procedures, and fragmented maintenance efforts across four separate implementations. The unification process synthesizes these disparate implementations into an integrated architecture that preserves the functional capabilities of all four security document modes while eliminating redundancy and establishing a standardized operational framework. This consolidation yields several critical advantages:

- **Enhanced Reproducibility:** A unified codebase with standardized deployment procedures substantially reduces the complexity of reproducing experimental results, lowering the barrier for validation by independent researchers and facilitating knowledge transfer.
- **Streamlined Maintenance and Benchmarking:** Consolidating four separate implementations into a single, well-structured system significantly simplifies long-term maintenance, ensuring that improvements and security patches can be ap-

plied consistently across all security document types. Moreover, the unified architecture facilitates systematic collection of benchmark data, as performance metrics can be gathered under identical environmental conditions and using consistent instrumentation across all modes, thereby enhancing the reliability and comparability of empirical measurements.

- **Systematic Comparative Analysis:** The unified architecture establishes a controlled experimental environment where different security document modes operate under identical infrastructure conditions, enabling rigorous performance comparisons and eliminating confounding variables introduced by implementation inconsistencies.

Throughout the consolidation process, deliberate methodological restraint was exercised to preserve the original design intent and core functionality of each PoC. Modifications were introduced only where necessary to achieve integration, explicitly avoiding the Ship of Theseus paradox wherein excessive alteration could fundamentally compromise the authenticity and validity of the original artifacts (BARKER, 2019). This approach ensures that benchmark results remain comparable to prior evaluations conducted on the individual PoCs. Beyond consolidation, recognizing that SPIFFE is inherently designed for cloud-native architectures, this research extends the unified prototype to investigate deployment within container-orchestrated environments, introducing an additional analytical dimension examining the computational and operational costs imposed by Kubernetes orchestration abstractions.

3.2 RELEVANT SCENARIOS

The transition of SPIFFE based identity management systems from development environments to production grade deployments necessitates understanding their behavior under container orchestration. While the SPIFFE-IdT project successfully enhanced SPIFFE capabilities through multiple security document implementations, culminating in LSVID's integration into the SPIRE framework, the operational characteristics of these enhancements within orchestrated environments remain unexplored.

Container orchestration platforms, particularly Kubernetes, have become the de facto standard for deploying cloud native microservices in production environments. However, orchestration introduces multiple abstraction layers (e.g., service discovery mechanisms, network proxies, scheduler overhead, resource management policies) that fundamentally alter the performance profile of deployed applications. For security critical systems like SPIFFE, where identity validation and cryptographic operations occur on the critical path of every service to service interaction, understanding the quantitative impact of these orchestration abstractions is essential for informed deployment decisions.

This thesis addresses this gap by systematically benchmarking the unified SPIFFE-IdT prototype across two deployment scenarios: a baseline Docker Compose environment that minimizes orchestration overhead, and a Kubernetes orchestrated environment introducing realistic production grade abstractions. The comparative analysis focuses on quantifying:

- **Latency overhead** introduced by Kubernetes networking layers (DNS resolution, service mesh, iptables/netfilter routing) compared to direct container to container communication. This measurement includes round trip times for authentication requests, network path analysis, and potential bottlenecks in the identity verification workflow.
- **Resource utilization patterns** under orchestration, including CPU and memory consumption attributable to Kubernetes components versus application workloads. This encompasses both steady state operation and peak utilization during high frequency identity operations.
- **Authentication and authorization delay** specifically for identity minting and validation operations in Kubernetes compared to the baseline environment. This analysis examines the entire lifecycle of security document processing, from initial creation through validation and verification.

By establishing empirical baselines for these metrics, this research provides actionable guidance for practitioners considering production deployment of SPIFFE based identity systems, while contributing quantitative data to inform future optimizations of both SPIFFE implementations and Kubernetes networking architectures.

3.3 TESTBED

The experimental evaluation deploys the unified SPIFFE-IdT prototype across two testbed configurations to systematically compare baseline container performance against orchestration-induced overhead. Both testbeds implement the same microservice architecture derived from the consolidation process, preserving the core workload components and communication patterns established during the SPIFFE-IdT project.

3.3.1 Workload Architecture

The deployed system consists of four primary workload types that collectively implement the SPIFFE-based federated identity flow. Although the communication patterns between these workloads were established through prior research (JESSUP et al., 2024; COCHAK. et al., 2024; COCHAK et al., 2024), the consolidation and orchestration deployment may introduce architectural adjustments while preserving core functionality. The fundamental components are:

1. **Subject Workload**: Serves as the application front-end, initiating authentication requests and interacting with external OAuth identity providers (e.g., Okta, Google). This workload receives external OAuth tokens and forwards them to the Asserting Workload for validation and SPIFFE identity issuance.
2. **Asserting Workload (Local IdP)**: Functions as the internal identity authority responsible for issuing and validating SPIFFE security documents. Upon receiving validated OAuth credentials from the Subject Workload, it mints SPIFFE-IdT security artifacts and participates in subsequent validation procedures according to the configured security document mode.
3. **Middle Tier**: One or more intermediary workloads that enforce authorization policies and perform mode-specific validation logic. These components introduce additional communication hops between the front-end and the database data retrieval
4. **Target Workload**: The backend resource server that executes protected operations (e.g., database queries) after all security validations are completed. This workload represents the ultimate destination for authenticated and authorized requests.

While specific communication flows will vary depending on the selected security document, later on explained, the presence of these four workload types remains consistent across all experimental scenarios. Each workload exposes both functional endpoints and observability endpoints (Prometheus metrics) to enable performance monitoring during evaluation.

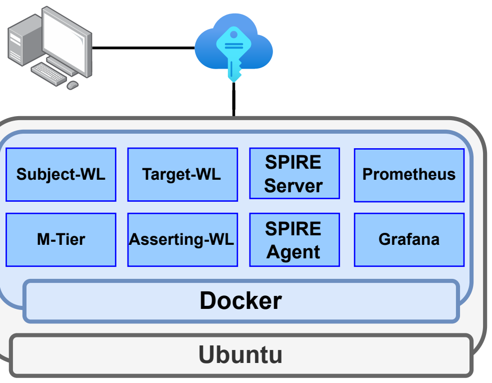
3.3.2 Hardware Specifications

All experiments were conducted on a bare metal server with the following specifications: GNU/Linux Ubuntu 24.04.2 LTS, 500 GiB RAM, and dual Intel Xeon E5-2690 v2 processors operating at 3.00 GHz (20 physical cores across 2 sockets, 40 threads with hyper-threading). This configuration provides more than sufficient computational resources to eliminate hardware bottlenecks and isolate orchestration platform overhead as the primary performance variable.

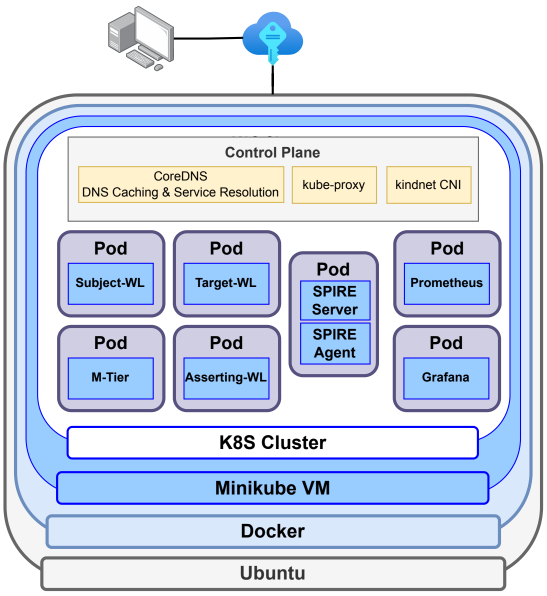
3.3.3 Testbed Configurations

Two distinct testbed configurations were deployed on the same physical hardware to ensure hardware-independent comparison, as illustrated in Figure 6. The architectural differences between these configurations directly impact service discovery and inter-container communication patterns.

Figure 6 – Testbeds design.



(a) Docker Compose Testbed.



(b) Kubernetes Testbed.

Source: The author.

The Docker Compose testbed, depicted in Figure 6a, deploys services directly on the host operating system using Docker's native container runtime. All workload containers (Subject-WL, Asserting-WL, Middle-Tier, and Target-WL) run alongside SPIRE infrastructure components within the same Docker environment. This configuration represents a baseline deployment scenario with minimal orchestration overhead.

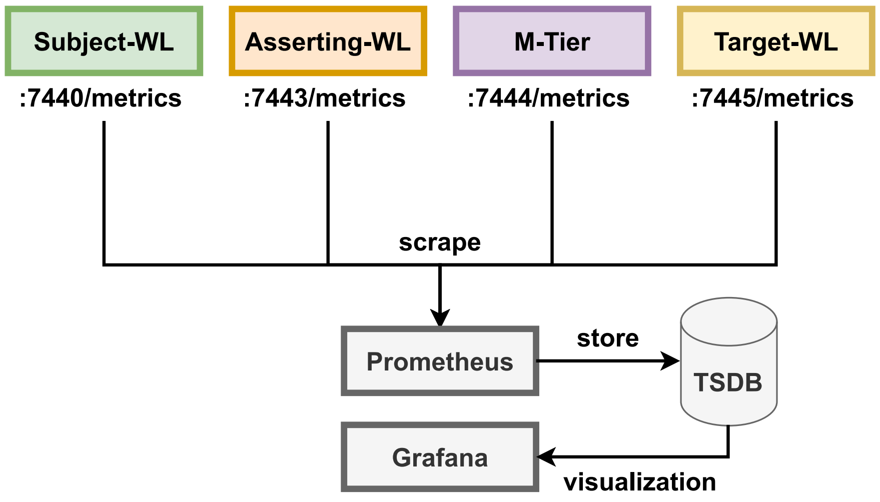
In contrast, the Kubernetes testbed shown in Figure 6b consists of a single-node Minikube cluster with 10 CPU cores allocated, running within a Docker container on the same bare metal host. The SPIFFE components (SPIRE Server and SPIRE Agent) are deployed alongside application workloads (Subject-WL, Asserting-WL, Middle-Tier, and Target-WL), with each component running as an independent pod. Each pod is allocated a single dedicated CPU resource to ensure performance isolation and eliminate scheduling interference.

Workloads access SPIFFE identities through the Workload API, exposed via a Unix domain socket at */run/spire/sockets/agent.sock* (SPIFFE, 2025c). This socket is written to the host filesystem by the SPIRE Agent and mounted into workload pods via

Kubernetes *hostPath* volumes (Kubernetes, 2025). Unix domain socket communication occurs through the host node's filesystem and constitutes Inter-Process Communication (IPC) rather than network-based communication (Kubernetes CSI, 2024).

Both testbed configurations incorporate a comprehensive observability infrastructure based on Prometheus and Grafana, as illustrated in Figure 7. This monitoring stack enables systematic performance analysis across all workload components and facilitates comparative evaluation of different security document modes under controlled conditions.

Figure 7 – Prometheus metrics collection and Grafana visualization.



Source: The author.

Prometheus (PROMETHEUS, 2025) serves as the metrics collection and storage backend, leveraging its pull-based scraping model that is particularly suited for MSAs. Each workload component in the unified architecture exposes dedicated metrics endpoints, and serve metrics in Prometheus's standardized exposition format via HTTP */metrics* paths. The Prometheus server scrapes application workload endpoints every 100 ms to capture fine-grained performance data during benchmark execution. This high-frequency scraping enables precise latency measurements and throughput analysis without requiring instrumentation changes to the application code. The collected time-series data is stored in Prometheus's local Time Series Database (TSDB), with the pull-based architecture decoupling metrics collection from application logic to ensure minimal monitoring overhead that does not interfere with benchmark measurements.

Grafana (LABS, 2025) provides the visualization layer, querying Prometheus's TSDB using PromQL to construct real-time dashboards displaying performance indicators. These visualizations enable both real-time monitoring during benchmark execution and post-hoc analysis, supporting the comparative performance evaluation presented in subsequent chapters. The observability infrastructure operates independently of application workloads across both Docker Compose and Kubernetes environments, ensuring metrics collection does not introduce measurement bias while providing comprehensive visibility into the behavior of all security document modes under evaluation.

Beyond the instrumentation and deployment infrastructure described above, careful attention was given to external dependencies that could introduce measurement variability. A notable difference between this evaluation and prior SPIFFE-IdT assessments (JESSUP et al., 2024; COCHAK. et al., 2024; COCHAK et al., 2024) concerns the external OAuth IdP configuration. Previous evaluations utilized Okta as the OAuth provider, as specified by the SPIFFE-IdT project requirements. However, Okta's infrastructure is geographically distant from our testbed location, with OAuth validation requests likely routed to servers in the US-East region, resulting in round-trip times of $100\,\mathrm{ms}$ or more for token validation operations.

For this study, we transitioned to Google OAuth (*https://accounts.google.com*) as the identity provider to eliminate this source of network latency variability. Network analysis via IPv6 address resolution (*2800:3f0:4001:814::200a*) and GeoIP lookup confirms JSON Web Key Sets (JWKS) requests are routed to Google's southamerica-east1 datacenter in São Paulo, Brazil, significantly reducing geographic distance compared to the previous Okta configuration. This geographically proximate identity provider eliminates intercontinental routing variability, ensuring that measured latencies reflect actual cryptographic and orchestration overhead rather than network propagation delays. Consequently, our measurements focus exclusively on the performance characteristics inherent to each security document mode and the orchestration-induced overhead introduced by the deployment environment, without confounding factors from geographically distant external services.

3.4 CHAPTER CONSIDERATIONS

This chapter established the methodological foundation for evaluating SPIFFE-IdT security artifacts within container-orchestrated environments. The architectural unification of four independent PoCs into a single coherent prototype enables systematic comparative analysis across security document modes under identical infrastructure conditions, eliminating implementation inconsistencies that could confound performance measurements. By deploying this unified system across both Docker Compose and Kubernetes testbeds on identical hardware, the research framework isolates orchestration-induced overhead as the primary performance variable while maintaining functional equivalence with prior SPIFFE-IdT evaluations.

The testbed infrastructure provides observability through Prometheus metrics collection and Grafana visualization, enabling performance analysis across all workload components. The transition from geographically distant OAuth providers (Okta in US-East) to geographically proximate identity services (Google in São Paulo) ensures that measured latencies reflect cryptographic and orchestration overhead rather than network propagation delays. This controlled experimental environment establishes the necessary conditions to answer the research questions concerning operational feasibility and performance impact of deploying SPIFFE-based artifacts within container orchestration platforms.

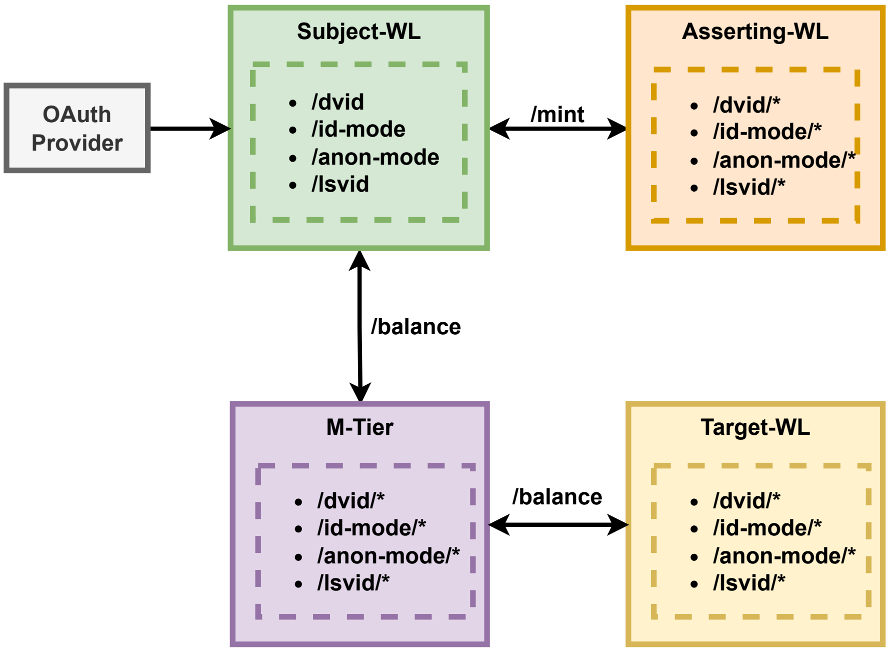
4 EXPERIMENTS AND DATA ANALYSIS

Before presenting the experimental method and empirical results, it is essential to describe the technical implementation of the architectural unification discussed in Chapter 3. While Chapter 3 established the motivation and benefits of consolidating the four independent PoCs, this chapter begins by detailing how this consolidation was achieved through a router-based architecture that preserves the functional capabilities of all security document modes while eliminating code duplication.

4.1 IMPLEMENTATION OF THE UNIFIED ARCHITECTURE

The consolidation of the four SPIFFE-IdT PoCs into a single coherent prototype required a fundamental redesign of the request dispatching mechanism. In the original implementations, each security document mode operated as a completely independent deployment with dedicated Subject-WL, Asserting-WL, M-Tier, and Target-WL components. The unified architecture, illustrated in Figure 8, transforms this fragmented deployment model into an integrated system through a two-tier routing structure: a configuration-driven dispatcher at the Subject-WL and a subrouter-based handler registry at the Asserting-WL.

Figure 8 – Unified PoC architecture with router based mode.



Source: The author.

4.1.1 Mode Selection at Subject-WL

The Subject-WL serves as the user-facing entry point for authentication flows. The unified web interface provides four mode-selection buttons. When a user clicks a button, the browser issues a request to a mode-specific path and the function handler inspects the request path and queries a configuration map to retrieve the mode-specific configuration. The configuration map associates each URL path with its corresponding Asserting-WL endpoint (e.g., */dvid/mint*, */id-mode/ecdsa-assertion*, */anon-mode/schnorr-assertion*, */lsvid/extendlsvid*) and an internal mode identifier. This declarative configuration approach improves maintainability by eliminating conditional branching throughout the codebase.

Once the configuration is retrieved, the handler invokes the appropriate mode-specific request function. Each function establishes an mTLS connection to the Asserting-WL and forwards the user's OAuth token to the mode-specific endpoint. This architecture isolates mode-specific logic within dedicated functions while enabling all modes to share common infrastructure for mTLS establishment, error handling, and response processing. The handler also manages session state by storing the selected mode in both server-side session storage and a client-side cookie, ensuring consistent routing across subsequent user interactions.

4.1.2 Middle-Tier Subrouter Architecture

The Asserting-WL is responsible for minting and validating security documents, and must expose distinct endpoints for each security document mode while operating as a unified deployment. The unified architecture implements this through a hierarchical routing structure that enables path prefix isolation between modes. The main router initializes four mode-specific subrouters, each associated with a distinct path prefix corresponding to one of the security document modes: */dvid*, */id-mode*, */anon-mode*, and */lsvid*. Within each subrouter, mode-specific handlers are registered for the operations supported by that security document type.

When the Asserting-WL receives a request, the main router examines the URL path prefix to determine which subrouter should handle it. This hierarchical routing structure ensures complete isolation between security document modes: each mode's handlers execute independently and have no visibility into the state or behavior of other modes, preserving functional equivalence with the original independent PoCs. The same subrouter-based routing logic is implemented in both the M-Tier and Target-WL components, ensuring consistent request dispatching throughout the entire multi-tier architecture.

4.1.3 Functional Preservation and Execution Flow

The router-based consolidation strategy preserves the core cryptographic operations, validation procedures, and document formats implemented in the original PoCs. Each mode-specific handler package encapsulates the complete implementation of its corresponding security document, maintaining the fundamental logic established in prior research. While the underlying cryptographic primitives remain unchanged, the unification process introduces additional validation steps and error handling mechanisms to prevent failures, a critical consideration in microservice architectures.

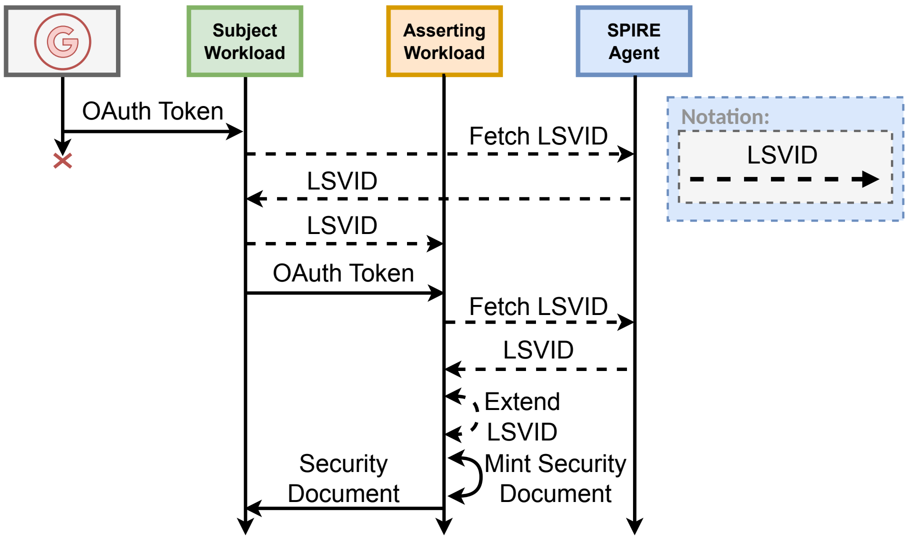
While the unified architecture employs consistent routing mechanisms across all security document modes, the execution flow varies significantly depending on the validation procedures specific to each mode. The system exhibits two distinct execution focal points: the *minting process*, where initial security artifacts are generated and bound to user identities, and the *validation process*, where these documents are extended, verified, and used to authorize access to protected resources across the multi-tier architecture.

For the purposes of this master thesis, we focus on these two main functional operations, minting and validation, explained in detail in subsequent sections. The primary research interest lies in identifying possible discrepancies in CPU and memory consumption of the new unified baseline compared to the original isolated implementations within the Minikube environment. Additionally, this work aims to capture and analyze the performance characteristics of each assertion mode under the unified architecture to establish a comprehensive performance baseline to understand the impact of the usage of orchestration.

4.1.3.1 Minting Process

The minting process is triggered when a user successfully authenticates with the external Google IdP and the authorization callback handler receives the OAuth token. At this stage, all four security document modes follow an identical execution flow: the Subject-WL extracts the OAuth token from the callback, stores it in the application environment, and invokes the unified handler that inspects the user's mode selection. The handler then establishes an mTLS connection to the Asserting-WL and forwards the OAuth token to the mode-specific minting endpoint. As illustrated in Figure 9, for DVID, ID-Mode, and Anon-Mode, the minting flow is straightforward: the Subject-WL sends the OAuth token to the Asserting-WL, which validates the token, mints the corresponding security document, and transmits it back for local storage.

Figure 9 – Minting process flow.



Source: The author.

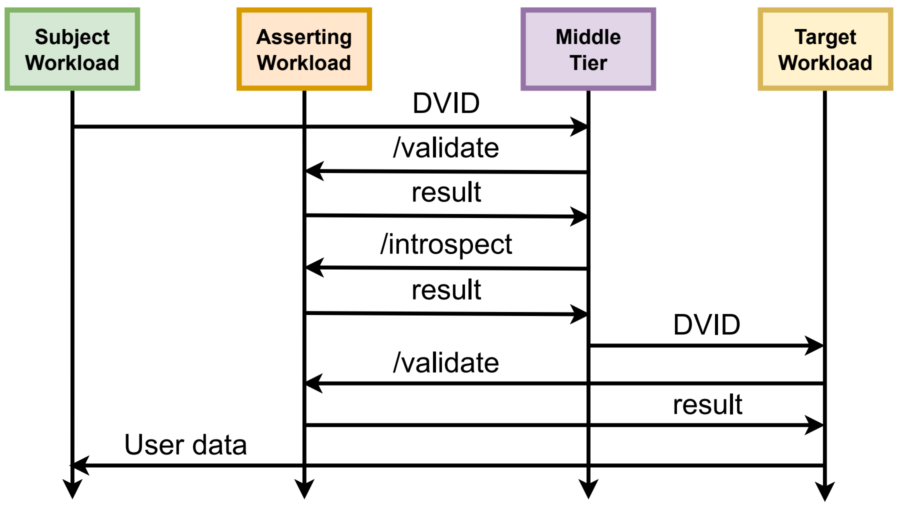
LSVID mode introduces additional operations involving the SPIRE Agent. Before contacting the Asserting-WL, the Subject-WL fetches its existing LSVID from the SPIRE Agent, extends it with a payload targeting the Asserting-WL, and transmits both the OAuth token and the extended LSVID to the Asserting-WL. The Asserting-WL validates the OAuth token, fetches its own LSVID from the SPIRE Agent, and extends the received LSVID by adding OAuth delegation claims, completing the chain of trust that the Subject-WL subsequently stores.

4.1.3.2 Validation Process

The validation process begins when the user initiates an operation requiring access to protected resources, in this implementation, requesting account balance information from the Target-WL. This process traverses the complete architecture (Subject-WL $\rightarrow$ M-Tier $\rightarrow$ Target-WL) and exhibits divergence across security document modes due to their distinct validation procedures and token extension mechanisms, as illustrated in Figures 10, 11, and 12.

In DVID mode (Figure 10), the Subject-WL directly forwards the security document during the minting to the M-Tier without modification. The M-Tier validates the DA-SVID through two sequential operations: first, it contacts the Asserting-WL's validation endpoint (*/dvid/validate*) to verify the signature and expiration; second, it retrieves a ZKP via the introspect endpoint (*/dvid/introspect*) to verify OAuth token validity without the Asserting-WL exposing the token itself. Following successful validation, the M-Tier forwards the unchanged security document to the Target-WL, which performs an additional validation step with the Asserting-WL before executing the database query and returning the result.

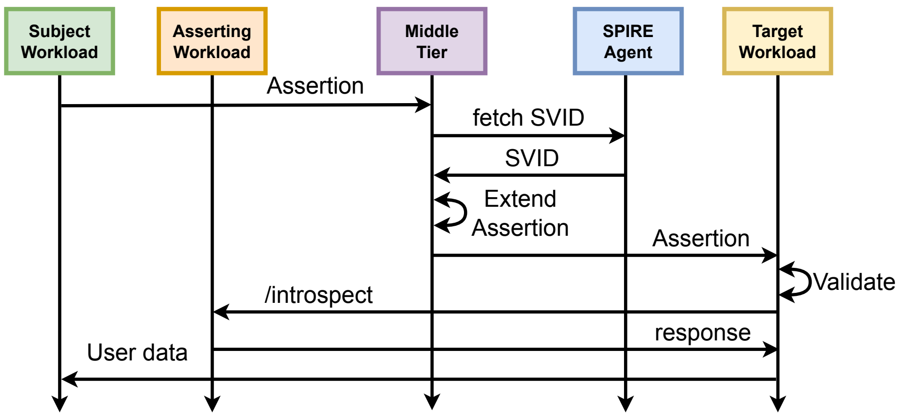
Figure 10 – DVID validation flow.



Source: The author.

ID-Mode and Anon-Mode (Figure 11) share a common structural flow through the architecture. The Subject-WL receives an assertion from the Asserting-WL, extends it, and forwards it to the M-Tier. The M-Tier performs validation, extends the assertion again, and forwards it to the Target-WL. The Target-WL validates the complete assertion chain, optionally calls the Asserting-WL's introspect endpoint for additional verification, and executes the database query. Detailed information about the validation and extension mechanisms employed by each mode are described in the related work (COCHAK. et al., 2024).

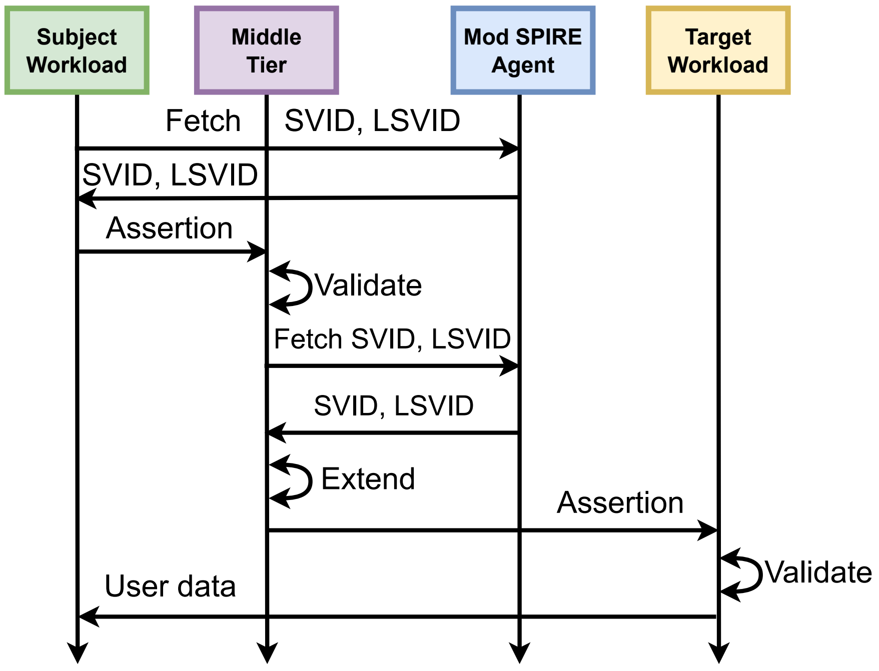
Figure 11 – ID-Mode and Anon-Mode validation flows.



Source: The author.

LSVID mode (Figure 12) implements nested token extension with bearer verification. The Subject-WL fetches both its X.509-SVID and LSVID token from the SPIRE Agent, constructs a payload containing its identity and the user information, extends the LSVID by nesting and signing, and transmits it to the M-Tier. The M-Tier validates the signature chain, performs bearer verification by confirming the mTLS certificate matches the LSVID issuer claim, then fetches its own identities from the SPIRE Agent and extends the LSVID again before forwarding to the Target-WL. The Target-WL validates the complete signature chain, performs bearer verification, and queries the local database without external calls to the Asserting-WL, as validation is entirely self-contained through the nested structure.

Figure 12 – LSVID validation flow.



Source: The author.

4.1.4 Connection Pooling Analysis Across Assertion Modes

Connection pooling constitutes a performance optimization technique that enables the reuse of existing mTLS connections rather than establishing new connections for each request. This approach significantly reduces the computational and network overhead associated with TCP handshakes, TLS certificate exchanges, and cryptographic negotiations (SHARIFIAN et al., 2019).

Establishing a new mTLS connection requires multiple round-trip communications: TCP handshake, TLS negotiation, certificate exchange and validation, SPIRE Agent interaction to fetch current SVIDs, and key agreement. Connection pooling eliminates some of these steps for subsequent requests. The cumulative effect across high-volume workloads can be substantial, particularly for modes like DVID where multiple sequential calls occur per request. As shown in Table 7, the four assertion modes implement connection pooling based on their architectural requirements. In DVID mode, connection pooling is implemented exclusively at the M-Tier, which maintains persistent connection pools to the Asserting-WL for validation and introspect operations, and to the Target-WL for forwarding authenticated requests. Each user request necessitates multiple sequential calls (validation, introspect, Target-WL query), making pooling particularly beneficial as it avoids three complete mTLS handshakes per request. The Target-WL creates fresh connections for each request, prioritizing implementation simplicity over performance optimization.

Table 7 – Connection Pooling Summary.

Mode	M-Tier Pools To	Target-WL Pools To
DVID	Asserting-WL, Target-WL	(none)
ID-Mode	Target-WL	Asserting-WL
Anon-Mode	(none)	Asserting-WL
LSVID	Target-WL	(none)

Source: The author.

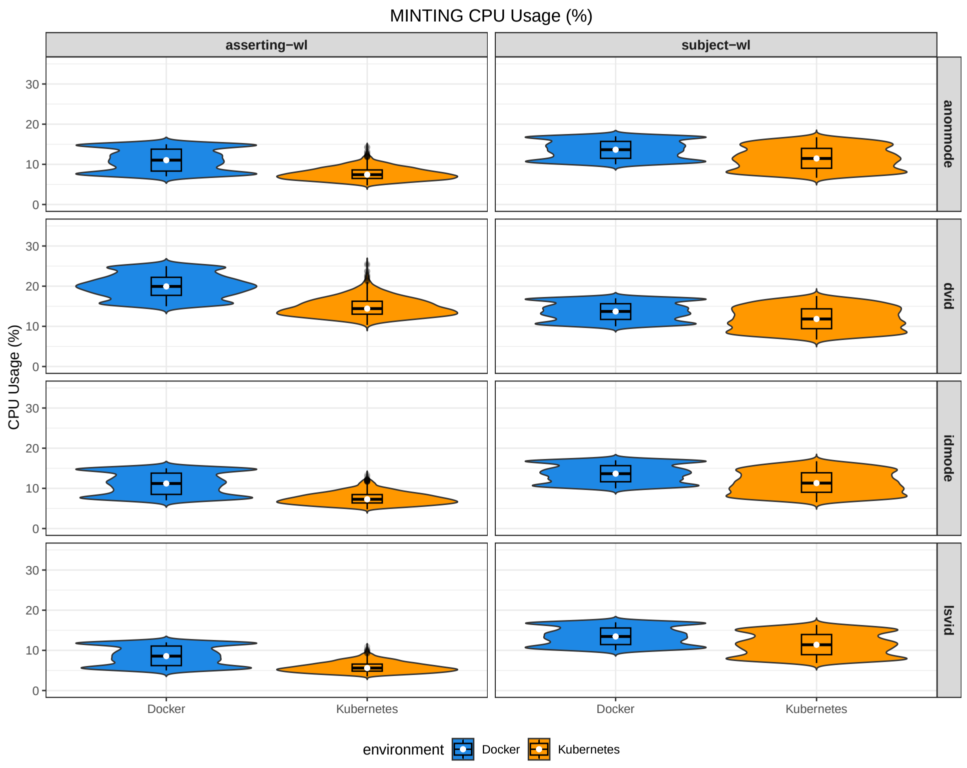
ID-Mode implements connection pooling at both the M-Tier and Target-WL, representing the most comprehensively optimized mode. The M-Tier maintains pooled connections to the Target-WL, while the Target-WL maintains pooled connections to the Asserting-WL for introspect operations when enabled. This bilateral pooling strategy ensures that both the primary forwarding path and the secondary validation path benefit from connection reuse, proving particularly valuable in high-throughput scenarios. LSVID mode implements connection pooling exclusively at the M-Tier for forwarding extended tokens to the Target-WL. Unlike other modes where Target-WL communicates with Asserting-WL, LSVID validation is entirely self-contained through local validation and bearer verification via mTLS certificate inspection, and database queries.

Table 7 summarizes the connection pooling strategies across the four assertion modes, reflecting distinct design philosophies from ID-Mode's comprehensive optimization to Anon-Mode's emphasis on implementation simplicity. Understanding these trade-offs proves essential for selecting the appropriate assertion mode for specific deployment scenarios and performance requirements.

4.2 DATA ANALYSIS AND RESULTS

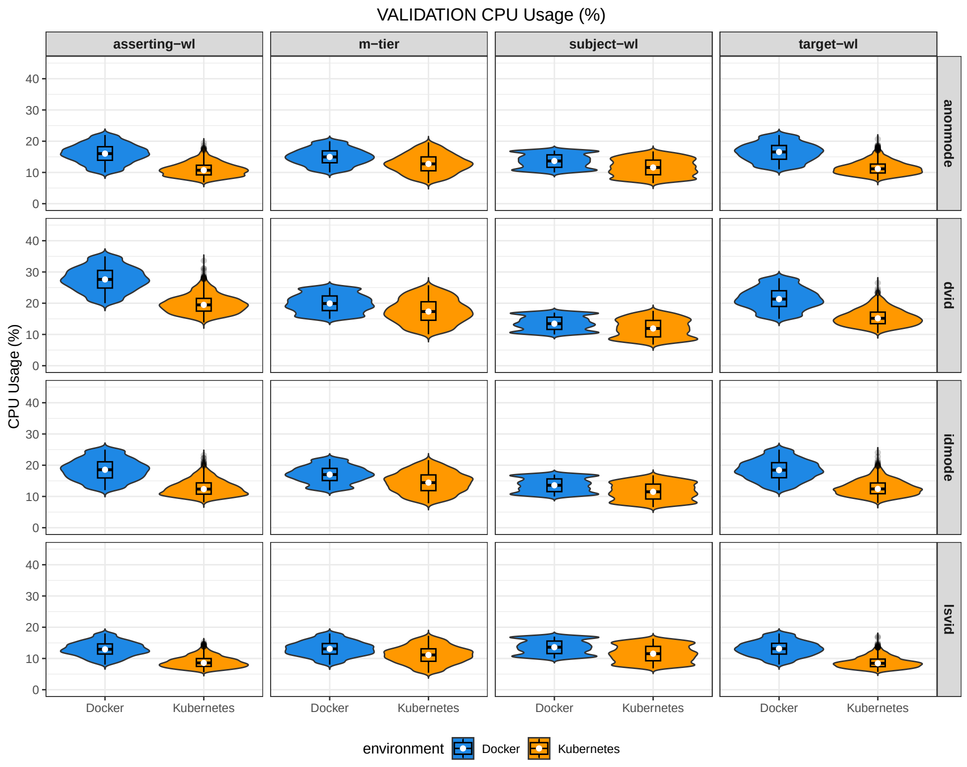
This section presents a comparative analysis of resource consumption metrics between Docker Compose and Kubernetes deployments across the four assertion modes. Comprehensive performance measurements were collected for CPU and memory usage during both minting and validation operations. For MSA, the metrics collected thus far did not matter as expected for the current intent, and the addition of an orchestration layer did not provide any kind of overhead when related to computational resource consumption as initially anticipated. These findings suggest that the observed variations in performance metrics are more likely application-related rather than attributable to the underlying architecture. The CPU utilization data reveals interesting patterns across deployment environments and operational modes. Figures 13 and 14 illustrate the distribution of CPU consumption for minting and validation operations respectively.

Figure 13 – CPU usage comparison for minting operations across assertion modes



Source: The author.

Figure 14 – CPU usage comparison for validation operations across assertion modes



Source: The author.

For minting operations, contrary to initial expectations, Kubernetes deployments consistently showed lower CPU utilization compared to Docker Compose across all assertion modes. This finding aligns with observations from (FERREIRA; SINNOTT, 2019), who reported that managed Kubernetes services often exhibited better performance than manually configured environments (FERREIRA; SINNOTT, 2019). Table 8 shows that for subject workloads, CPU usage remained fairly consistent across modes (13.56-13.69% for Docker vs. 11.43-11.91% for Kubernetes). The asserting workload showed more pronounced differences, with Docker consuming between 8.62-19.99% CPU compared to Kubernetes at 5.78-14.79%. It should be noted that although CPU utilization varies, this variability is partly attributable to the Prometheus scraping interval, which was set to 100ms for this thesis. This interval was chosen because typical operations occur within 10-50ms, making 100ms a reasonable monitoring window. This sampling rate may influence the visual representation of CPU behavior in the collected data.

Table 8 – CPU Usage for Minting Operation - Docker vs Kubernetes (%).

Workload	**CPU Usage (%) - Minting**							
	DVID		**ID-Mode**		**Anon-Mode**		**LSVID**	
	Docker	**K8s**	**Docker**	**K8s**	**Docker**	**K8s**	**Docker**	**K8s**
subject-wl	13.69 ± 2.23	11.91 ± 2.86	13.65 ± 2.24	11.43 ± 2.77	13.64 ± 2.29	11.54 ± 2.81	13.56 ± 2.27	11.46 ± 2.76
asserting-wl	19.99 ± 2.94	14.79 ± 2.28	11.15 ± 2.72	7.50 ± 1.50	11.08 ± 2.76	7.64 ± 1.51	8.62 ± 2.40	5.78 ± 1.24

Source: The author.

Validation operations exhibited a similar pattern, as shown in Table 9, with Kubernetes deployments consistently utilizing 15-35% less CPU resources than their Docker counterparts. Notably, the asserting workload under DVID showed the highest overall CPU consumption (27.67% ± 3.84% for Docker vs. 19.75% ± 2.97% for Kubernetes), which aligns with its more complex validation requirements. The M-tier workload demonstrated the smallest difference between environments (typically 15-20% reduction in Kubernetes), suggesting that its processing tasks are less affected by the deployment environment.

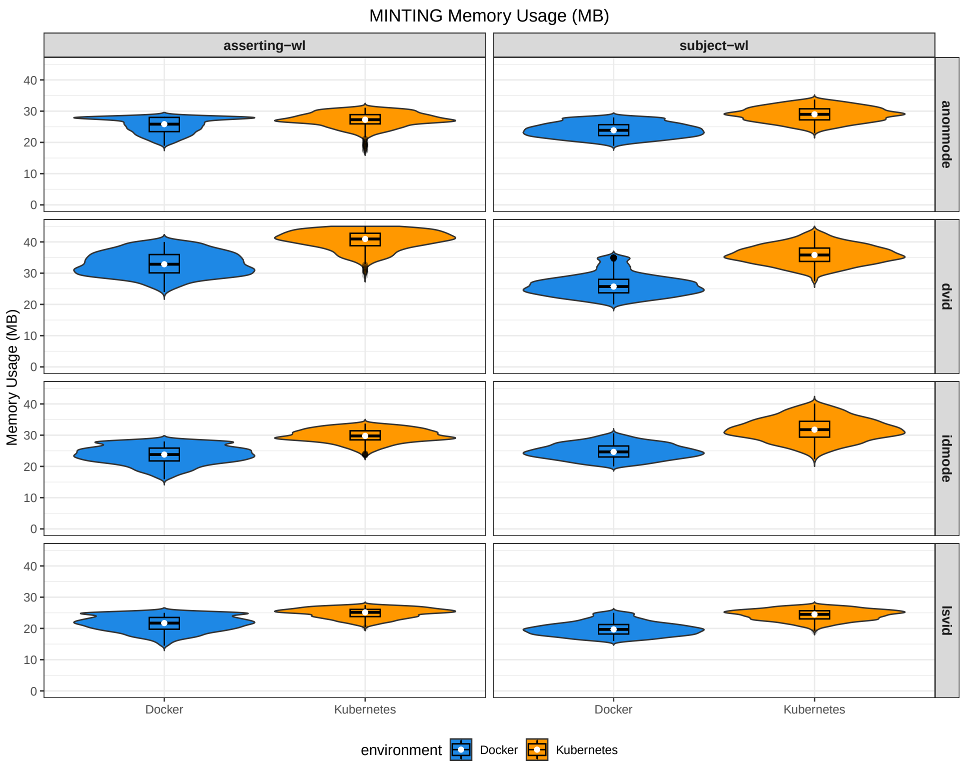
Table 9 – CPU Usage for Validation Operation - Docker vs Kubernetes (%).

Workload	**CPU Usage (%) - Validation**							
	DVID		**ID-Mode**		**Anon-Mode**		**LSVID**	
	Docker	**K8s**	**Docker**	**K8s**	**Docker**	**K8s**	**Docker**	**K8s**
subject-wl	13.58 ± 2.24	11.93 ± 2.92	13.60 ± 2.24	11.55 ± 2.74	13.67 ± 2.24	11.56 ± 2.74	13.62 ± 2.24	11.54 ± 2.69
asserting-wl	27.67 ± 3.84	19.75 ± 2.97	18.57 ± 3.40	12.76 ± 2.54	16.03 ± 3.04	10.93 ± 2.18	13.01 ± 2.34	8.85 ± 1.80
m-tier	19.99 ± 2.94	17.46 ± 3.87	17.01 ± 2.68	14.43 ± 3.34	15.00 ± 2.59	12.80 ± 3.02	13.11 ± 2.38	11.14 ± 2.78
target-wl	21.47 ± 3.47	15.46 ± 2.63	18.49 ± 3.25	12.76 ± 2.48	16.46 ± 2.93	11.45 ± 2.14	13.16 ± 2.42	8.74 ± 1.78

Source: The author.

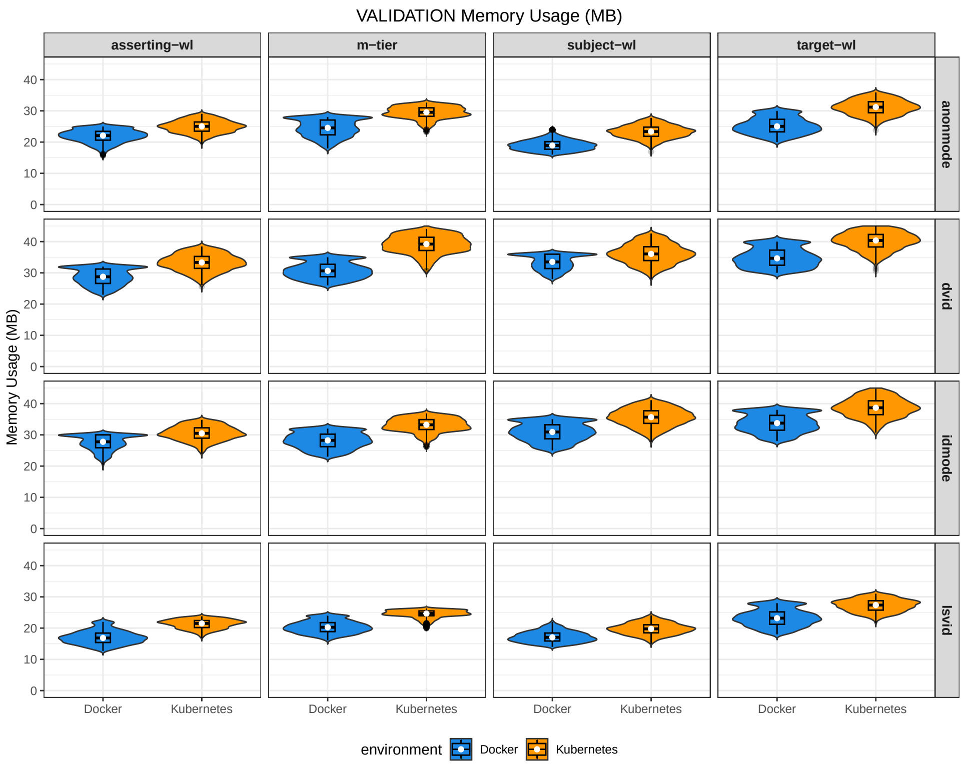
In contrast to CPU metrics, memory consumption patterns revealed an inverse relationship between Docker Compose and Kubernetes. Figures 15 and 16 visualize these differences.

Figure 15 – Memory usage comparison for minting across assertion modes



Source: The author.

Figure 16 – Memory usage comparison for validation across assertion modes



Source: The author.

Table 10 shows that for minting operations, Kubernetes deployments consistently consumed more memory than Docker Compose across all modes and workloads. The difference is most pronounced in DVID mode, with Kubernetes using approximately 25-37% more memory (e.g., subject-wl: 26.16 ± 3.38 MB in Docker vs. 35.91 ± 3.07 MB in Kubernetes). However, as TURIN et al. note, “memory is time independent" in container systems, with memory being "acquired and released" rather than continuously consumed like CPU resources (TURIN et al., 2023). This fundamental difference in resource management suggests that the observed memory variations might be related to application-specific behaviors rather than inherent orchestration platform characteristics.

Table 10 – Memory Usage for Minting Operation - Docker vs Kubernetes (MB).

Workload	Memory Usage (MB) - Minting							
	DVID		ID-Mode		Anon-Mode		LSVID	
	Docker	K8s	Docker	K8s	Docker	K8s	Docker	K8s
subject-wl	26.16 ± 3.38	35.91 ± 3.07	24.78 ± 2.37	31.85 ± 3.63	23.96 ± 2.25	28.95 ± 2.33	19.89 ± 2.17	24.32 ± 1.73
asserting-wl	33.01 ± 3.69	41.48 ± 3.53	23.66 ± 2.89	29.77 ± 2.12	25.35 ± 2.50	27.21 ± 2.29	21.50 ± 2.50	24.89 ± 1.58

Source: The author.

Validation operations display similar memory consumption patterns as shown in Table 11. Kubernetes deployments used 15-30% more memory across all workloads and modes. The target-wl component exhibited the highest memory requirements in both environments, particularly for DVID and ID-Mode (e.g., DVID target-wl: 34.93 ± 3.00 MB in Docker vs. 40.90 ± 3.27 MB in Kubernetes). This aligns with TURIN et al.’s fundamental observation that “memory is time independent" in container systems, with memory being “acquired and released" rather than continuously consumed like CPU resources. Their research demonstrates that this memory management model leads to different resource allocation patterns than CPU utilization, explaining our consistent observation of higher memory usage in Kubernetes across all test scenarios. Furthermore, they observed that “containers and pods affect each other’s consumption and performance when running on the same machine", which explains why these memory patterns remain consistent despite the architectural differences between deployment environments.

Table 11 – Memory Usage for Validation Operation - Docker vs Kubernetes (MB).

Workload	Memory Usage (MB) - Validation							
	DVID		ID-Mode		Anon-Mode		LSVID	
	Docker	K8s	Docker	K8s	Docker	K8s	Docker	K8s
subject-wl	33.30 ± 2.39	36.10 ± 3.12	30.88 ± 2.86	35.63 ± 2.84	19.01 ± 1.69	23.31 ± 2.05	17.22 ± 1.75	19.77 ± 1.83
asserting-wl	28.66 ± 2.63	33.26 ± 2.71	27.44 ± 2.41	30.54 ± 2.29	21.90 ± 2.05	24.98 ± 1.98	17.02 ± 2.17	21.28 ± 1.48
m-tier	30.81 ± 2.59	39.07 ± 2.94	28.16 ± 2.54	33.15 ± 2.21	24.45 ± 2.77	29.46 ± 1.94	20.37 ± 1.96	24.49 ± 1.30
target-wl	34.93 ± 3.00	40.90 ± 3.27	33.75 ± 2.92	38.68 ± 3.03	25.26 ± 2.63	31.14 ± 2.40	23.32 ± 2.74	27.26 ± 2.00

Source: The author.

Examining the standard deviations across all measurements reveals significant overlap between Docker Compose and Kubernetes distributions. While mean values consistently show differences, the standard deviation ranges indicate that these differences are often within the same statistical distribution. For example, in CPU usage for validation operations, the Asserting-wl in Anon-Mode shows Docker at 16.03 ± 3.04% vs. Kubernetes at 10.93 ± 2.18%. Despite the 5.1 percentage point difference in means, the overlapping distributions suggest the difference may not be statistically significant for individual requests. This statistical overlap supports TURIN et al.'s conclusion that resource consumption patterns are highly application-dependent rather than solely determined by the orchestration platform.

Contrary to initial expectations of higher computational overhead in Kubernetes due to orchestration complexity, the data shows that Kubernetes deployments consistently consumed less CPU while requiring more memory. This apparent paradox is supported by FERREIRA; SINNOTT's comparative study of managed Kubernetes services, which found that these environments often “instead of introducing overheads, provide performance improvements" for certain operations. Their research confirms that resource performance variations are substantially driven by the underlying infrastructure and application behavior rather than by Kubernetes itself. This finding challenges the conventional assumption that additional orchestration layers necessarily increase computational burden, suggesting instead that Kubernetes' sophisticated scheduling algorithms may optimize CPU utilization at the cost of higher memory allocation for infrastructure components.

The consistent pattern across all four assertion modes indicates that the underlying deployment architecture has a more significant impact on resource utilization than the specific assertion method. DVID mode showed the highest resource consumption in both environments, consistent with its more complex validation flow and multiple connection requirements. LSVID consistently demonstrated the lowest resource footprint, aligning with its design goal of local validation and reduced network communication. These results corroborate TURIN et al.'s methodology for predicting resource consumption, which emphasizes that workload characteristics and deployment architecture are more determinative of resource utilization than orchestration overhead.

The violin plots in Figures 13 through 16 visually confirm the substantial overlap in the distribution shapes between Docker and Kubernetes environments, supporting the assertion that while differences exist, they remain within similar statistical ranges. This finding is particularly significant for deployment planning, as it suggests that migration between container orchestration environments may not fundamentally alter the expected resource utilization profile of the system.

4.2.1 Network Performance Analysis

In addition to CPU and memory metrics, network performance represents a critical dimension for understanding the impact of orchestration platforms on security-focused microservices. Through fine-grained instrumentation of token minting and validation workflows across the unified PoC with multiple security documents (DVID, ID-Mode, Anon-Mode, LSVID), we demonstrate that orchestration overhead exhibits significant heterogeneity depending on operation characteristics and connection management strategies.

Token minting performance exhibits moderate orchestration overhead with significant variation across identity modes. Kubernetes deployments consistently demonstrate higher mean execution times compared to Docker Compose across all modes, though the magnitude of overhead varies considerably depending on the cryptographic operations involved.

Table 12 – Token Minting execution time cost (ms).

Mode	Deploy	Total	Highest Operation	% from total
DVID	Compose	59.35 ± 11.57	Zkp generation: 30.65 ± 3.24	51.6%
DVID	K8s	63.98 ± 21.11	Zkp generation: 30.60 ± 15.54	47.8%
IDMODE	Compose	16.99 ± 3.86	Jwks http fetch: 15.07 ± 2.83	88.7%
IDMODE	K8s	21.50 ± 4.81	Jwks http fetch: 19.54 ± 3.68	90.9%
ANONMODE	Compose	18.21 ± 2.26	Jwks http fetch: 14.46 ± 1.73	79.4%
ANONMODE	K8s	22.68 ± 3.23	Jwks http fetch: 18.75 ± 2.44	82.7%
LSVID	Compose	17.14 ± 115.30	Jwks http fetch: 16.43 ± 115.30	95.9%
LSVID	K8s	18.25 ± 15.79	Jwks http fetch: 17.52 ± 15.74	96.0%

Source: The author.

For DVID mode, which employs ZKP generation for delegated identity, minting requires 59.35 ± 11.57 ms on Docker Compose versus 63.98 ± 21.11 ms on Kubernetes, representing a 7.8% increase in mean time. ZKP generation dominates the minting process in both platforms, accounting for approximately 51.6% of total time in Docker Compose and 47.8% in Kubernetes.

ID-Mode and Anon-Mode exhibit more pronounced orchestration overhead, with increases of 26.6% (16.99 ms to 21.50 ms) and 24.5% (18.21 ms to 22.68 ms) respectively. In both modes, JWKS HTTP fetching dominates execution time, representing 88.7%-90.9% for ID-Mode and 79.4%-82.7% for Anon-Mode. This external service dependency amplifies the impact of Kubernetes networking abstractions. Each JWKS fetch must traverse kube-proxy iptables/netfilter rules, cluster Domain Name System (DNS) resolution, and service virtual IP routing layers absent in Docker Compose's direct bridge networking.

LSVID mode demonstrates the smallest absolute orchestration overhead (6.5% increase from 17.14 ms to 18.25 ms) but exhibits exceptionally high variability in Docker Compose (standard deviation of 115.30 ms). This anomalous variance likely stems from intermittent external service latency during JWKS fetching, which constitutes 95.9%-96.0% of total execution time. The high percentage indicates that LSVID minting is almost entirely constrained by external service performance rather than internal processing.

Validation performance must be analyzed by understanding the architectural relationship between M-Tier and Target workloads. The M-Tier service acts as an intermediary that validates incoming tokens and then forwards requests to the Target backend service. Consequently, the "Target workload request" operation visible in Table 13 represents the complete execution time of the Target workload shown in Table 14.

Table 13 – Validation execution time cost - M-Tier workload (ms).

Mode	Deploy	Total	Highest Operation	% from total
DVID	Compose	109.07 ± 2.41	Target workload request: 100.79 ± 2.18	92.4%
DVID	K8s	109.92 ± 20.19	Target workload request: 101.58 ± 19.91	92.4%
IDMODE	Compose	42.80 ± 2.46	Target workload request: 41.42 ± 2.45	96.8%
IDMODE	K8s	42.89 ± 15.39	Target workload request: 41.50 ± 15.39	96.8%
ANONMODE	Compose	103.53 ± 2.27	mTLS client setup: 100.30 ± 2.20	96.9%
ANONMODE	K8s	126.23 ± 34.66	mTLS client setup: 122.99 ± 34.64	97.4%
LSVID	Compose	2.25 ± 0.43	Target connect: 1.01 ± 0.20	44.9%
LSVID	K8s	2.36 ± 0.57	Target connect: 1.10 ± 0.37	46.6%

Source: The author.

For DVID validation, the M-Tier workload requires 109.07 ms (Compose) and 109.92 ms (K8s), with the Target workload request consuming 100.79 ms and 101.58 ms respectively, representing 92.4% of total M-Tier time. This indicates that M-Tier validation overhead (token validation, forwarding logic) accounts for only 8-9 ms, while the majority of latency stems from the Target service.

Table 14 – Validation execution time cost - Target workload (ms).

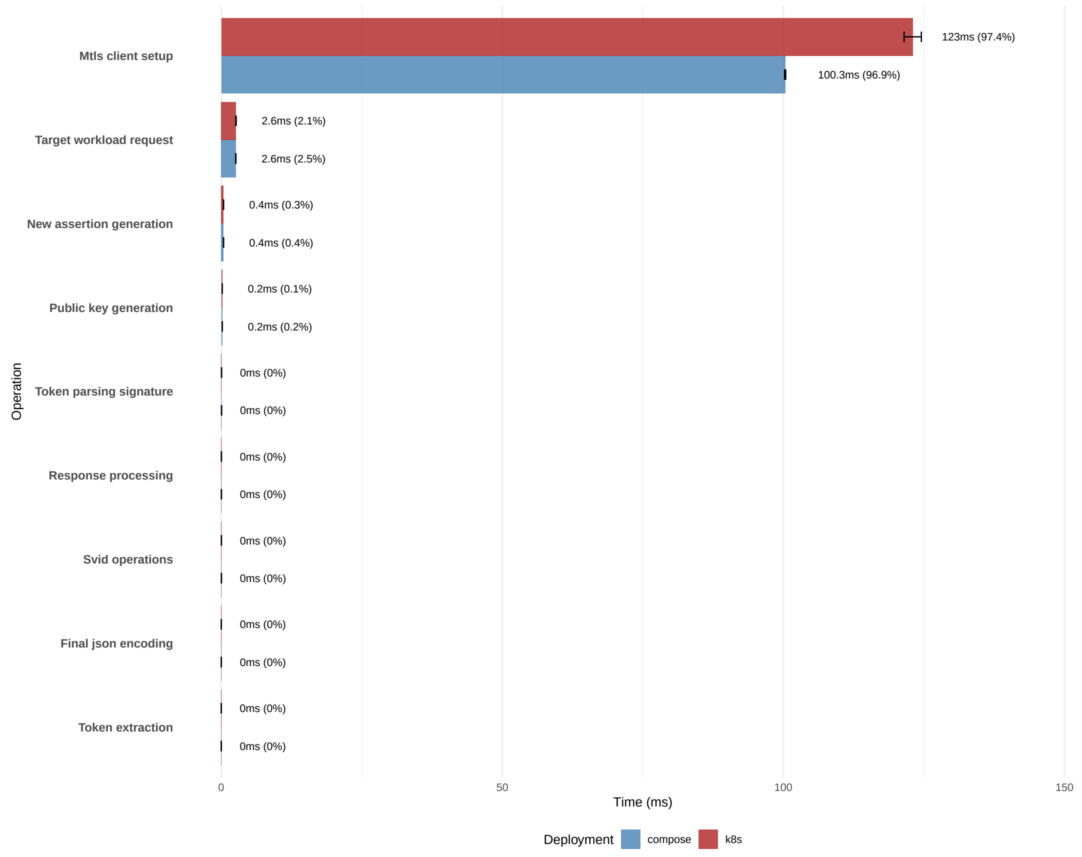
Mode	Deploy	Total	Highest Operation	% from total
DVID	Compose	100.79 ± 2.18	mTLS client setup: 97.45 ± 2.13	96.7%
DVID	K8s	101.58 ± 19.91	mTLS client setup: 98.34 ± 19.89	96.8%
IDMODE	Compose	41.42 ± 2.45	Introspect call: 40.90 ± 2.44	98.7%
IDMODE	K8s	41.50 ± 15.39	Introspect call: 40.98 ± 15.41	98.7%
ANONMODE	Compose	2.59 ± 0.31	Introspect call: 1.59 ± 0.30	61.4%
ANONMODE	K8s	2.62 ± 0.68	Introspect call: 1.58 ± 0.63	60.3%
LSVID	Compose	0.63 ± 0.16	Lsvid validation: 0.42 ± 0.11	66.7%
LSVID	K8s	0.63 ± 0.22	Lsvid validation: 0.42 ± 0.19	66.7%

Source: The author.

Examining Table 14, we see that DVID Target processing is dominated by “mTLS client setup" (96.7%-96.8% of total time), which involves retrieving X.509 certificates from the SPIRE Workload API and configuring mTLS. Despite this establish-validate-teardown pattern, orchestration overhead remains minimal in mean time (0.8% increase for both M-Tier and Target). However, Kubernetes exhibits substantially higher variability with a 8-10× variance increase.

Similar patterns emerge with ID-Mode validation, with M-Tier times of 42.80 ms (Compose) and 42.89 ms (K8s), where Target workload requests account for 41.42 ms and 41.50 ms (96.8% of M-Tier time). The orchestration overhead remains minimal at 0.2%-0.7%, but Kubernetes again shows significantly higher variability (15.39 ms standard deviation versus 2.45-2.46 ms for Compose).

Figure 17 – M-Tier ANONMODE validation.



Source: The author.

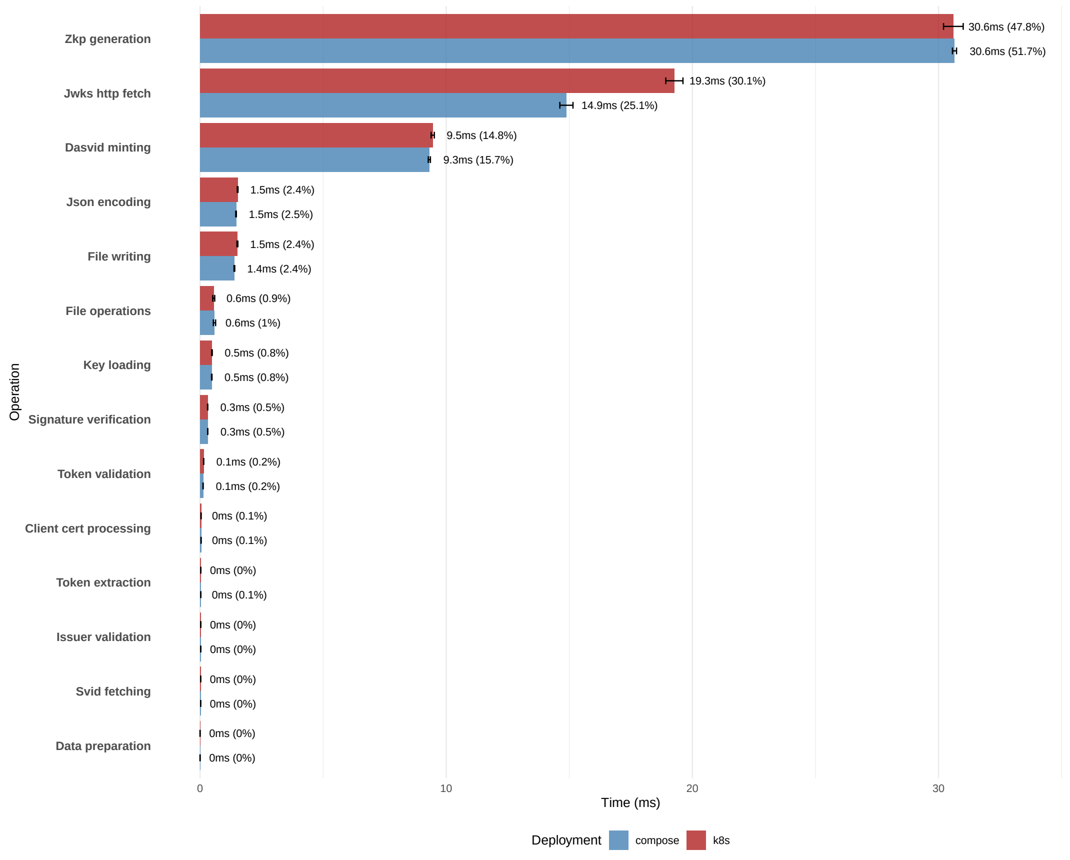
Anon-Mode validation exhibits the most substantial orchestration impact and provides critical insight into connection establishment costs. As shown in Figure 17, Anon-Mode was intentionally configured without connection pooling to isolate the overhead of repeated mTLS handshakes. M-Tier time increases 22.0% from 103.53 ms (Compose) to 126.23 ms (K8s). The dominant operation is “mTLS client setup" (96.9%-97.4% of M-Tier time), which requires 100.30 ms (Compose) versus 122.99 ms (K8s)—a 22.6% increase.

Each connection establishment must traverse the full Kubernetes service mesh stack: DNS resolution via CoreDNS, service virtual IP lookup, kube-proxy iptables/netfilter rule evaluation, and pod network routing through the CNI plugin. In contrast, Docker Compose connections utilize direct container-to-container networking on the Docker bridge network, bypassing these intermediate layers entirely.

The critical insight is that Anon-Mode's orchestration sensitivity stems entirely from repeated M-Tier connection establishment overhead, not from Target processing complexity or validation logic. By comparing Anon-Mode (no pooling, 22.6% mTLS setup overhead) against DVID (with pooling, 0.8% Target connect overhead), we quantify that connection pooling reduces orchestration overhead by approximately 21.8%. This reveals that connection establishment, specifically the repeated traversal of Kubernetes networking layers, represents the primary source of orchestration-induced performance degradation for inter-service communication.

LSVID validation shows minimal orchestration overhead across both tiers. M-Tier validation requires 2.25 ms (Compose) versus 2.36 ms (K8s)—a 4.9% increase with Target connection establishment accounting for 1.01 ms and 1.10 ms (44.9%-46.6% of M-Tier time). Target validation itself is extremely lightweight at 0.63 ms for both platforms. The identical performance across platforms suggests that when cryptographic operations are lightweight and connections are pooled, orchestration architecture has negligible impact on validation latency.

Figure 18 – DVID minting performance breakdown.



Source: The author.

Our evaluation reveals three key findings with respect to network performance:

- First, operations involving mTLS connection establishment without connection pooling experience the most substantial orchestration overhead. Anon-Mode validation, which performs repeated mTLS handshakes without connection reuse, exhibits 22.6% higher mean latency under Kubernetes compared to Docker Compose, with each connection traversing Kubernetes' multi-layer networking stack.
- Second, connection pooling effectively mitigates orchestration overhead: modes employing connection reuse (DVID, ID-Mode, LSVID) demonstrate minimal mean

latency differences (0.8%-4.9% increases), as the connection establishment cost is amortized across multiple requests. This is clearly demonstrated in Figure 18 for DVID minting operations, where despite complex cryptographic operations, the connection overhead remains minimal.

- Third, Kubernetes consistently exhibits 2–10× higher performance variability compared to Docker Compose across all modes, even when mean latencies remain similar. This increased jitter stems from the dynamic nature of Kubernetes networking components (pod scheduling, iptables rule evaluation, CNI routing decisions) and represents a fundamental characteristic of the platform rather than a transient artifact.

These findings represent a conservative lower bound for orchestration overhead, as our evaluation used Minikube's minimal networking configuration without advanced features such as network policies, service meshes, or overlay networks. Production Kubernetes deployments with more sophisticated networking stacks would likely exhibit greater overhead, particularly for security-intensive workloads requiring frequent cryptographic operations and service-to-service communication.

4.3 CHAPTER CONSIDERATIONS

The analysis of resource consumption across Docker Compose and Kubernetes deployments provides key insights into containerized microservice performance. Computational resources (CPU and memory) show minimal impact from orchestration platform choice, with differences being more application-related than architecture-dependent. The performance cost is effectively distributed across the cluster, with Kubernetes control plane services having negligible impact on application workloads.

Our investigation across different security token modes revealed that network performance is where Kubernetes introduces significant overhead, particularly for operations requiring multiple connections. DVID mode with its complex validation flow showed the greatest sensitivity to orchestration platform, though connection pooling at the M-Tier effectively reduced the overhead to just 0.8%. ID-Mode's bilateral connection pooling strategy demonstrated the most comprehensive performance optimization, maintaining minimal orchestration overhead (0.2-0.7%) despite its complex validation chain. Anon-Mode, intentionally configured without connection pooling, exhibited the highest orchestration sensitivity (22.6% increase in mTLS setup time), providing critical insight into the cost of repeated connection establishment. LSVID's lightweight design and pooled connections resulted in extremely efficient validation (4.9% overhead) and the lowest absolute execution times of any mode, confirming that self-contained cryptographic approaches can minimize network overhead.

These findings demonstrate that connection pooling serves as the essential optimization technique for microservices in Kubernetes, reducing orchestration overhead by approximately 21.8% when comparing non-pooled operations against pooled ones. Each security token mode's performance characteristics were primarily determined by its connection management strategy rather than by the inherent complexity of its cryptographic operations, suggesting that network architecture decisions have greater performance impact than the choice of orchestration platform itself.

5 CONSIDERATIONS & FUTURE WORK

This master's thesis represents the culmination of an exceptionally challenging and often frustrating. The unification process proved to be an extremely tenuous undertaking, substantially more complex than initially anticipated, requiring comprehensive refactoring of four PoCs. More than significant time was invested in understanding the nuanced interactions between components, reconstructing the original design intent, and implementing a router-based architecture that preserved functional equivalence while eliminating redundancy. The development process was particularly challenging. Debugging the unified implementation revealed unexpected edge cases and integration challenges that were not apparent in the isolated deployments. Many weeks were spent troubleshooting subtle issues with certificate handling and token propagation across the microservice tiers.

Initial performance testing yielded results that were not exactly as expected, somewhat defying our hypotheses about orchestration overhead and necessitating deeper investigation into networking patterns and connection management strategies. This investigative pivot ultimately yielded our most significant insight: that connection pooling effectively neutralizes Kubernetes orchestration overhead, providing a 21.8% performance improvement compared to non-pooled operations.

The benchmarking methodology itself presented unexpected challenges that required significant adjustments. The initial approach attempted to collect performance metrics directly within the application code, instrumenting key functions to measure execution times and resource utilization. However, this approach quickly proved problematic. Benchmarking within the application itself made the results unreliable, since the measurement process consumed computational resources that the application needed. As a result, the benchmark and the application competed for the same resources, introducing significant noise and distorting the data.

This realization necessitated a complete redesign of the benchmarking strategy. The solution involved implementing a containerized Prometheus instance as an external observer, completely separate from the application under test. This approach eliminated the resource contention issue by ensuring the monitoring infrastructure operated independently from the application components. Prometheus periodically scraped metrics from application endpoints without interfering with normal operation, providing more reliable and consistent measurements. This methodology proved effective across both Docker Compose and Kubernetes environments, enabling valid comparison between deployment models.

A notable finding from this research concerns the limitations of CPU and memory metrics when evaluating microservice architectures. While these metrics provided valuable insights in previous SPIFFE-IdT phases, they proved less informative in the orchestrated environment. In microservices, performance bottlenecks often stem from network communication, service discovery delays, and connection management rather than computational resource constraints. Services might show minimal CPU utilization while experiencing significant end-to-end latency due to these distributed factors. This suggests that request flow metrics and service interaction patterns provide more meaningful performance indicators for security document validation in microservice architectures than traditional resource utilization measurements.

During the course of this master's thesis research, the author contributed significantly to the academic literature, resulting in the publication of five peer-reviewed papers:

- Jessup, A., Cochak, H. Z., Koslovski, G. P., Pillon, M. A., Miers, C. C., Correia, P. H. B., Marques, M. A., & Simplicio, M. A. (2024). **DVID: Adding Delegated Authentication to SPIFFE Trusted Domains**. In L. Barolli (Ed.), Advanced Information Networking and Applications (pp. 289–300). Springer Nature Switzerland.
- Cochak, H., Neto, M., Miers, C., Marques, M., & Simplicio Jr., M. A. (2024). **Enhancing SPIFFE/SPIRE Environment with a Nested Security Token Model**. Proceedings of the 14th International Conference on Cloud Computing and Services Science - CLOSER, 184-191.
- Cochak, H. Z., Miers, C. C., Correia, P. H. B., Marques, M. A., & Simplicio, M. A. (2024). **Lightweight SPIFFE Verifiable Identity Document (LSVID): A Nested Token Approach for Enhanced Security and Flexibility in SPIFFE**. 2024 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 9-16.
- Cardoso, L. C., Marques, M. A., Correia, P. H. B., Cochak, H. Z., Miers, C. C., & Simplicio, M. A. (2025). **Next-Generation SPIFFE/SPIRE Identity Management Systems with Post-Quantum Cryptography Algorithms**. IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 154-163.
- Cochak, H. Z., Miers, C. C., Marques, M. A., & Simplicio, M. A. (2025). **Container Orchestration Impact on SPIFFE Identity Artifacts: A Performance Analysis of Docker vs Kubernetes**. 16th IEEE International Conference on Cloud Computing Technology and Science (CloudCom).

Through this discovery and the comprehensive unification work, this research successfully concludes the three-year SPIFFE-IdT project, consolidating into a unified architecture with reproducible deployment procedures. Future work could explore the development of new security documents incorporating post-quantum cryptographic algorithms, addressing the emerging threat of quantum computing to current cryptographic foundations. Particularly promising avenues include lattice-based schemes like CRYSTALS-Kyber or FALCON, and hash-based signatures like SPHINCS+.

Building upon our connection pooling findings, further research could examine horizontal scalability under high-volume workloads, leveraging Kubernetes' autoscaling capabilities to dynamically adjust resource allocation based on authentication demand patterns. Integrating the unified prototype with advanced service mesh technologies like Istio could provide additional security enforcement layers while potentially mitigating the connection establishment overhead identified in our research. Extending the performance analysis to distributed multi-node Kubernetes clusters would provide insights into how geographic dispersion affects identity validation latencies, particularly relevant for global-scale deployments.

BIBLIOGRAPHY

ABOBA, D. B. D.; WOOD, J. **Authentication, Authorization and Accounting (AAA) Transport Profile**. RFC Editor, 2003. RFC 3539. (Request for Comments, 3539). Disponível em: <www.rfc-editor.org/info/rfc3539>.

ALLIANCE, C. S. **CSA Security Guidance for Critical Areas of Focus in Cloud Computing**. 2023. <https://cloudsecurityalliance.org/research/guidance/>.

AQASIZADE, H.; ATAIE, E.; BASTAM, M. Kubernetes in action: Exploring the performance of kubernetes distributions in the cloud. **Software: Practice and Experience**, Wiley, 2025. Published online: 2 July 2025.

ASCENSÃO, P. et al. Assessing kubernetes distributions: A comparative study. In: **2024 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.]: IEEE, 2024.

BARKER, S. Identity, persistence, and the ship of theseus. **Philosophy Compass**, 2019.

BARLETTA, M. et al. Slo-aware orchestration for 5G network functions and microservices. **Journal of Network and Systems Management**, 2025. In press.

BERNERS-LEE, T.; FIELDING, R. T.; MASINTER, L. M. **Uniform Resource Identifier (URI): Generic Syntax**. RFC Editor, 2005. RFC 3986. (Request for Comments, 3986). Disponível em: <https://www.rfc-editor.org/info/rfc3986>.

BOEYEN, S. et al. **Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**. RFC Editor, 2008. RFC 5280. (Request for Comments, 5280). Disponível em: <www.rfc-editor.org/info/rfc5280>.

BÖHM, S.; WIRTZ, G. Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes. In: **ZEUS**. [S.l.: s.n.], 2021. p. 65–73.

BRAUN, I.; HOFFMANN, M.; MÖRSEBURG, R. Implementation of a web-based audience response system as microservice application vs monolithic application. **MONOLITHIC APPLICATION**, 2019.

COCHAK., H. et al. Enhancing spiffe/spire environment with a nested security token model. In: INSTICC. **Proceedings of the 14th International Conference on Cloud Computing and Services Science - CLOSER**. [S.l.]: SciTePress, 2024. p. 184–191. ISBN 978-989-758-701-6. ISSN 2184-5042.

COCHAK, H. Z. et al. Lightweight spiffe verifiable identity document (lsvid): A nested token approach for enhanced security and flexibility in spiffe. In: **2024 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2024. p. 9–16.

DIN, M. S. U. et al. A testbed implementation of microservices-based in-network computing framework for information-centric iovs. In: IEEE. **2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)**. [S.l.], 2022. p. 1–5.

FAYOS-JORDAN, R. et al. Performance comparison of container orchestration platforms with low cost devices in the fog, assisting internet of things applications. **Journal of Network and Computer Applications**, 2020.

FELDMAN, D. et al. **Solving the Bottom Turtle — a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity**. 1. ed. 2020. ISBN 978-0-578-77737-5.

FERREIRA, A. P.; SINNOTT, R. A performance evaluation of containers running on managed kubernetes services. In: IEEE. **2019 IEEE international conference on cloud computing technology and science (CloudCom)**. [S.l.], 2019. p. 199–208.

FOWLER, M.; LEWIS, J. Eb/ol. microservices definition of this new architectural term. **Microservices Definition of this New Architectural Term**, 2014.

GOLDWASSER, S.; MICALI, S.; RACKOFF, C. The knowledge complexity of interactive proof systems. **SIAM Journal on computing**, SIAM, v. 18, n. 1, p. 186–208, 1989.

GONZALEZ, N. et al. A quantitative analysis of current security concerns and solutions for cloud computing. **Journal of Cloud Computing: Advances, Systems and Applications**, Springer, v. 1, p. 1–18, 2012.

GRASSI, P. A.; GARCIA, M. E.; FENTON, J. L. Digital identity guidelines. **NIST special publication**, v. 800, p. 63–3, 2017.

JESSUP, A. et al. Dvid: Adding delegated authentication to spiffe trusted domains. In: BAROLLI, L. (Ed.). **Advanced Information Networking and Applications**. Cham: Springer Nature Switzerland, 2024. p. 289–300. ISBN 978-3-031-57916-5.

JONES, M. B. **JSON Web Algorithms (JWA)**. RFC Editor, 2015. RFC 7518. (Request for Comments, 7518). Disponível em: <https://www.rfc-editor.org/info/rfc7518>.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. **JSON Web Signature (JWS)**. RFC Editor, 2015. RFC 7515. (Request for Comments, 7515). Disponível em: <https://www.rfc-editor.org/info/rfc7515>.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. RFC Editor, 2015. RFC 7519. (Request for Comments, 7519). Disponível em: <https://www.rfc-editor.org/info/rfc7519>.

KJORVEZIROSKI, V.; FILIPOSKA, S. Kubernetes distributions for the edge: Serverless performance evaluation. **The Journal of Supercomputing**, Springer, v. 78, n. 11, p. 13728–13755, 2022.

KOUKIS, G. et al. Performance evaluation of kubernetes networking approaches across constraint edge environments. In: **2024 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.]: IEEE, 2024. p. 1–7.

KOZIOLEK, H.; ESKANDANI, N. Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift. In: **Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering**. New York, NY, USA: ACM, 2023. (ICPE ’23), p. 17–29.

Kubernetes. **Volumes: hostPath**. 2025. <https://kubernetes.io/docs/concepts/storage/volumes/#hostpath>. Online Documentation. Access in: October 11, 2025.

Kubernetes CSI. **Deploying a CSI Driver on Kubernetes**. 2024. <https://kubernetes-csi.github.io/docs/deploying.html>. Online Documentation. Access in: October 11, 2025.

KUMAR, R.; GOYAL, R. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. **Computer Science Review**, v. 33, p. 1–48, 2019. ISSN 1574-0137. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1574013718302065>.

LABS, G. **Grafana: The open observability platform**. 2025. <https://grafana.com/grafana/>. Online. Access in: October 10, 2025.

MALER, E.; REED, D. The venn of identity: Options and issues in federated identity management. **IEEE security & privacy**, IEEE, v. 6, n. 2, p. 16–23, 2008.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. [S.l.]: Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2011.

MICROSOFT. **What is Cloud Computing**. 2024. <https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/>. Online Forum Post. Access in: Sep 15. 2024.

MICROSOFT. **What is unified communications as a service?** 2024. <https://www.microsoft.com/en-us/microsoft-teams/unified-communications-as-a-service>. Online Forum Post. Access in: Sep 15. 2024.

OKTA. **What Is Federated Identity?** 2024. Online forum post. Accessed on November 14, 2024. Disponível em: <https://www.okta.com/identity-101/what-is-federated-identity/>.

PAPADOPOULOS, G. T. Kubernetes in edge and cloud computing: A comparative study of k3s, k0s, microk8s, and k8s. In: **2025 6th International Conference in Electronic Engineering & Information Technology (EEITE)**. [S.l.]: IEEE, 2025.

PEDNEKAR, S. et al. A comparative analysis of kubernetes and openshift based on workloads using different hardware architecture. In: **2024 International Conference on Computing and Technology**. [S.l.]: IEEE, 2024.

PROMETHEUS. **Overview**. 2025. <https://prometheus.io/docs/introduction/overview/>. Online Forum Post. Access in: January 12. 2025.

RAMADAN, I. et al. Evaluating kubernetes distributions: Insights from stress testing scenarios. In: **2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)**. [S.l.]: IEEE, 2025.

REDHAT. **What is CaaS**. 2024. <https://www.redhat.com/en/topics/cloud-computing/what-is-caas>. Online Forum Post. Access in: Sep 15. 2024.

ROSE, S. et al. **Zero Trust Architecture**. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2020. Disponível em: <https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420>.

SECURITY, C. U.S. Department of H.; (CISA), I. S. A. **Continuous Diagnostics and Mitigation (CDM) Identity, Credential, and Access Management (ICAM) Reference Architecture**. [S.l.], 2023.

SECURITY, C. U.S. Department of H.; (CISA), I. S. A. **ICAM 101 Briefing for Public Safety Officials**. 2023. Online forum post. Accessed on November 04, 2024. Disponível em: <https://www.cisa.gov/sites/default/files/2023-02/17_0515_ICAM_101_Briefing_for_Public_Safety_Officials_FINAL508.pdf>.

SHARIFIAN, A. et al. An http connection pool for reducing web latency. In: **IEEE International Conference on Web Services (ICWS)**. [S.l.: s.n.], 2019. p. 98–105.

SPIFFE. **Special Interest Group SPIFFE Specification**. 2024. Online Forum Post. Access in: December 04. 2024. Disponível em: <https://groups.google.com/a/spiffe.io/g/sig-specification>.

SPIFFE. **SPIFFE Standards**. 2024. <https://github.com/spiffe/spiffe/tree/main/standards>. SPIFFE Online Documentation. Access in: Oct 04. 2025.

SPIFFE. **SPIFFE | CNCF**. 2025. <https://www.cncf.io/projects/spiffe/>. Online Forum Post. Access in: December 04. 2025.

SPIFFE. **SPIFFE Concepts**. 2025. Https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/. SPIFFE Online Documentation. Access in: Oct 04. 2025.

SPIFFE. **SPIFFE Quickstart Kubernetes**. 2025. <https://spiffe.io/docs/latest/try/getting-started-k8s/>. SPIFFE Online Documentation. Access in: Oct 04. 2025.

STAFFORD, V. Zero trust architecture. **NIST special publication**, v. 800, p. 207, 2020.

SUN, Y.; NANDA, S.; JAEGER, T. Security-as-a-service for microservices-based cloud applications. In: IEEE. **2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.], 2015. p. 50–57.

TELENYK, S. et al. A comparison of kubernetes and kubernetes-compatible platforms. In: **2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)**. [S.l.]: IEEE, 2021. v. 1, p. 313–317.

TURIN, G. et al. Predicting resource consumption of kubernetes container systems using resource models. **Journal of Systems and Software**, Elsevier, v. 203, p. 111750, 2023.

YAKUBOV, D.; HÄSTBACKA, D. Comparative analysis of lightweight kubernetes distributions for edge computing: Performance and resource efficiency. In: **Service-Oriented and Cloud Computing**. Cham: Springer, 2025. (ESOCC 2025, Lecture Notes in Computer Science, v. 15547), p. 81–95.

YAKUBOV, D.; HÄSTBACKA, D. Comparative analysis of lightweight kubernetes distributions for edge computing: Security, resilience and maintainability. In: **Service-Oriented and Cloud Computing**. Cham: Springer, 2025. (ESOCC 2025, Lecture Notes in Computer Science, v. 15547), p. 96–104.

ČILIć, I. et al. Performance evaluation of container orchestration tools in edge computing environments. In: **Sensors**. [S.l.: s.n.], 2023.

APPENDIX A – RESULTS ON PAST IMPLEMENTATIONS/DEPLOYS

Throughout the course of the project, several papers have been created, each focusing on the benchmarking of different PoCs related to the development and integration of identity documents within the SPIFFE ecosystem. These papers collectively aim to provide empirical data on the system's performance, analyzing key metrics such as execution time, CPU consumption, memory utilization, and token size growth across phases of the project.

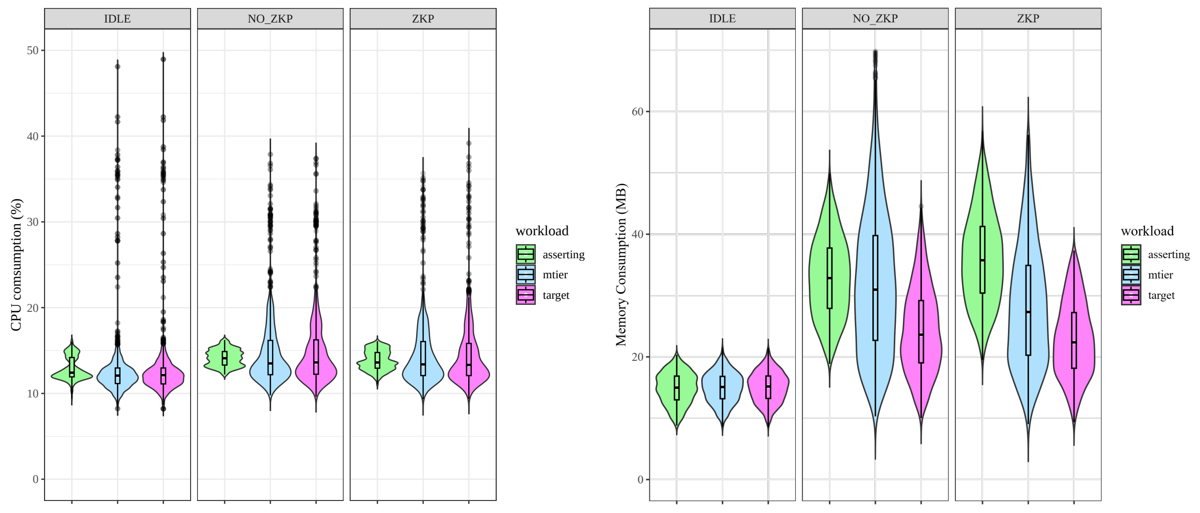
The results presented in these papers are organized according to the different phases of the project, each introducing new techniques, protocols, and challenges. While these papers provide valuable insights, it should be noted that the results obtained in the published works (JESSUP et al., 2024), (COCHAK. et al., 2024), and (COCHAK et al., 2024) are not directly considered in the present thesis. This is because they were obtained with a different architectural approach (single deployment for each PoC, Docker Compose only) and were conducted prior to the writing of this thesis. Nevertheless, these preliminary studies were instrumental in guiding the architectural decisions and performance evaluation methodology adopted for the unified proof of concept presented herein. The complete analysis of the preliminary results can be found in three papers, (JESSUP et al., 2024), (COCHAK. et al., 2024), and (COCHAK et al., 2024).

A.1 PHASE 1 - DVID

As explained in Subsection 2.2.1.1, the focus was on the development and benchmarking of the DVID, enabling SPIFFE workloads to authenticate users without introducing additional trusted entities. The benchmarking targeted key operations involved in the minting and validation endpoints of the DVIDs, with a specific focus on performance metrics such as execution time, CPU consumption, memory usage, and the impact of RSA-ZKP proofs on system efficiency. The results, outlined in (JESSUP et al., 2024), are summarized as follows partially detailed in Figure 19.

CPU and Memory Consumption did not significantly differ between scenarios with and without RSA-ZKP, contrary to initial expectations. As shown in Figure 19, both CPU and memory usage remained relatively consistent across scenarios. This suggests that the overhead typically associated with cryptographic operations like RSA-ZKP was minimal. The efficient memory management provided by the Golang garbage collector and potential local storage optimizations may have contributed to the negligible variance observed. Regarding execution time, adding RSA-ZKP resulted in a

Figure 19 – Phase 1 - Resources consumption.



(a) CPU consumption (b) Memory consumption

Source: (JESSUP et al., 2024).

noticeable increase in time. The creation of a DVID document took 221.80 ± 37.22 ms without the RSA-ZKP, but this increased to 461.07 ± 77.78 ms when the proof was included, adding approximately 240 ms to the operation. Similarly, validation of a document containing RSA-ZKP took 270.13 ± 76.13 ms, compared to just 11.61 ± 0.84 ms for documents without the proof. This substantial increase in validation time highlights the computational overhead of validating RSA-ZKP proofs, particularly in cloud environments.

A.2 PHASE 2 - NESTED MODEL

Previously detailed in Subsection 2.2.1.2, this phase targeted the benchmarking of key operations of the nested token model, focusing on execution time, CPU consumption, memory usage, token size growth, and the impact of each signature schemes on system performance. Table 15 summaries the resource consumption of workloads during execution while Table 16 illustrates the time cost of specific functions. The complete work is found in (COCHAK. et al., 2024).

Table 15 – Phase 2 - Resource consumption.

	CPU			**Memory**		
Workload	**Idle (%)**	**Anon-Mode (%)**	**ID-Mode (%)**	**Idle (MB)**	**Anon-Mode (MB)**	**ID-Mode (MB)**
Front-End	14,3 ± 4,0	26,3 ± 5,4	28,7 ± 1,7	13.4 ± 2.3	19.7 ± 3.7	31.0 ± 6.4
Local IdP/TTP	14,4 ± 4,0	26,3 ± 5,4	28,8 ± 4,0	13.6 ± 2.3	18.3 ± 3.4	26.1 ± 5.0
Middle-Tier$_1$	14,3 ± 4,0	26,3 ± 5,3	28,8 ± 4,0	13.6 ± 2.4	22.2 ± 4.7	28.6 ± 5.6
Middle-Tier$_2$	14,3 ± 4,0	26,2 ± 5,4	28,8 ± 4,0	13.6 ± 2.4	22.3 ± 4.7	29.1 ± 5.8
Middle-Tier$_3$	14,3 ± 4,0	26,3 ± 5,3	28,8 ± 3,9	13.4 ± 2.4	22.3 ± 4.7	29.0 ± 5.6
Middle-Tier$_4$	14,3 ± 4,0	26,3 ± 5,3	28,7 ± 4,0	13.4 ± 2.4	22.3 ± 4.7	28.7 ± 5.8
Middle-Tier$_5$	14,3 ± 4,0	26,3 ± 5,4	28,8 ± 3,9	13.4 ± 2.4	22.5 ± 4.8	29.2 ± 5.9
Target	14,7 ± 3,9	26,3 ± 5,3	28,8 ± 4,0	13.9 ± 2.5	25.0 ± 5.7	34.9 ± 7.1

Adapted from: (COCHAK. et al., 2024).

The memory consumption results show a consistent trend of increasing usage as the token becomes more nested. As expected, the resource access layer exhibits the highest memory usage due to its role in final validation and value storage. The overall increase in memory consumption corresponds to the growth in token size, which occurs with each nesting and issuance process across the components. The comparison between Anon-Mode and ID-Mode highlights that ID-Mode incurs a higher memory cost. This is primarily due to the need to store and redirect sets of SVID certificates for validation at each component. In contrast, the nested token exhibits linear growth in its payload, meaning the impact of nesting extensions is constrained with each hop.

Despite the differences in memory usage, the execution time for both modes in handling the nested token can be considered efficient, with minimal variation during the issuance process. The use of identity documents in ID-Mode directly affects computational resource consumption, although it still requires less time for token validation compared to Anon-Mode. While Anon-Mode benefits from lower resource consumption due to the use of concatenated signatures, it comes at the cost of considerably higher validation and execution times compared to ID-Mode.

Table 16 – Phase 2 - Execution Time.

	ID-Mode		**Anon-Mode**	
Workload	**Token Minting (μs)**	**Validation (μs)**	**Token Minting (μs)**	**Validation (μs)**
Front-end	217,85 ± 347,56	351,23 ± 84,56	1007,87 ± 249,24	-
Middle-Tier$_1$	187,26 ± 34,39	708,39 ± 166,30	992,67 ± 234,95	-
Middle-Tier$_2$	189,86 ± 41,57	1067,33 ± 195,27	1015,01 ± 246,86	-
Middle-Tier$_3$	189,69 ± 36,34	1482,79 ± 352,94	995,63 ± 234,02	-
Middle-Tier$_4$	188,54 ± 36,60	1800,60 ± 376,99	1030,39 ± 325,15	-
Middle-Tier$_5$	196,26 ± 38,60	2209,10 ± 466,36	1039,25 ± 302,71	-
Target	**-**	2554,14 ± 472,53	-	4616,95 ± 1012,38

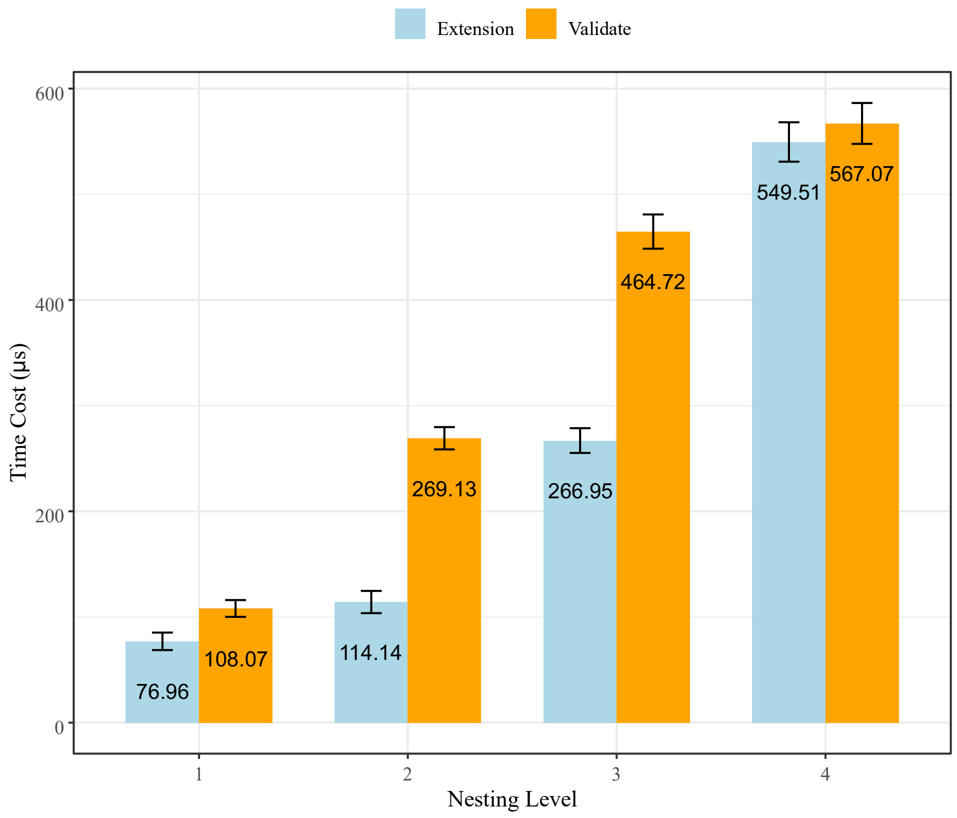
Adapted from: (COCHAK. et al., 2024).

For the ID-Mode, the token minting cost is relatively consistent across different components, but the Front-End workload exhibits a higher standard deviation compared to the Middle-Tier components. Validation times, however, increase as the token is nested more times. This increase is due to the added complexity of validating multiple signatures, leading to higher execution time and resource consumption as more extensions are added. In Anon-Mode, the execution time for token minting is similar across all workloads, but the minting process takes longer compared to ID-Mode. This is due to the additional key extraction process required before signing the new token. Regarding validation, Anon-Mode shows significantly higher costs compared to ID-Mode, as it utilizes a concatenated signature approach. This method requires recursive public key computation during signature validation, resulting in increased execution time.

A.3 PHASE 3 - LSVID

Detailed in Section 2.2.1.3, this phase, the last of the project, used the Nested Model scheme, specifically the ID-Mode, to implement and evaluate the LSVID, with the project main objective to integrate it into the SPIFFE framework. In this phase, the focus was placed on evaluating the performance of the document, with the metrics detailed in Table 4. Figure 20 picture partially some obtained results.

Figure 20 – Phase 3 - Execution Time.



Source: (COCHAK et al., 2024).

The results point out the cost of the LSVID approach. Both the minting and validation operations for LSVIDs are fast, all completed within the time domain required-less than 1 ±s, indicating their suitability for low-latency environments. This makes LSVID particularly beneficial for authentication systems where speed is a crucial factor. One of the key advantages of the LSVID over JWT-SVID is its ability to handle nested tokens. While JWT-SVIDs natively support nesting, it is not implemented in practice. In scenarios where hierarchical or complex authorization structures are required, this becomes a significant limitation. Each additional level of nesting in JWT-SVID necessitates the issuance of a separate token, adding complexity and overhead due to the need to independently manage, validate, and handle each token.

In JWTs, nesting is achieved by embedding one JWT inside another, which requires each nested token to be treated as an independent unit, including its own header, payload, and signature. Each JWT-SVID is self-contained, and typically ranges from 2 kB to 4 kB, depending on data size and signature length. To achieve a nesting effect with JWT-SVIDs, you would need to embed multiple tokens, essentially creating a chain of JWTs. For example, nesting four JWT-SVIDs would result in a combined size that could range from 8 kB to 16 kB, depending on each individual token’s size. This combined size is generally larger than that of a single LSVID with the same number of nesting levels.

ATESTADO DE VERSÃO FINAL

Eu, CHARLES CHRISTIAN MIERS, professor(a) do curso de MESTRADO EM COMPUTAÇÃO APLICADA, declaro que esta é a versão final aprovada pela comissão julgadora da dissertação/tese intitulada: **“Container Orchestration Impact on SPIFFE Identity Artifacts: A Performance Analysis of Docker vs Kubernetes”** de autoria do(a) acadêmico HENRIQUE ZANELA COCHAK.

JOINVILLE, 02 de DEZEMBRO de 2025.

Assinatura digital do(a) orientador(a):

gov.br
Documento assinado digitalmente
CHARLES CHRISTIAN MIERS
Data: 02/12/2025 21:22:10-0300
Verifique em https://validar.iti.gov.br

CHARLES CHRISTIAN MIERS