



Universidade Federal de Pernambuco
Centro de Informática
Graduate Program in Computer Science

Lucas Aurelio Gomes Costa

Zero Trust and service meshes on microservice cloud-based applications: A comparative study

Recife

2024

Lucas Aurelio Gomes Costa

Zero Trust and service meshes on microservice cloud-based applications: A comparative study

Research presented to the Graduate Program in Computer Science of the Centro de Informática of the Universidade Federal de Pernambuco as a partial requirement for obtaining the Masters degree in Computer Science.

Area of Concentration: Computer networks and distributed systems

Advisor: Carlos André Guimarães Ferraz

Recife

2024

Catálogo na fonte
Bibliotecária: Luiza de Oliveira/CRB 1316

C837z Costa, Lucas Aurelio Gomes.

Zero Trust and service meshes on microservice cloud-based applications: a comparative study / Lucas Aurelio Gomes Costa – 2024.
93 f. il., tab., fig.

Orientador: Carlos André Guimarães Ferraz.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. Centro de Informática. Programa de Pós-graduação em Ciência da Computação, Recife, 2024.
Inclui referências.

1. Service meshes. 2. Zero trust. 3. Istio. 4. Linkerd. 5. Microserviços. 6. Google cloud. I. Ferraz, Carlos André Guimarães. II. Título.

004.6

CDD (23. ed.)

UFPE - CCEN 2024 – 92

Lucas Aurelio Gomes Costa

Zero Trust and service meshes on microservice cloud-based applications: A comparative study

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 07 de Fevereiro de 2024.

BANCA EXAMINADORA

Prof. Dr. Nelson Souto Rosa (Examinador Interno)
Centro de Informática/UFPE

Prof. Dr. Ioram Schechtman Sette (Examinador Externo)
CESAR School

Prof. Dr. Carlos André Guimarães Ferraz (Orientador)
Centro de Informática/UFPE

I dedicate this research to myself, my greatest critic and supporter, for looking for more and never remaining satisfied, for pushing forward when it seemed impossible, and for making my dreams come true.

ACKNOWLEDGEMENTS

- To God, for guiding me through challenges and keeping me on the path of light.
- To my parents: thank you for supporting my decisions and helping me overcome setbacks.
- To my patient and comprehensive advisor, thank you for pushing me to discover my best even when I struggled.
- To my friends: thank you for understanding my absences and supporting me in my research.

Nothing has meaning on its own. Meaning is something that gets determined later. Humans are born, grow up, and die all without meaning. It's only when a life is over that you finally see what it meant. [Romani Archaman, 2018, Prelude, s. 2]

ABSTRACT

Migrating microservices to a cloud environment poses challenges for maintaining security. Though Zero-Trust architecture provides guidelines on protecting the services, protecting the applications is still a major concern for companies. Research has shown that service meshes, such as Istio or Linkerd, can facilitate protection for services in a Kubernetes environment.

This study aims to understand how service meshes can enable Zero-Trust approaches to service-to-service communication. Investigating how Zero-Trust protection aligns with service mesh capabilities, how it can affect service communication performance, and how Istio and Linkerd compare to each other in terms of security and performance.

This research used experiments as the key part of the process to fulfill its objectives. A proof-of-concept architecture was implemented to facilitate experiments, while the experiments were divided into two categories (security and performance) and the results were used to compare Istio to Linkerd.

Analysis of the experiments has shown that Linkerd is faster than Istio while providing similar levels of protection.

The results indicate that different security configurations for service meshes decrease service communication performance and how these configurations align with Zero-Trust guidelines. Based on this information, companies seeking to enforce Zero-Trust protection to services in the cloud must consider the compromises required between performance and security.

Keywords: service meshes; zero trust; istio; linkerd; microservices; cloud; google cloud; kubernetes.

RESUMO

A migração de microsserviços para um ambiente na nuvem apresenta desafios na preservação da segurança. Apesar da arquitetura *Zero-Trust* prover diretrizes para a proteção dos serviços, garantir a segurança das aplicações permanece uma das principais preocupações para as empresas. A pesquisa apresentou que *service meshes*, como Istio ou Linkerd, podem facilitar a proteção para serviços em ambientes Kubernetes.

Esta pesquisa busca entender como *service meshes* podem habilitar abordagens Zero-Trust para a comunicação de serviço a serviço, investigando como a proteção *Zero-Trust* se alinha com as capacidades de uma *service mesh*, como isso afeta o desempenho da comunicação serviço a serviço e como Istio e Linkerd comparam-se em termos de segurança e desempenho.

Esta pesquisa utilizou experimentos como a parte principal do processo para atingir seus objetivos. Uma arquitetura de prova de conceito foi implementada para facilitar os experimentos. Os experimentos foram divididos em duas categorias, segurança e desempenho, e os resultados foram utilizados para comparar Istio a Linkerd.

A análise dos experimentos apresentou que Linkerd é mais rápido que Istio enquanto provê níveis semelhantes de proteção.

Os resultados indicam que diferentes configurações de segurança para as *service meshes* reduzem o desempenho da comunicação serviço a serviço e também como tais configurações alinham-se às diretrizes de *Zero Trust*. Baseando-se nessas informações, empresas que buscam aplicar a proteção *Zero-Trust* aos serviços na nuvem precisam considerar um equilíbrio entre desempenho e segurança.

Palavras-chaves: service meshes; zero trust; istio; linkerd; microsserviços; nuvem; google cloud; kubernetes.

LIST OF FIGURES

Figure 1 – Heavy cloud usage growth.	16
Figure 2 – Cloud challenges of organizations.	17
Figure 3 – Service meshes usage.	18
Figure 4 – Kubernetes architecture by example.	27
Figure 5 – Service to service communication without Istio.	30
Figure 6 – General architecture of the Istio mesh.	32
Figure 7 – General architecture of the Linkerd mesh.	35
Figure 8 – Service mesh layer between application and orchestration.	38
Figure 9 – Latency results with Istio in a multi-cloud environment.	39
Figure 10 – Research method overview.	42
Figure 11 – Project architecture.	43
Figure 12 – Front end’s user interface.	44
Figure 13 – Illustration of the results user interface (UI).	47
Figure 14 – Security case where no services are meshed.	57
Figure 15 – Security case where all services are meshed.	58
Figure 16 – Security case where only the Calculator service is not meshed.	59
Figure 17 – Security case where only the Security Checker and Experimenter services are not meshed.	60
Figure 18 – Security case where <i>msc-external</i> is meshed, but the <i>mastership</i> namespace is not.	61
Figure 19 – Security case where <i>msc-external</i> is not meshed, but the <i>mastership</i> names- pace is.	62
Figure 20 – Services used on round-trip time (RTT) evaluation.	63
Figure 21 – Confirming Transport Layer Security (TLS) with Istio.	66
Figure 22 – Confirming Transport Layer Security (TLS) with Linkerd.	67
Figure 23 – Candlestick graph for round-trip times (RTTs) found on the preliminary results.	73
Figure 24 – Histogram for round-trip times (RTTs) found on the preliminary and local evaluation for the no-mesh scenario.	74

Figure 25 – Histogram for round-trip times (RTTs) found on the preliminary evaluation for the Istio scenario.	74
Figure 26 – Histogram for round-trip times (RTTs) found on the preliminary evaluation for the Linkerd scenario.	75
Figure 27 – Candlestick graph for round-trip times (RTTs) found on the local results. .	76
Figure 28 – Histogram for round-trip times (RTTs) found on the local evaluation for the Istio scenario.	77
Figure 29 – Histogram for round-trip times (RTTs) found on the local evaluation for the Linkerd scenario.	77
Figure 30 – Candlestick graph for round-trip times (RTTs) found on the local environ- ment with greater numbers.	79
Figure 31 – Histogram for round-trip times (RTTs) found on the local evaluation with bigger numbers for the no-mesh scenario.	79
Figure 32 – Histogram for round-trip times (RTTs) found on the local evaluation with bigger numbers for the Istio scenario.	80
Figure 33 – Histogram for round-trip times (RTTs) found on the local evaluation with bigger numbers for the Linkerd scenario.	80
Figure 34 – Candlestick graph for round-trip times (RTTs) found on the cloud results. .	82
Figure 35 – Histogram for round-trip times (RTTs) found on the cloud evaluation for the no-mesh scenario.	82
Figure 36 – Histogram for round-trip times (RTTs) found on the cloud evaluation for the Istio scenario.	83
Figure 37 – Histogram for round-trip times (RTTs) found on the cloud evaluation for the Linkerd scenario.	83

LIST OF TABLES

Table 1 – Academic works found in the literature review.	37
Table 2 – Security cases.	56
Table 3 – Workload for performance evaluation.	64
Table 4 – Security results found when no mesh is applied.	68
Table 5 – Security results found when the Istio mesh is applied, the * mark noteworthy results.	69
Table 6 – Security results found when the Linkerd mesh is applied, the * mark noteworthy results.	70
Table 7 – Summary of the security results.	71
Table 8 – Comparison between the different scenarios on the preliminary evaluation.	72
Table 9 – Candlestick graph data of the preliminary results.	73
Table 10 – Comparison between the different scenarios on the local evaluation.	75
Table 11 – Candlestick graph data of the local results.	76
Table 12 – Comparison between the evaluation of the different scenarios on the local environment with bigger numbers.	78
Table 13 – Candlestick graph data of the results found with the local environment with bigger numbers.	79
Table 14 – Comparison between the different scenarios on the cloud evaluation.	81
Table 15 – Candlestick graph data of the cloud results.	81
Table 16 – Extreme outliers summary.	84
Table 17 – Summary of mutual Transport Layer Security (mTLS) results.	87
Table 18 – Summary of the authorization results.	87
Table 19 – Comparison between Istio and Linkerd.	87

LIST OF ACRONYMS

API	Application Programming Interface	43
AWS	Amazon Web Services	16
EKS	Elastic Kubernetes Service	38
GCP	Google Cloud Platform	16
GKE	Google Kubernetes Engine	22
IaaS	Infrastructure as a Service	23
mTLS	Mutual Transport Layer Security	29
NIST	National Institute of Standards and Technology	22
PaaS	Platform as a Service	23
RTT	Round-Trip Time	41
SaaS	Software as a Service	23
SMBs	Small and Midsize Businesses	16
TLS	Transport Layer Security	30
UI	User Interface	44
VPN	Virtual Private Network	17
ZT	Zero Trust	16

CONTENTS

1	INTRODUCTION	16
1.1	ZERO TRUST	17
1.2	SERVICE MESHES	18
1.3	MOTIVATION	19
1.4	RESEARCH AIMS	19
1.5	CONTRIBUTIONS	20
1.6	ON THIS RESEARCH	20
2	THEORETICAL FOUNDATION	22
2.1	CLOUD	22
2.1.1	The five essential characteristics	22
2.1.2	The three service models	23
2.1.3	The four deployment models	23
2.2	MICROSERVICES	24
2.2.1	Microservices and scalability	25
2.3	KUBERNETES	25
2.3.1	How it works	26
2.4	ZERO TRUST	27
2.4.1	Zero Trust principles	27
2.4.2	Mutual Transport Layer Security	29
2.5	SERVICE MESHES	29
2.5.1	Data and control planes	30
2.5.2	Istio	30
2.5.2.1	<i>Architecture</i>	31
2.5.2.2	<i>Security</i>	32
2.5.3	Linkerd	33
2.5.3.1	<i>Architecture</i>	34
2.5.3.2	<i>Security</i>	34
2.5.4	Zero Trust on the service meshes	35
2.6	LITERATURE REVIEW	36
2.6.1	Search query and results	36

2.7	RELATED WORKS	38
2.7.1	A Look at Service Meshes	38
2.7.2	Performance Analysis of Zero-Trust multi-cloud	39
2.7.3	Zero Trust Network Security Model in containerized environments	39
2.8	FINAL REMARKS	40
3	METHOD	41
3.1	RESEARCH METHOD	41
3.2	PROJECT	42
3.2.1	Architecture	42
3.2.1.1	<i>Calculator</i>	43
3.2.1.2	<i>Experimenter</i>	43
3.2.1.3	<i>Frontend</i>	44
3.2.1.4	<i>Analyzer</i>	45
3.2.1.5	<i>Security Checker</i>	45
3.2.1.6	<i>Reverse Proxy</i>	45
3.2.2	How to use	46
3.3	SECURITY CONFIGURATIONS	47
3.3.1	Security configurations for Istio	48
3.3.2	Security configurations for Linkerd	49
3.4	FINAL REMARKS	50
4	EXPERIMENTS	52
4.1	EXPERIMENT TYPES	52
4.2	SYSTEM SPECIFICATION	52
4.2.1	Local system specification	52
4.2.2	Cloud system specification	53
4.2.3	Port forwarding	53
4.3	SECURITY TESTS	54
4.3.1	TLS tests	54
4.3.1.1	<i>Istio TLS verification</i>	54
4.3.1.2	<i>Linkerd TLS verification</i>	55
4.3.2	HTTP status tests	55
4.3.2.1	<i>Case 1: No service meshed</i>	56

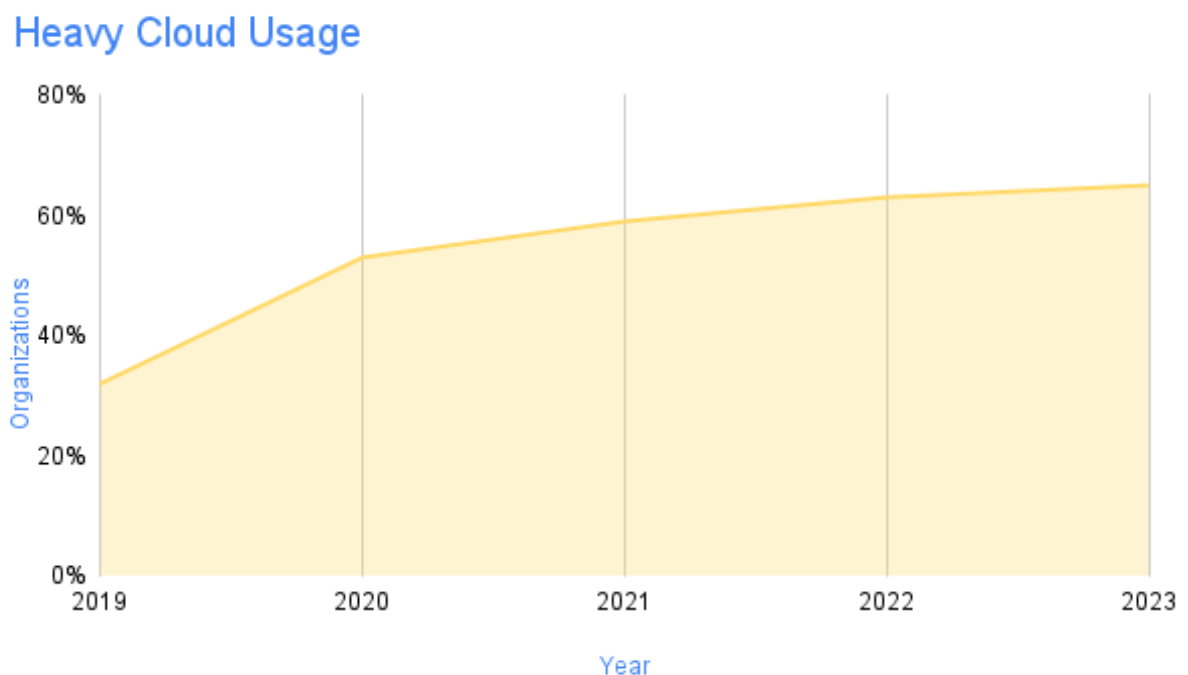
4.3.2.2	<i>Case 2: All services meshed</i>	57
4.3.2.3	<i>Case 3: Calculator not meshed</i>	58
4.3.2.4	<i>Case 4: Security Checker and Experimenter not meshed</i>	59
4.3.2.5	<i>Case 5: External meshed</i>	60
4.3.2.6	<i>Case 6: External not meshed</i>	61
4.4	PERFORMANCE EVALUATION	62
4.4.1	Workload	64
4.5	FINAL REMARKS	65
5	RESULTS	66
5.1	SECURITY RESULTS	66
5.1.1	TLS results	66
5.1.1.1	<i>Istio TLS results</i>	66
5.1.1.2	<i>Linkerd TLS results</i>	67
5.1.1.3	<i>TLS results observations</i>	67
5.1.2	HTTP status results	67
5.1.2.1	<i>Istio HTTP status results</i>	68
5.1.2.2	<i>Linkerd HTTP status results</i>	69
5.1.2.3	<i>Summarizing HTTP status results</i>	71
5.1.2.4	<i>Cloud security differences</i>	71
5.2	PERFORMANCE RESULTS	72
5.2.1	Preliminary results	72
5.2.2	Local results	75
5.2.3	Local results with bigger numbers	77
5.2.4	Cloud results	80
5.2.5	Performance observations	83
5.3	FINAL REMARKS	85
6	CONCLUSION	86
6.1	RESULTS	86
6.2	CONTRIBUTIONS	87
6.3	LIMITATIONS AND FUTURE WORKS	88
6.4	FINAL REMARKS	89
	REFERENCES	90

1 INTRODUCTION

The global pandemic of 2020 has shaken the World. In its wake, nearly every organization was forced to embrace Zero Trust (ZT), as employees went remote and digital transformation became critical to sustainability [Microsoft, 2021].

The move towards a digital transformation is highlighted in the increase in public cloud usage, e.g., Amazon Web Services (AWS) and Google Cloud Platform (GCP), by companies which grew from 32% as heavy users in 2019 to 65% in 2023 [Flexera, 2019, Flexera, 2023]. Figure 1 shows the growth of heavy public cloud usage of organizations from 2019 to 2023 [Flexera, 2019, Flexera, 2020, Flexera, 2021, Flexera, 2022, Flexera, 2023]. Heavy cloud usage is defined as running more than 25% of workloads in public cloud [Flexera, 2022].

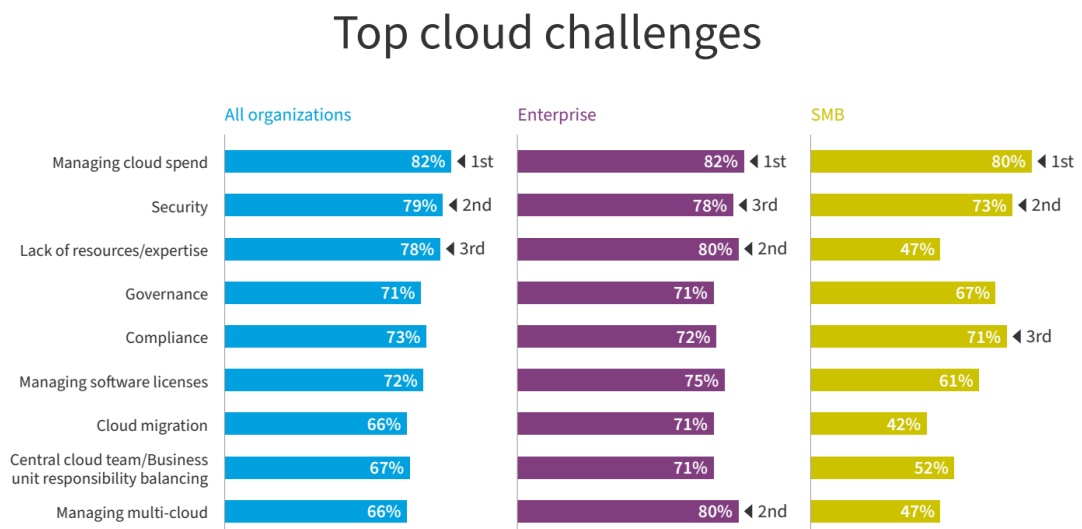
Figure 1 – Heavy cloud usage growth.



Source: Adapted from [Flexera, 2019, Flexera, 2020, Flexera, 2021, Flexera, 2022, Flexera, 2023].

Migration to the cloud brings challenges to organizations, with security as one of the top challenges. Considering Enterprises as organizations with more than one thousand employees and Small and Midsize Businesses (SMBs) as organizations with fewer than one thousand employees, security is one of the top challenges of organizations in the cloud [Flexera, 2023]. Figure 2 presents these challenges.

Figure 2 – Cloud challenges of organizations.



Source: [Flexera, 2023, p. 6].

This research investigates ZT and service meshes on microservice cloud-based applications. It will primarily discuss ZT and service meshes, hence this Chapter will provide an overview of their current state and importance, with the related justifications for the present research. Additionally, the research aims and contributions will be discussed and a brief overview of the incoming chapters will be presented.

1.1 ZERO TRUST

With the rise of cloud usage and security challenges, global ZT investment is projected to grow from USD 19.6 B in 2020 to USD 51.6 B in 2026 [Sermoud et al., 2021]. Zero Trust is a collection of concepts and ideas for enforcing accurate, least-privilege access, per-request decisions to services [Stafford, 2020]. These concepts can be described into three major guidelines [Microsoft, 2021]:

1. Verify explicitly: Always authenticate and authorize.
2. Use least-privilege access: Limit access to the minimum amount required at that time.
3. Assume breach: Minimize blast-radius, encrypt, and monitor.

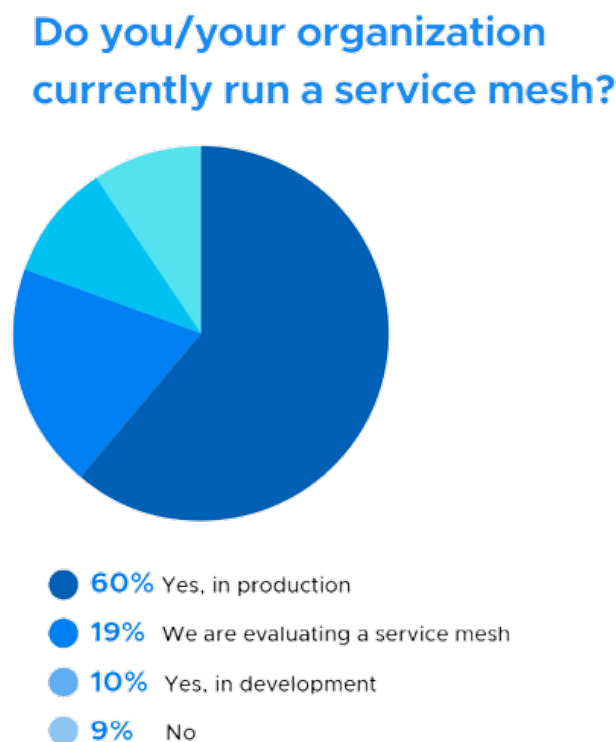
These guidelines offer a base for the decisions and policies to be implemented by security

experts, aiming to reduce risks while also maintaining usability. It goes against the assumptions that requests inside a perimeter, e.g., virtual private network (VPN), should be authorized. With ZT, only the minimum required access to services and resources should be authorized and it should be based on more criteria than the location of the client [Sermoud et al., 2021]. Zero Trust is further discussed in Section 2.4.

1.2 SERVICE MESHES

According to [Cloud Native Computing Foundation, 2022], 70% of respondents use a service mesh in a production or development environment, as presented in Figure 3.

Figure 3 – Service meshes usage.



Source: [Cloud Native Computing Foundation, 2022, p. 2].

The survey interviewed 253 participants from different countries, mostly from Europe and North America, of which 72% were SMBs. It also investigated which service meshes were more used and Linkerd came on top with 73% followed by Istio with 34%.

A service mesh is an infrastructure layer that provides different services to the applications (e.g., network connectivity, network resilience, observability, and security) [Chandramouli and Butcher, 2023]. Service meshes can add a wide range of features to the applications, but

these features might affect application performance, still, they provide these features in a decoupled and uniform manner, which can be ideal for companies looking for a ZT transition. As presented by [Buoyant, 2023]:

The service mesh gives you features that are critical for running modern server-side software in a way that's uniform across your stack and decoupled from application code.

In this scenario, service meshes can be an additional layer to help in applying ZT principles to applications, helping companies be successful in this transition. This research aims to investigate this possibility.

1.3 MOTIVATION

Though Zero Trust is increasing in relevance and service meshes provide means to fulfill this goal [Chandramouli et al., 2021, Chandramouli, 2022, Butcher, 2021, Shoemaker and Estes, 2021, Buoyant, 2023], few works detail how service meshes capabilities are aligned with ZT, how to evaluate the performance effects of applying ZT with a service mesh to an application, and most notably, how service meshes compare to each other.

Hence, additional investigation is required to address these gaps. Thus, this work seeks to deepen the research in this area.

1.4 RESEARCH AIMS

This research is based on the knowledge that security in the cloud is a relevant subject for companies, as disclosed in [Flexera, 2023]. Hence, it intends to understand how service meshes can enable Zero-Trust approaches to service-to-service communication. Furthermore, the following questions are used as guidelines for the research:

1. How can service meshes be applied to enforce Zero Trust principles in service-to-service communication?
2. How does Zero-Trust protection affect service-to-service performance?
3. How do widely adopted service meshes, Istio and Linkerd, compare to each other in terms of security and performance?

While ZT is a subject with approaches to both user and service criteria, this research is limited to the service-to-service view of Zero Trust.

1.5 CONTRIBUTIONS

This work contributes to the field on different fronts. First, it shows how service meshes can be used in alignment with ZT principles. Then, it expands and improves performance and security results found in previous studies, by performing additional experiments, utilizing extra metrics, and describing steps that enable researchers to perform their experiments in security and performance, when using service meshes. Finally, it compares two of the most used service meshes, Istio and Linkerd.

1.6 ON THIS RESEARCH

This Chapter discussed the importance of Zero Trust and service meshes, providing details on why this research is relevant. Additionally, it presented the justification, research aims, and contributions.

It is important to emphasize that the second research question: “How does Zero-Trust protection affect service-to-service performance?” is focused on the effects of service meshes in the network communication performance.

This document is organized into five chapters, including the current one, Introduction.

- Chapter 2 provides the definitions of the most relevant subjects for this research, such as cloud, microservices, service meshes, and Zero Trust. It also presents a literature review for this research and discusses the related works.
- Chapter 3 presents the research method used, the proof-of-concept application built to enable the experiments, the configurations set to the service meshes, and how they align with the specified Zero Trust guidelines.
- Chapter 4 provides additional details on the setup required for performing the experiments. It discusses the workload used, the security and performance experiments, and the goals of said experiments.
- Chapter 5 shows and discusses the results found during the experiments.

- Finally, Chapter 6 provides a review of what is discussed in the research, the synthesis of the answers to the research questions, the limitations of this study, and opportunities for future works.

2 THEORETICAL FOUNDATION

This Chapter defines and details the most relevant subjects to understand this research. Firstly an overview of relevant subjects, such as cloud, microservices, and more will be presented, then an in-depth description of service meshes will be made, and finally, a literature review and related works will be presented.

2.1 CLOUD

As discussed in Chapter 1, cloud computing has become an important topic for technology experts. One of the important questions that experts need to address is which provider to use as there is a wide range of cloud providers, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). This research uses GCP as the cloud provider for its Google Kubernetes Engine (GKE).

It is also important to define the term cloud. In this research, it is considered the definition of cloud provided by the National Institute of Standards and Technology (NIST) is used [Mell and Grance, 2011]:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”

2.1.1 The five essential characteristics

According to [Mell and Grance, 2011], the five essential characteristics of cloud are as follows:

- On-demand self-service: The consumer can unilaterally provision computing resources.
- Broad network access: Capabilities are available over a network and accessed through standard mechanisms.

- Resource pooling: The provider's resources are pooled to serve multiple consumers. The consumer generally does not know the exact location of the provided resources but may be able to specify it at a higher level of abstraction.
- Rapid elasticity: Allocated resources can be provisioned and released as needed.
- Measured service: Resource usage can be monitored and controlled to provide transparency for both the provider and the consumer.

2.1.2 The three service models

According to [Mell and Grance, 2011], the three cloud service models are as follows:

- Software as a Service (SaaS): The capability provided to the consumer is to use the provider's application running on a cloud infrastructure. Additionally, the consumer does not manage or control the underlying infrastructure.
- Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure, using the consumer's applications with tools supported by the provider. The consumer has no control over the cloud infrastructure, but has over the deployed applications and possibly additional settings for the hosting environment.
- Infrastructure as a Service (IaaS): The consumer can provision processing, storage, network, and other fundamental resources, but does not control underlying cloud infrastructure.

2.1.3 The four deployment models

According to [Mell and Grance, 2011], the four cloud deployment models are as follows:

- Private cloud: Infrastructure provisioned by a single organization. It can be on or off premises.
- Community cloud: Infrastructure is provisioned by a specific group of consumers with shared concerns. It can be on or off premises.

- Public cloud: The cloud infrastructure is provisioned for open use and exists on the premises of the cloud provider.
- Hybrid cloud: The cloud infrastructure is a composition of two or more distinct infrastructures (private, community, or public).

For the deployment models, an additional one can be considered: Multi-cloud. Similarly to a hybrid cloud, multi-cloud also considers deployment on more than one infrastructure, however, they can be of the same model, but should be from two different cloud providers, typically two different public cloud providers [Google, 2024].

2.2 MICROSERVICES

Microservices is an architectural style where each service has a single responsibility. According to [Thönes, 2015], a *microservice* is an application that can be deployed, scaled, and tested independently, with a single responsibility. They offer both challenges and advantages when compared to their monolithic counterparts. A monolith is a software application whose modules cannot be executed independently [Dragoni et al., 2017].

A *container* is a building block for a microservice. It can be defined as a lightweight virtualization that can isolate and control resources for a set of processes [Amaral et al., 2015].

Microservices are focused on a single responsibility and thus easier to maintain, but the complexity of the monolithic architecture is moved to the communication and error handling between the services. Concerned with fault tolerance, Leslie Lamport once said [ACM, 2013]:

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”

Microservices take an additional effort in this area when related to the monolithic approach. What was once a function call, can now be a request to a microservice, susceptible to connection errors and status handling issues.

2.2.1 Microservices and scalability

When a system uses a microservice architecture and identifies that a set of services is overloaded with requests while another set is not, it is possible to deploy replicas of the overloaded services while not changing the computing power allocated to the other services. This feat would normally not be possible in a monolithic system as its parts need to be deployed all at once, enabling scalability concerning the entire computing power. It is also important to note differences between microservices architecture and service-oriented architecture, while a service-oriented architecture enables the deployment of additional replicas of services (X-axis scalability) that require additional resources, the services in a microservice architecture can be much smaller and allow for functional decomposition scalability (Y-axis scalability), meaning the addition of replicas for the functions (microservices) that need more resources [Márquez et al., 2018].

Some monolithic applications were later decoupled and deployed as large services, following the service-oriented architecture. Microservices can then be seen as an improvement for service-oriented architecture which emerged due to the high pace needed for changes in the system that were loosely coupled, and highly scalable [Márquez et al., 2018].

Since services in a microservices architecture are by definition, loosely coupled, therefore, they can be easier to maintain, to allocate specific teams for handling the service, add new functions, or cancel the feature without other teams or services being affected by it, thus matching the high pace needed for big projects.

2.3 KUBERNETES

While microservices are small and can be updated and restarted faster than their monolithic counterparts with adequate failure recovery mechanisms, a tool capable of leveraging these benefits is necessary. Kubernetes is a container orchestrator, a platform that enables the automated deployment, scaling, and management of containerized applications [Vayghan et al., 2019], thus working with the advantages of microservices to help support applications.

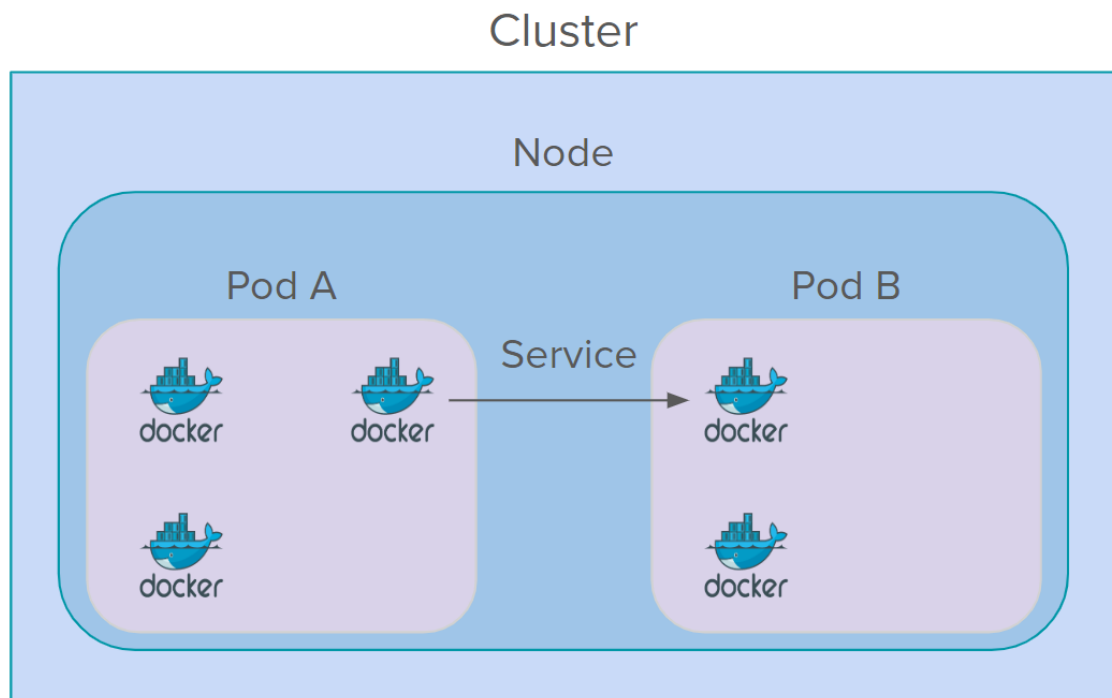
2.3.1 How it works

To deploy an application in Kubernetes, the application must be defined based on Kubernetes elements, these elements are presented in Figure 4.

- A *workload* is an application running on Kubernetes. The workload can be composed of different parts that work together, called containers. These containers use images from services such as Docker.
- Kubernetes deploys containers inside pods. A *pod* is a group of one or more containers with shared storage and network resources. Pods are described as the atomic entity of Kubernetes.
- Kubernetes deploys pods into nodes, a physical or virtual machine depending on the cluster. *Nodes* are part of a *cluster* which is where the workload is deployed.
- Kubernetes also provides *namespaces*, which enable the separation of elements inside a cluster. This means a namespace can have multiple pods in different nodes but in the same cluster.
- As a distributed application, communication between the components is essential. *Services* are an abstract way to expose an application running on a set of pods as a network service. It is not required to create a service for communicating between containers within the same pod as they share the network. Still, it is with services that communication between pods becomes possible, even if they are in different namespaces.

Before moving an application to the cloud to use Kubernetes and paying additional fees, it is often important to run tests locally, e.g., to check if the pods are well-defined or if the communication between the services is working properly. Minikube is a Kubernetes distribution that can help to create a Kubernetes instance on the local machine. It can be run on different operating systems, e.g., Linux, Windows, and MacOS [Sysdig, 2023]. This research uses Minikube for these purposes.

Figure 4 – Kubernetes architecture by example.



Source: The author.

2.4 ZERO TRUST

Zero Trust can be defined as a collection of concepts and ideas for enforcing accurate, least-privilege access, per-request decisions to services [Stafford, 2020].

Zero Trust criticizes the assumption that implicit trust should be granted, based solely on physical or network location [Stafford, 2020]. That is the idea behind virtual private network (VPN) access, perimeters with access authorized or not. Zero Trust requires explicit verification, context-based authorization, least-privilege access, credentials, and more to protect the services.

2.4.1 Zero Trust principles

There are different definitions for the Zero Trust principles, though they can overlap. It is possible to see ten guidelines for Zero Trust, based on [Sermoud et al., 2021]:

1. Above everything, assume breach.
2. No one is trustworthy.

3. Do not authorize access before verifying.
4. Do not base access on IP.
5. Establish access based on context.
6. Take care of your local and remote resources.
7. Authenticate, authorize, and then allow access.
8. Do not change your network, focus on your user.
9. Monitor user access.
10. Do not provide additional privileges than required.

Though this definition provides explicit guidelines, it is also possible to see more compact principles that can include multiple principles mentioned by [Sermoud et al., 2021]. One of these definitions is provided by [Microsoft, 2021]. It defines it with three guidelines, which will be followed in this research:

1. Verify explicitly: Always authenticate and authorize.
2. Use least-privilege access: Limit access to the minimum amount required at that time.
3. Assume breach: Minimize blast-radius, encrypt, and monitor.

It is possible to include some of the principles provided by [Sermoud et al., 2021] by the definition provided by [Microsoft, 2021]:

1. Verify explicitly: Can include principles 3, 7.
2. Use least-privilege access: Can include principles 5, 8 and 10
3. Assume breach: Can include principles 1, 2 4, 6 and 9.

This encapsulation considers the interpretation of the author of the present research, but different interpretations can be perceived for the alignment between the definitions of the ZT principles. Additionally, more guidelines can be made for what principles ZT should follow, but they can be included in the three provided by [Microsoft, 2021].

It is then possible to verify a focus on authentication, authorization, and monitoring when considering the ZT architecture. This research especially considers the authentication and authorization in ZT.

Additional details on the strategies employed by this research for aligning with the ZT principles are provided in Chapter 3.

2.4.2 Mutual Transport Layer Security

Mutual Transport Layer Security (mTLS) uses encryption and authentication, and both client and server authenticate each other. It can help protect applications against different types of security attacks as described in [Koschel et al., 2021]. As ZT states that encrypting the communication between the services is crucial to increase security, this research uses the mTLS provided by the service meshes to achieve the encryption goal.

2.5 SERVICE MESHES

Nowadays, many applications are deployed using a microservices architecture, with the business logic split into different services. These services might need to communicate with each other and additional capabilities are usually required, such as security, observability, authentication, and more [The Istio Authors, 2023a]. *Service meshes* facilitate the addition of these features to microservices communication by the usage of a sidecar proxy, which performs all inbound and outbound network communication [Ashok et al., 2021].

A service mesh is thus an infrastructure layer that can be added to applications to grant these different capabilities in a uniform and decoupled way. The application code and the service mesh configuration can change without affecting one another, unlike adding libraries to provide this feature directly to the application code [Buoyant, 2023].

Though conceptually, service meshes are not restricted to a specific orchestrator (e.g., Kubernetes), both Istio and Linkerd require a Kubernetes cluster.

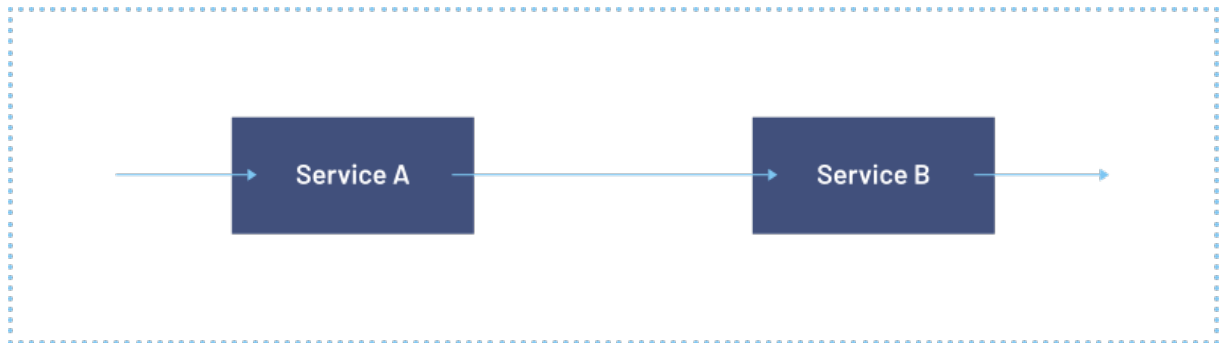
It is important to emphasize that while service meshes can provide important features for running services, they share the resources with the services and are expected to affect overall performance and add complexity. Thus, there are scenarios where a service mesh might not be the best solution, e.g., small projects, and time-sensitive applications such as trading and

online multiplayer gaming.

2.5.1 Data and control planes

Without a service mesh such as Istio or Linkerd, the communication between services often occurs directly, as displayed in Figure 5. This creates issues when adding authentication to the communication between the services, load-balancing, or monitoring is required.

Figure 5 – Service to service communication without Istio.



Source: [The Istio Authors, 2023a].

Service meshes can typically be divided into their data and control planes. The data plane consists of proxies deployed alongside the services that intercept the requests and perform additional operations (request certificate to the control plane). As for the control plane, it coordinates the proxies and also offers service discovery, Transport Layer Security (TLS) certificate issuing, metrics aggregation, and more [Buoyant, 2023].

Service meshes face some challenges, as adding the proxies to the service increases the number of interactions per request and shares CPU and memory resources with these services.

There are different service meshes, each with its particularities and possible configurations. In this project, two of the most commonly used service meshes were studied: Istio [The Istio Authors, 2023b] and Linkerd [Linkerd Authors, 2023d]. Both Istio and Linkerd can be configured to change their behaviors by *.yaml* files, similarly to Kubernetes.

2.5.2 Istio

Istio is an open-source service mesh that can be run on Kubernetes, launched in 2017, with load balancing, service-to-service authentication, and monitoring capabilities. It is the most used of the service meshes according to [Ashok et al., 2021] and the second according

to [Cloud Native Computing Foundation, 2022], and has features that include but are not limited to:

- Security to service-to-service communication with TLS encryption, identity-based communication, and authorization.
- Load balancing over different protocols, such as HTTP and TCP.
- Control of traffic behavior with routing rules and retries.
- Metrics, logs, and traces for all traffic within the cluster (e.g., Prometheus [The Istio Authors, 2023c]).

Istio can also connect to other endpoints running outside the cluster. It can also be used in a multi-cloud application, as shown by [Rodigari et al., 2021].

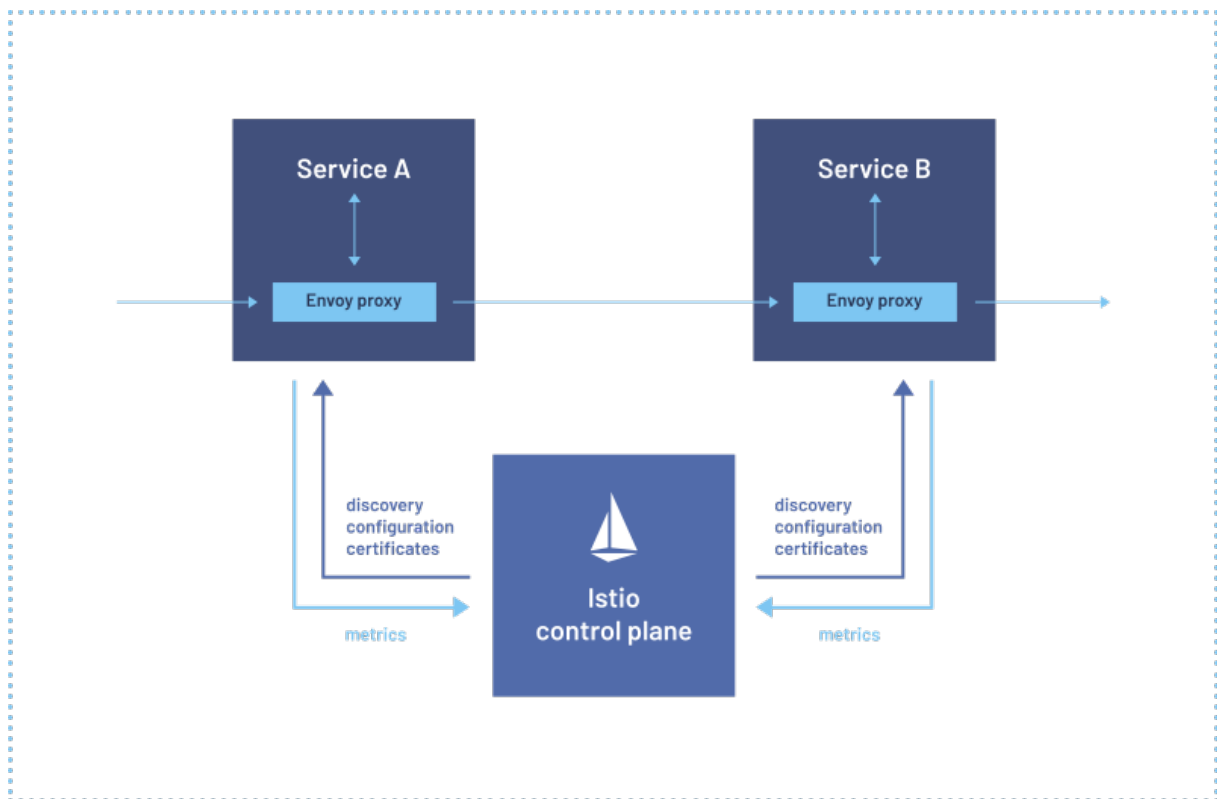
2.5.2.1 *Architecture*

By applying the services to an Istio service mesh, which adds the Envoy proxy to the pods, the communication changes to the diagram displayed in Figure 6. It is possible to see the two components of Istio: the data plane and the control plane.

- The data plane is the communication between the services. Istio uses a proxy to intercept all network traffic, allowing for decisions based on the configuration settings. An Envoy proxy is deployed along with each service in the cluster.
- The control plane uses the defined configuration and its view of the services to configure the proxies, updating them as necessary, dynamically.

Envoy is a high-performance C++ distributed proxy [Envoy Project Authors, 2024], designed for single-service applications. It runs side-by-side with applications, to add capabilities such as load-balancing, observability, and applying network abstraction that is platform agnostic.

Figure 6 – General architecture of the Istio mesh.



Source: [The Istio Authors, 2023a].

2.5.2.2 Security

Security with Istio is focused on 3 main challenges that microservices must address [The Istio Authors, 2023d]:

1. Protection against man-in-the-middle attacks. Istio addresses this problem with traffic encryption.
2. Flexible service access control. Fine-grained access policies are how Istio tries to solve this problem.
3. Monitoring to audit what service accessed what resource at what time. Istio offers easy integration with monitoring tools and standard monitoring on its own.

Istio authentication and authorization explain how it tries to solve security challenges.

- Authentication: Istio offers authentication for both service-to-service and end-user communication. This work considers only service-to-service communication and thus its authentication. Mutual Transport Layer Security can be enforced, including strong identities

for the services. The control plane has its certificate authority for key certificate management, using X.509 certificates. Keys and certificates are requested automatically by an Istio agent that runs alongside the Envoy proxy. It is important to note that identities are available on the certificates, but service names are made available by a DNS service included in Istio.

All traffic from a client is re-routed to the Envoy proxy which performs the TLS handshake with the server's Envoy proxy and verifies if the identity of that service is allowed for running the server. Then the traffic is forwarded to the Envoy proxy of the server that authorizes the request and, if authorized, forwards it to the back-end service.

- Authorization: Istio offers a wide range of possibilities for configuring authorizations. It is possible to set a policy to mesh, namespace, and workload-wide levels. These policies can allow or deny access from a service to other services. It is also possible to specify finer-grained policies with Istio, for access based on HTTP request method (*e.g.*, *GET* and *POST*) and endpoint-wise policies. The authorization is natively done on Envoy, displaying high performance.

2.5.3 Linkerd

Linkerd is an open-source service mesh to be run on Kubernetes launched in 2016. It is known as the first service mesh launched publicly [Buoyant, 2023] and it is also the most used service mesh [Cloud Native Computing Foundation, 2022]. It offers features for debugging, observability, reliability, and security.

Similarly to the general service mesh conception, Linkerd has both the data and control plane. The control plane can be added to the cluster and then it is possible to inject the data plane into the workloads.

A noticeable detail from Linkerd is that it is focused on being lightweight, and as such, many additional features can be added by installing add-ons, reducing the introduction of excessive latency by the default implementation. In this regard it is possible to see a key difference from Istio: Istio uses Envoy as a proxy for the data plane while the engineers of Linkerd opted to build their proxy called Linkerd2-proxy which is specific for Linkerd. This proxy is lightweight, less complex, and more secure when compared to Envoy [Linkerd Authors, 2023c].

2.5.3.1 Architecture

The control plane has several components that help to manage the service mesh:

- The destination service is used for service discovery information, even related to TLS, to fetch policy information, retries, timeouts, and more.
- The identity service acts as a Certificate Authority that offers the TLS certificates for proxy-to-proxy connections with mTLS.
- The proxy injector acts by adding the Linkerd proxies to the pods.

The data plane consists of the micro-proxies deployed as sidecar containers to the application pods. These proxies intercept TCP connections to and from the pods and follow the rules enforced by the control plane (e.g., applying mTLS, metrics monitoring, and load balancing).

Linkerd also offers a command line interface (CLI) to enable users to interact with control and data planes. It is possible to see Linkerd's architecture in Figure 7.

2.5.3.2 Security

Linkerd solves security challenges like Istio tries to solve them.

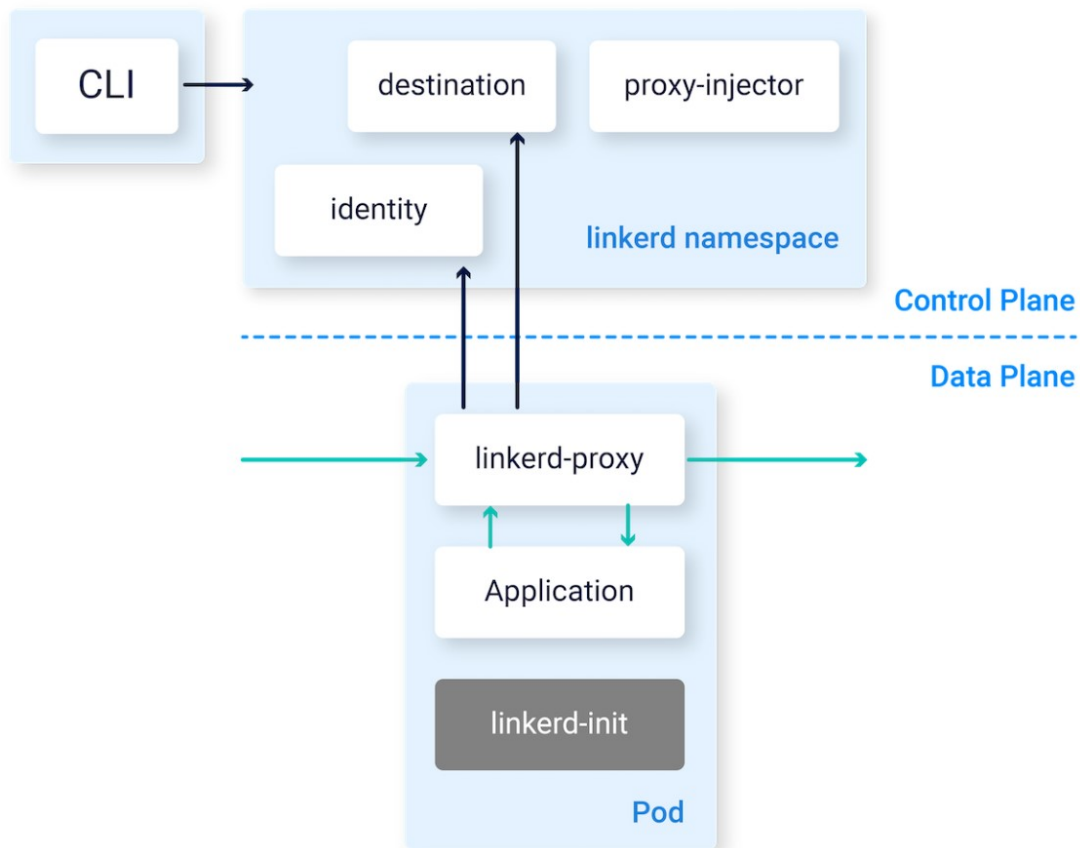
1. Traffic encryption with mTLS.
2. Fine-grained access policies to the services.
3. Metrics and logs capture as well as integration with monitoring tools.

For authentication, Linkerd can be configured to enforce mTLS to proxy-to-proxy communication, ensuring that all requests leaving the pods can be secured without requiring code changes. The communication between the data and control planes is also secured by mTLS. The control plane acts as a certificate authority and issues TLS certificates for the proxies.

For authorization, Linkerd can use policies at the namespace, workload, or pod level. Valid policies are:

- all-unauthenticated: allow all requests. This is the default configuration.
- all-authenticated: allow requests from meshed clients only.

Figure 7 – General architecture of the Linkerd mesh.



Source: [Linkerd Authors, 2023a].

- cluster-authenticated: allow requests from meshed clients in the same cluster.
- deny: deny all requests.

Linkerd also offers finer-grained policies (e.g., route or port-level policies).

2.5.4 Zero Trust on the service meshes

Having these security features for authentication, encryption, authorization, and more, it explains how Istio and Linkerd can be used as ZT solutions to service-to-service communication.

- Verify explicitly: Strong identities for the services and static and dynamic authorization policies can enforce that all requests are fully authenticated and authorized.
- Use least-privilege access: The principle least privilege can also be considered when setting the authorization policies for the services, enabling only the access needed for

each of the services.

- Assume breach: Mutual Transport Layer Security integrated with monitoring tools may facilitate the identification and protection against breaches.

2.6 LITERATURE REVIEW

This research included a literature review to understand the state of the art involving service meshes and Zero Trust. This Section aims to explain this review as well as relevant findings. Furthermore, some of the publications found during this review presented connected ideas to this research and are further detailed in Section 2.7.

2.6.1 Search query and results

The literature review employed the following specifications:

- Search string: *kubernetes AND cloud AND “zero trust” AND (istio OR linkerd) AND “service mesh” AND -iot.*
- Search mechanism: Google Scholar.
- Date period: No limiting starting date and limiting updates up to October 2023.

The search string requires the results to mention kubernetes, cloud, zero trust, and service mesh on it, to have istio or linkerd while excluding iot (Internet of Things). Internet of Things was excluded from the query as a delimitation on the cloud, focused on service meshes, was needed. Including IOT in the results could result in academic works outside the scope of the present research. The search produced the following results which were then filtered:

- Initial results: 52.
- Exclusion criteria:
 - Content must be publicly available for free, not requiring a specific network or access.
 - Content must be in English.

- Content must be in PDF format.
 - Content must be a scientific paper, article, or white paper.
- Results after filtering: 14.

It is noteworthy that out of the fifty-two results, forty-three were published from 2021 to October 2023, indicating that service meshes and ZT are increasing in relevance and are evolving fields. This is more prominent when it is considered that ZT originated in 2009-2010 [Kindervag et al., 2010] and service meshes in 2016 with Linkerd [Buoyant, 2023].

Of the fourteen results, two works [Koschel et al., 2021], [Rodigari et al., 2021] are more relevant to this research. These works will be discussed in Section 2.7, as they provide insights into implementing and evaluating service meshes.

The literature indicates that service mesh is a popular approach to implementing ZT. Research conducted for NIST [Chandramouli et al., 2021, Chandramouli, 2022] has detailed the advantages and use cases for service meshes in implementing ZT, such as the separation between configuration and deployment from the application to the security. Other academic works [Butcher, 2021], [Shoemaker and Estes, 2021] also highlight service meshes as the current solution for applying ZT to microservice-based applications.

There is limited research on verifying configurations and performance of service meshes for ZT. Additionally, none of the reviewed research compares different service mesh tools. Table 1 displays all the academic works found during the literature review, ordered by their title.

Table 1 – Academic works found in the literature review.

Title	Year	Strongly Related
A look at service meshes	2021	Yes
Accessible Telemetry Streams using a Zero Trust Architecture for the Flight Operations Directorate	2021	No
Acila: attaching identities of workloads for efficient packet classification in a cloud data center network	2022	No
An Enhanced CICD Pipeline: A DevSecOps Approach	2023	No
Application Based Access Control for Remote RDBMS	2022	No
Attribute-based access control for microservices-based applications using a service mesh	2021	No
DoD Learning Enclave: Realizing the Defense-wide Learning Ecosystem	2022	No
Implementation of DevSecOps for a Microservices-based Application with Service Mesh	2022	No
KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments	2023	No
Performance Analysis of Zero-Trust multi-cloud	2021	Yes
Security Technologies and Research Challenges on Microservice-Based NFV.	2020	No
SoK: Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices	2022	No
Towards secure and leak-free workflows using microservice isolation	2021	No
Zero Trust Architecture	2021	No

Source: The author.

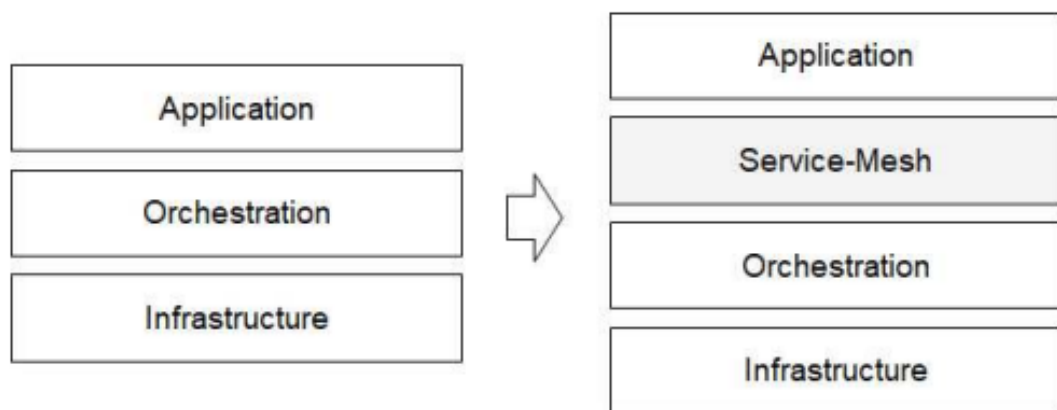
2.7 RELATED WORKS

This Section aims to discuss the research found to be strongly aligned with this one. Out of the fourteen results considered in Section 2.6, two are considered as related works: [Koschel et al., 2021] and [Rodigari et al., 2021]. Though not found during the literature review, a third was encountered additionally: [de Weever and Andreou, 2020].

2.7.1 A Look at Service Meshes

The research conducted by [Koschel et al., 2021] provides an elaborate description of service meshes, how they work, and how they can be configured. It describes the service mesh as a layer between orchestration (Kubernetes) and the application, as displayed in Figure 8. It discusses the architecture of Istio in further detail than delivered by this research, even providing code snippets that may allow readers to create their configurations for Istio.

Figure 8 – Service mesh layer between application and orchestration.



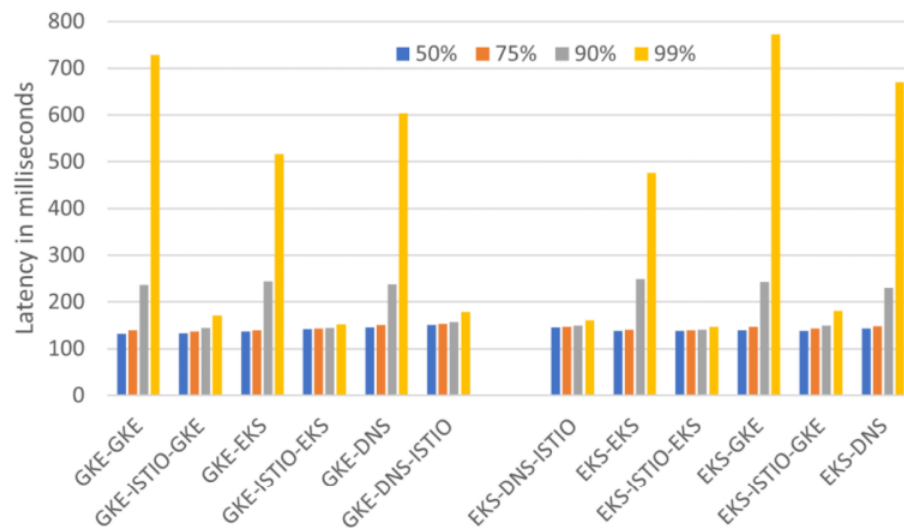
Source: [Koschel et al., 2021, p. 2].

It briefly touches upon ZT concerning service meshes, but it does not offer any comparisons between different service meshes. Moreover, the authors do not elaborate on the relationship between ZT and service mesh capabilities. Additionally, the article does not provide any guidelines on how to verify the security configurations and evaluate the performance.

2.7.2 Performance Analysis of Zero-Trust multi-cloud

The study conducted by [Rodigari et al., 2021] used Istio in a multi-cloud environment with Kubernetes on Google Kubernetes Engine (GKE) from Google Cloud Platform (GCP) and Elastic Kubernetes Service (EKS) from Amazon Web Services (AWS). They used a DNS load balancer to route traffic between the two Kubernetes clusters. The performance indicators considered were CPU and memory usage, and latency. The study found that Istio produced a lower latency variability, as shown in Figure 9. However, it was observed that CPU and memory usage may increase.

Figure 9 – Latency results with Istio in a multi-cloud environment.



Source: [Rodigari et al., 2021, p. 3].

It does not compare different service meshes and how they perform with various security configurations. It also fails to address how these configurations align with ZT or how to verify that they are functioning as intended. Additionally, the article states that further performance evaluations are necessary.

2.7.3 Zero Trust Network Security Model in containerized environments

de Weever and Andreou [de Weever and Andreou, 2020] discuss the importance of communication encryption and monitoring in containerized environments. Their study proposes the integration of two tools, Cilium and Hubble, to enhance the security and visibility of meshed

services in Istio. The researchers also suggest the use of flags to verify the application of mTLS and propose sniffing the communication to ensure that it is encrypted.

de Weever and Andreou offer guidance on verifying mTLS but do not compare service meshes or assess ZT impact on performance.

2.8 FINAL REMARKS

This Chapter has presented the necessary definitions, especially regarding Zero Trust and service meshes. It provided details on a literature review and the related works to this research. It showed that Zero Trust is an evolving subject with limited research on service meshes. Furthermore, details on how service meshes work and how they can be used to align with Zero Trust guidelines were discussed.

Though this research focuses on the details regarding descriptions for service-to-service uses, ZT and service meshes also can be used to protect client-to-service communication. The different policies described to protect services for service-to-service can also be used for client-to-service while using authentication mechanisms integrated with the service meshes, rather than service credentials for a user.

3 METHOD

This Chapter details the method undertaken for this research and the necessary tools employed to accomplish it. The research method will be presented, and then Section 3.2 will provide information on the proof-of-concept application created to study the research questions. In addition, Section 3.3 will discuss the security configurations employed in the study.

Additionally, the code used for this project is available on [Lucas Costa, 2024].

3.1 RESEARCH METHOD

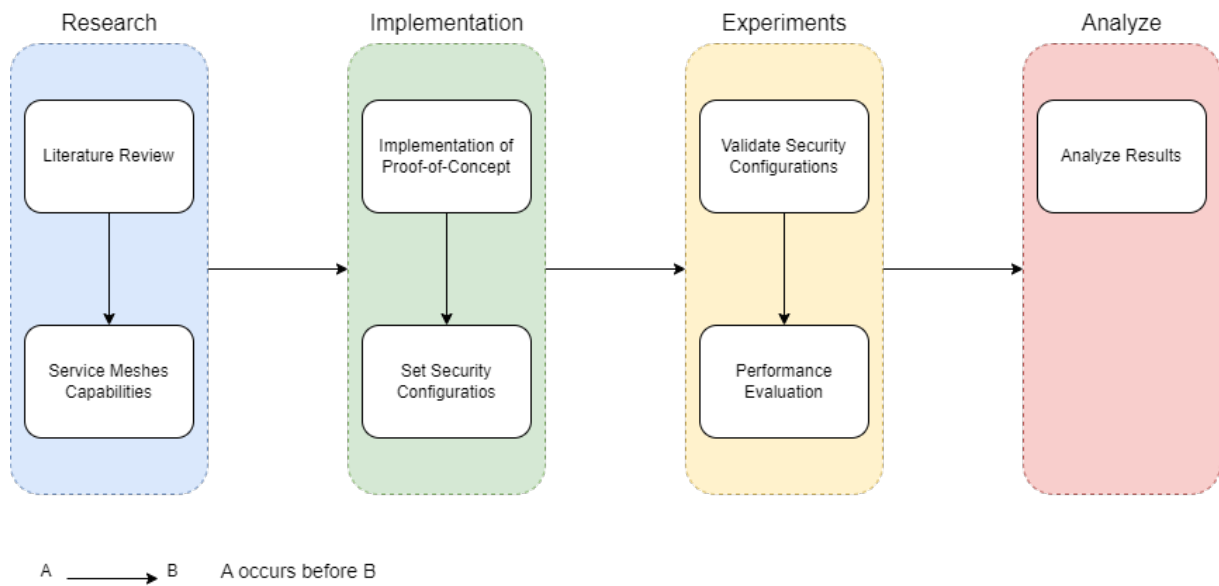
This study is focused on applied research and uses experimental research to investigate the questions presented in Section 1.4.

As a proof-of-concept for this work, an application was developed, detailed in Section 3.2, and evaluated using the metric round-trip time (RTT) to understand how Zero Trust (ZT) protection affects application performance. A similar approach for the evaluation, but with distinct metrics and focus, was used by [Rodigari et al., 2021]. Though the experiments are part of the applied method, further details on the preparation for the experiments and the experiments themselves are available in Chapter 4 for an isolated view.

This research compares two service meshes, Istio and Linkerd. This additional investigation is relevant for understanding compromises and advantages for each of the service meshes while also developing knowledge in this field.

An overview of the research method is displayed in Figure 10.

Figure 10 – Research method overview.



Source: The author.

3.2 PROJECT

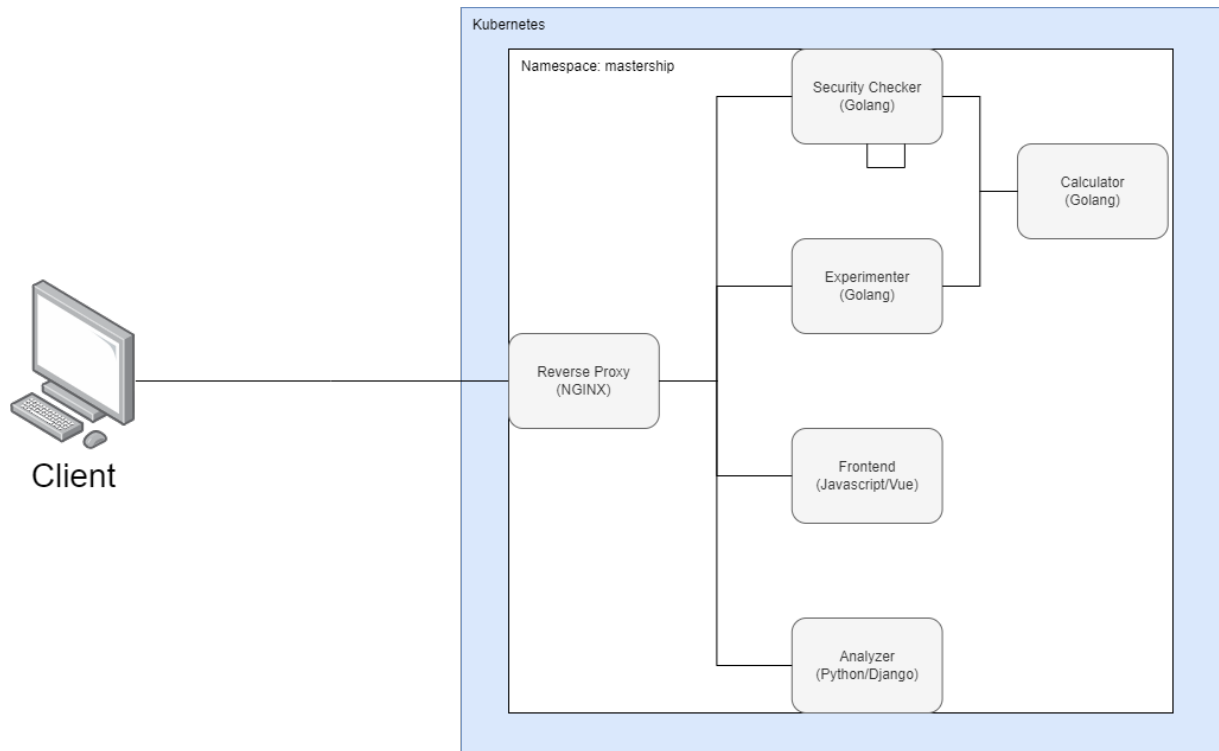
The proof-of-concept project developed offers a user interface where it is possible to request the execution of an experiment with a set of parameters that can be selected on run time. The execution returns all RTTs in milliseconds, the mean and median RTTs, and the standard deviation inferred from them, as well as the RTT90, RTT95, and RTT99. RTT is the time it takes for the Experimenter to get the response from a request to the Calculator service.

This Section explains the architecture and the features of the proof-of-concept application.

3.2.1 Architecture

The application is split into different services, each responsible for a single step. The services were built on languages like Golang, Python, and JavaScript. They are containerized with Docker and then specified into Kubernetes elements. Figure 11 shows how the services are organized and how they interact.

Figure 11 – Project architecture.



Source: The author.

3.2.1.1 Calculator

This service offers four endpoints, each with a different mathematical operation: sum, subtract, multiply, and divide. The caller sends two numbers to this service on any endpoint to get back the result of the operation. It serves as a representative operation to be called by the Experimenter service to verify the time it takes to obtain the response.

The Calculator service is not exposed to external access. It can only be reached through another service in the network. This service was written in Golang for its performance and application programming interface (API) simplicity.

3.2.1.2 Experimenter

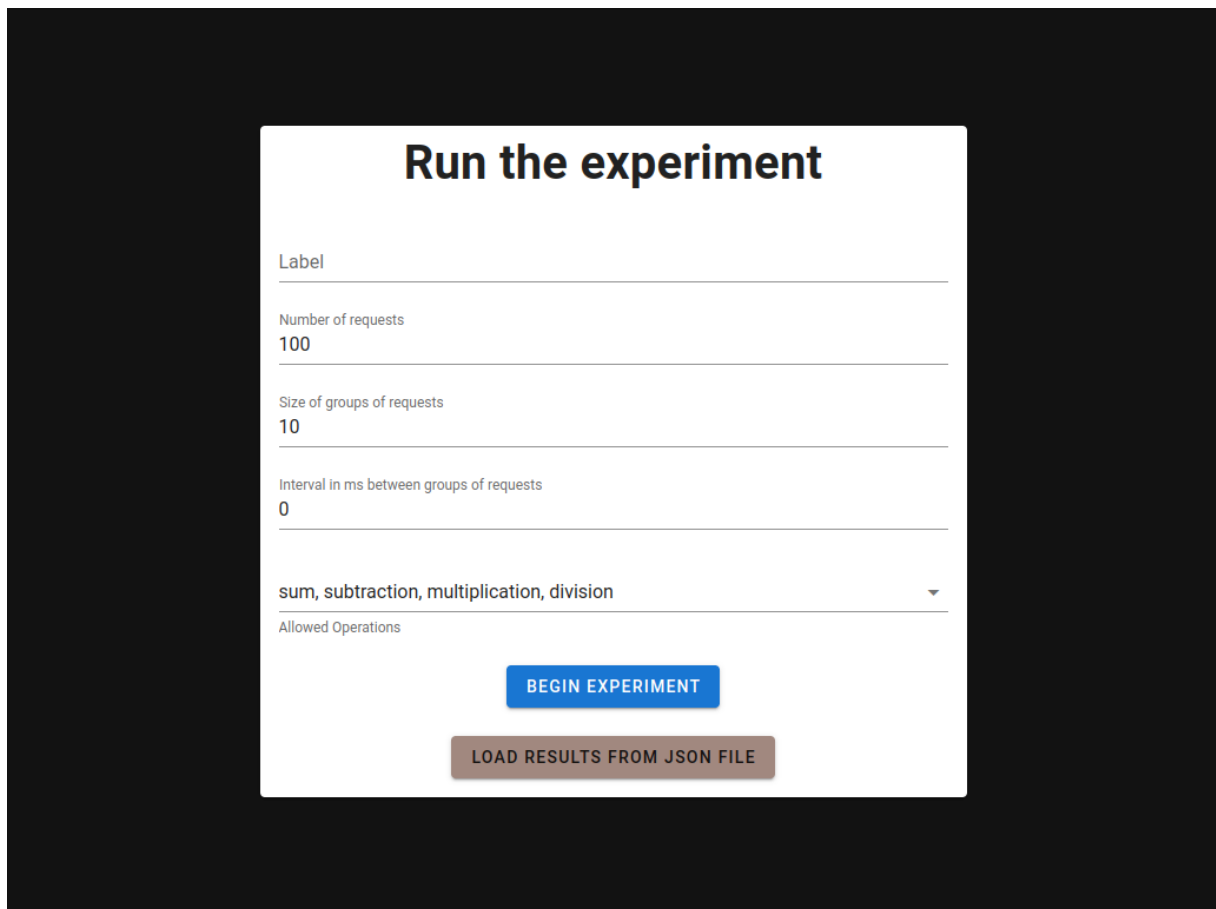
The Experimenter service is the core part of the application. It is the key to the execution of an experiment. It receives the request with the desired parameters from the client and experiments accordingly. It makes sequential requests to the Calculator service grouped by the specified amount and adds a time interval between each group. The service then holds the RTTs and the number of failures, if any, and will then calculate the mean, median, and

standard deviation values to return them to the caller. This service was also implemented using Golang.

3.2.1.3 Frontend

As a means to facilitate the usage of the application, the Frontend service was developed. It offers a user interface that facilitates setting the parameters required to conduct the experiments. It also enables the user of the proof-of-concept to download the results as a JSON file, call the analysis, and use the results of previous experiments as input. The Frontend service enables access to all exposed features of the project through the user interface (UI), except the Security Checker. The home page can be seen in Figure 12. This service was developed with JavaScript, Vue, and Vuetify as frameworks.

Figure 12 – Front end's user interface.



Run the experiment

Label

Number of requests
100

Size of groups of requests
10

Interval in ms between groups of requests
0

sum, subtraction, multiplication, division

Allowed Operations

BEGIN EXPERIMENT

LOAD RESULTS FROM JSON FILE

Source: The author.

3.2.1.4 *Analyzer*

This service extracts additional metrics from the results of an experiment. It can find the RTT90, RTT95, and RTT99 from an experiment run, meaning the RTTs where 90, 95, and 99% of the values are below it, respectively. It could be developed further to generate graphs and apply statistical tests to the results.

This service was written in Python with Django REST as a framework for API building. Python is a great programming language for applying statistical metrics. It has extensive and well-established libraries that can help with this implementation.

3.2.1.5 *Security Checker*

The Security Checker was built to facilitate the security tests specified in Section 4.3. It was also implemented in Golang. It can perform requests to two services: Calculator service and Security Checker, in the same or a different namespace. It is thus used to verify if access to a service is authorized in the security tests.

It calls itself to verify if a request is authorized to a service where the destination service is not the Calculator. This research added specific authorization policies for the Calculator service, which could deny access to it, but the experiments need to verify if a request would be authorized without the authorization policy specific to the Calculator, thus the Security Checker can perform requests to itself, which is necessary to validate security scenarios.

3.2.1.6 *Reverse Proxy*

A reverse proxy was applied to enable a single entry point, optimizing speed and communication between the external world and the services inside the cluster. All external access to the services must go through the reverse proxy service which then forwards the requests to the Experimenter, Analyzer, Security Checker, and Frontend services, depending on the endpoint called. This service uses NGINX for reverse proxy.

3.2.2 How to use

This section presents how to use the proof-of-concept application for performance experiments and for storing and loading results.

1. With the application running, head to its URL address.
2. Add the desired parameters for the experiment and then press the “BEGIN EXPERIMENT” button.
 - a) Alternatively, it is also possible to press the “LOAD RESULTS FROM JSON FILE” button to use the results and parameters from a previous execution from a JSON file.
 - b) The Figure 13 displays the result of the execution or loading of an experiment.
 - c) With the results of an experiment, it is possible to download these results as a JSON file by clicking on the “DOWNLOAD RESULTS” button.
3. To run the analysis, click on the “ANALYZE” button.
 - a) After running the analysis, the results will be downloaded as a CSV file with only the additional information the analysis found.

Figure 13 – Illustration of the results UI.

Analyze the results

Label
Example

Number of requests
100

Size of groups of requests
10

Interval in ms between groups of requests
0

sum, subtraction, multiplication, division

Allowed Operations
Mean in Milliseconds
0.0791099999999997

Median in Milliseconds
0.0655

Standard deviation in Milliseconds
0.11474013203757434

Total number of request failures on test
0

RTTS in Milliseconds
1.209,0.115,0.067,0.067,0.058,0.139,0.156,0.075,0.07,0.061,0.066,0.067,0.088,0.066,0.067,0.088,0.064,0.065,0.058,0.048,0.07,0.053,0.058,0.067,0.061,0.058,0.052,0.06,0.056,0.064,0.06,0.119,0.06,0.065,0.054,0.057,0.071,0.059,0.051,0.061,0.067,0.069,0.057,0.053,0.066,0.058,0.085,0.068,0.06,0.067,0.059,0.062,0.073,0.068,0.057,0.059,0.075,0.059,0.052,0.083,0.064,0.064,0.074,0.056,0.0

ANALYZE

DOWNLOAD RESULTS

CANCEL

Source: The author.

3.3 SECURITY CONFIGURATIONS

As explained in Section 2.4, ZT is a broad subject with different approaches and guidelines for protection. Thus, this research considered three keys to ZT:

1. Verify explicitly:

- Authorize access based on service credentials.

2. Use least-privilege access:

- Allow only the Experimenter service to access the Calculator service.

3. Assume breach:

- Encrypt communication between meshed services.
- Restrict access to meshed services.

While both Istio and Linkerd offer additions to security in their default configuration, additional improvements were required by this research. Istio and Linkerd use credentials managed by their controller for communication between the services, thus fulfilling the first item. They use mutual Transport Layer Security (mTLS) as a standard for communication between meshed services, though they do not enforce it. It means that services out of the mesh can access meshed services. Finally, they do not block access to services based on least privilege, as it is application- and project-specific logic.

Based on these notes, additional security configurations were required, but they are available by the service meshes. Sections 3.3.1 and 3.3.2 will present the settings added to the service meshes to accomplish the security goals.

3.3.1 Security configurations for Istio

For Istio, the security improvements include:

- Peer authentication with *STRICT* mode for the namespace, enforcing communication to meshed services to come exclusively from other meshed services and using mTLS.
- Two authorization policies were specified:
 - Deny access coming from other namespaces.
 - Authorize only the Experimenter service to access the Calculator service. The service account for the Experimenter is used for identifying that the service is the Experimenter for this policy.

Peer authentication is a definition used by Istio to dictate how to handle traffic in the sidecar proxy added to the pods. Specifically, it is possible to use it to block requests coming from not-meshed services and ensure mTLS usage.

Authorization policies are used for authorizing or blocking access to services in the mesh. With these additional settings, Istio fulfills the desired Zero Trust guidelines:

1. Verify explicitly

- Authorize access based on service credentials: The controller generates credentials for each service, and the proxies use it for authentication when communicating to another meshed service, as default.

2. Assume breach

- Encrypt communication between meshed services: Istio adds mTLS for communication between meshed services as default.
- Restrict access to meshed services: The peer authentication and authorization policy enforces access to meshed services to come from meshed services in the same namespace.

3. Use least-privilege access

- Allow only the Experimenter service to access the Calculator service: The authorization policy authorizes only the Experimenter to communicate with the Calculator.

3.3.2 Security configurations for Linkerd

For Linkerd, the security configurations include:

- Default inbound policy for the namespace set as “cluster-authenticated”. It ensures communication to meshed services comes exclusively from other meshed services, using mTLS, and the origin service should be in the same cluster.
- A server and its related authorization policy for the Calculator service, used to authorize only the Experimenter service to access the Calculator service. Similarly to the settings for Istio, the service account for the Experimenter is used for identifying that the service is the Experimenter for this policy.

Linkerd uses authorization policies for authorizing or blocking access to services. It offers two types of authorization policies: Default policies and fine-grained policies. The default

policies work from namespace to pod level and with fewer options for managing access, while the fine-grained policies can bypass the default policies and can go up to ports, routes, and more. Fine-grained policies can block access to a service if it does not originate in a specific service.

A server is one way to create a reference to a meshed service that can then be referenced by a fine-grained policy to achieve the desired access control.

With these additional settings, Linkerd fulfills the desired Zero Trust guidelines as follows:

1. Verify explicitly

- Authorize access based on service credentials: The controller generates credentials for each service, and the proxies use it for authentication when communicating to another meshed service, as default.

2. Assume breach

- Encrypt communication between meshed services: Linkerd adds mTLS for communication between meshed services as default.
- Restrict access to meshed services: The default inbound policy enforces access to meshed services to come from meshed services in the same cluster.

3. Use least-privilege access

- Allow only the Experimenter service to access the Calculator service: The authorization policy for the server authorizes only the Experimenter to communicate with the Calculator.

3.4 FINAL REMARKS

This Chapter has presented the method for this research and an overview of the architecture implemented to enable the experiments. It also discussed the security configurations added to Istio and Linkerd and how they relate to Zero-Trust security.

The security configurations added to Istio and Linkerd do not enforce least-privilege access to all services, only to the Calculator service. The reason is that this research seeks to understand how service meshes can be applied to enforce Zero Trust principles in service-to-service communication and their impact on performance. Thus, this research can accomplish it

with the added configurations, not requiring the additional settings for duplicating a behavior already verified in a specific service, as is shown in Chapter 4. Still, for an application in production, it would be advisable to include the authorization policies that enforce least-privilege access to all the services.

It is relevant to highlight that Istio is configured to block access from other namespaces while Linkerd is configured to deny access from other clusters. It does not imply that it is impossible to offer the same level of protection for both, but due to time constraints in this research, the settings differ in their intended protection.

While this Chapter presented an overview of the architecture and details on the configurations used, further explanations are available for the desired in-depth description in Chapter 4. It describes the system specification, workload for performance tests, security cases, and more.

4 EXPERIMENTS

This Chapter is a continuation of the method described in Chapter 3 and focuses on detailing the experiments. The types of experiments will be discussed with their objectives, the system, and workload, the metrics, and the details specific to their execution.

4.1 EXPERIMENT TYPES

As a means of comparing Istio to Linkerd, this research uses two types of experiments:

- Security tests: Used to ensure all configurations for the authorizations are working as intended and access to the services is allowed or blocked accordingly. As for the authentications, this study trusts the specification of both Linkerd and Istio and does not examine further than disclosed in Chapters 2 and 3, thus focusing on the authorizations and mutual Transport Layer Security (mTLS).
- Performance tests: Once the security configurations are in place, it is necessary to verify how much the added security impacts the performance of the service-to-service communication and which service mesh is faster than the other.

4.2 SYSTEM SPECIFICATION

Although the target environment for this research is the cloud, it was essential to ensure proper configuration and set a baseline for the performance of the system by doing local tests. Therefore, it was necessary to separate the experiments into local and cloud. Both employ the same configurations for the services and service meshes and are described in Kubernetes resources.

To enable Kubernetes, a Minikube environment was used for the local experiments. For the cloud experiments, Google Kubernetes Engine (GKE) was utilized.

4.2.1 Local system specification

The local was performed in the following system:

- Machine: A Dell notebook Vostro 15 5510.
- CPU: 11th Gen Intel(R) Core(TM) i7-11390H.
- Memory: 16GB.
- Operating system: Ubuntu 22.04.
- Network speed: Download speed of 200Mbps and Upload speed of 100Mbps.

The local Minikube cluster used to provide a local Kubernetes environment operated in the following specifications:

- CPU: 4 Cores.
- Memory: 8GB.

4.2.2 Cloud system specification

The GKE environment operated in the following specifications:

- Machine type: Custom.
- CPU: 4 vCPUs.
- Memory: 8GB.
- Operating system: Container Optimized OS.

4.2.3 Port forwarding

When running with Minikube, one alternative to accessing the running services is forwarding the traffic to a specific port. The command displayed in the code below shows how to forward after all the components are running on the cluster.

```
kubectl -n $MSC_NAMESPACE port-forward svc/msc-reverse-proxy 8080:80
```

It will forward traffic from *localhost:8080* to port 80 for the *msc-reverse-proxy* service in the selected namespace. The namespace variable present on the command is the namespace where the application is running.

4.3 SECURITY TESTS

The first part of the security tests is to check if Transport Layer Security (TLS) configuration is applied successfully to the communication between the services. The second part consists of a verification of the status returned from a request to an endpoint of the application. These confirmations will be better detailed in the incoming sections.

4.3.1 TLS tests

A question was raised when initially preparing these tests: “How to verify if TLS configuration is working for service-to-service communication?”. A noteworthy part of this question is that Istio and Linkerd apply mTLS by their default configuration, so no additional settings were required. Yet it is important to verify if their default configuration is in place. Thankfully, Istio and Linkerd have add-ons that can help perform this verification. These add-ons might not display information as seen in tools such as Wireshark [Wireshark Foundation, 2023], but they provide a way to view the flags used in the communication. With those flags, it is possible to check if the TLS flags related to the service meshes are set as anticipated. This will not provide access to the actual packet and its encryption, but that is not the objective of this section. It is exclusively to verify if the configuration is configured as intended while whether the TLS from the service meshes works can be subject to other research.

To perform this verification, it was necessary to send a request to the Experimenter, which then attempts to communicate with the Calculator and observe this traffic with a tool provided by the service meshes, checking if the associated TLS flag is set accordingly – see architecture presented in Figure 11.

4.3.1.1 Istio TLS verification

For Istio, Prometheus was used for verifying the TLS communication. It is possible to install it in the Istio mesh and then access details of network communication between the services. Prometheus can automatically collect these details.

4.3.1.2 Linkerd TLS verification

As for the TLS verification for Linkerd, Viz [Linkerd Authors, 2023b] was used, as it is an extension to Linkerd that collects data from network communication between the services and provides brief information on whether the TLS is used on it. The command `linkerd viz -n mastership tap deployment` displays in the terminal the details of the communication between the services.

4.3.2 HTTP status tests

The HTTP status tests consider the status returned as the response of a request to a service. For simplicity, the services used for the status tests are the Security Checker and the Experimenter. The experiments use the former for performing requests to itself or another Security Checker in a different namespace and to the Calculator service in the same or different namespace. The experiments use the latter for performing requests to the Calculator service in the same namespace. As such, the Calculator service is also involved in the tests as it is the destination for some tests, and the Reverse Proxy service is also part of the experiments since it is used as an entry point to the application.

Although security cases do not mention the Front-end and Analyzer services, this research accessed the application to perform the requests on the performance experiments. These experiments required using these services, thus ensuring communication to the Front-end and Analyzer services.

To better organize the tests to be performed, six distinct cases were prepared, as described in Table 2. These cases consist of which services are meshed and which are not.

They also specify which service will be involved in the security tests by following the tracking arrows on the images. These arrows present the flow of the requests, from left to right. It is important to remember that though the arrows show one-directional flow, all requests have their respective responses and it is the status of the responses the experiments verify.

This Section aims to describe these cases and their results. It is also noteworthy to mention that these tests were performed with all security configurations (specified in Section 3.3) set for the service meshes and that they were performed locally before experimenting on the cloud to ensure that the configurations worked as anticipated in both environments.

Table 2 – Security cases.

Case	Analyzer	Calculator	Experimenter	Frontend	Reverse Proxy	Security Checker	External Security Checker
1: No service meshed	Not meshed	Not meshed	Not meshed	Not meshed	Not meshed	Not meshed	Not meshed
2: All services meshed	Meshed	Meshed	Meshed	Meshed	Meshed	Meshed	Meshed
3: Calculator not meshed	Meshed	Not meshed	Meshed	Meshed	Meshed	Meshed	N/A
4: Security Checker and Experimenter not meshed	Meshed	Meshed	Not meshed	Meshed	Meshed	Not meshed	N/A
5: External meshed	Not meshed	Not meshed	Not meshed	Not meshed	Not meshed	Not meshed	Meshed
6: External not meshed	Meshed	Meshed	Meshed	Meshed	Meshed	Meshed	Not meshed

Source: The author.

4.3.2.1 Case 1: No service meshed

The case displayed in Figure 14 defines all services as not meshed, as it intends to verify the communication to and from not meshed services, even from a different namespace, and includes five tests:

1. Request from Experimenter to Calculator: Verify if requests from the Experimenter not meshed to the Calculator not meshed, originating from the same namespace, are authorized.
2. Request from Security Checker to Calculator: Verify if requests from non-meshed services to the Calculator not meshed, originating from the same namespace, are authorized.
3. Request from Security Checker to itself: Verify if requests from non-meshed services to non-meshed services, originating from the same namespace, are authorized.
4. Request from external Security Checker to Calculator: Verify if requests from non-meshed services to the Calculator not meshed, originating from a different namespace, are authorized.
5. Request from external Security Checker to Security Checker: Verify if requests from non-meshed services to non-meshed services, originating from a different namespace, are authorized.

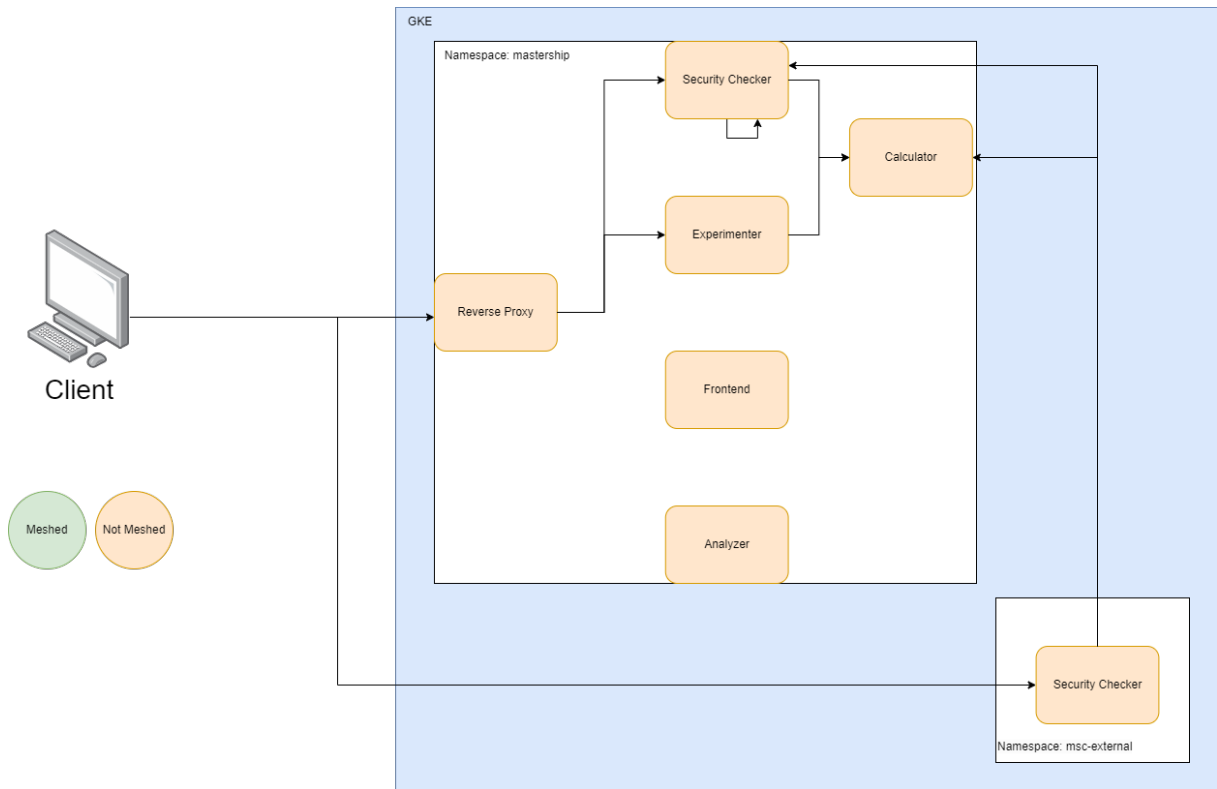
The external Security Checker uses the same definitions as the Security Checker, though in another namespace, the *msc-external* instead of the *mastership* namespace. It is used to check the communication to different namespaces.

To perform the requests using source services from the *mastership* namespace, it was used a port forwarding to the Reverse Proxy service, as described in the command presented in Section 4.2.3 since it is the entry point to the application. As for requests originating from the

external Security Checker, it was necessary to forward traffic directly to it as the Reverse Proxy employs configurations for access to the services on the *mastership* namespace exclusively.

These considerations also apply to the other security cases.

Figure 14 – Security case where no services are meshed.



Source: The author.

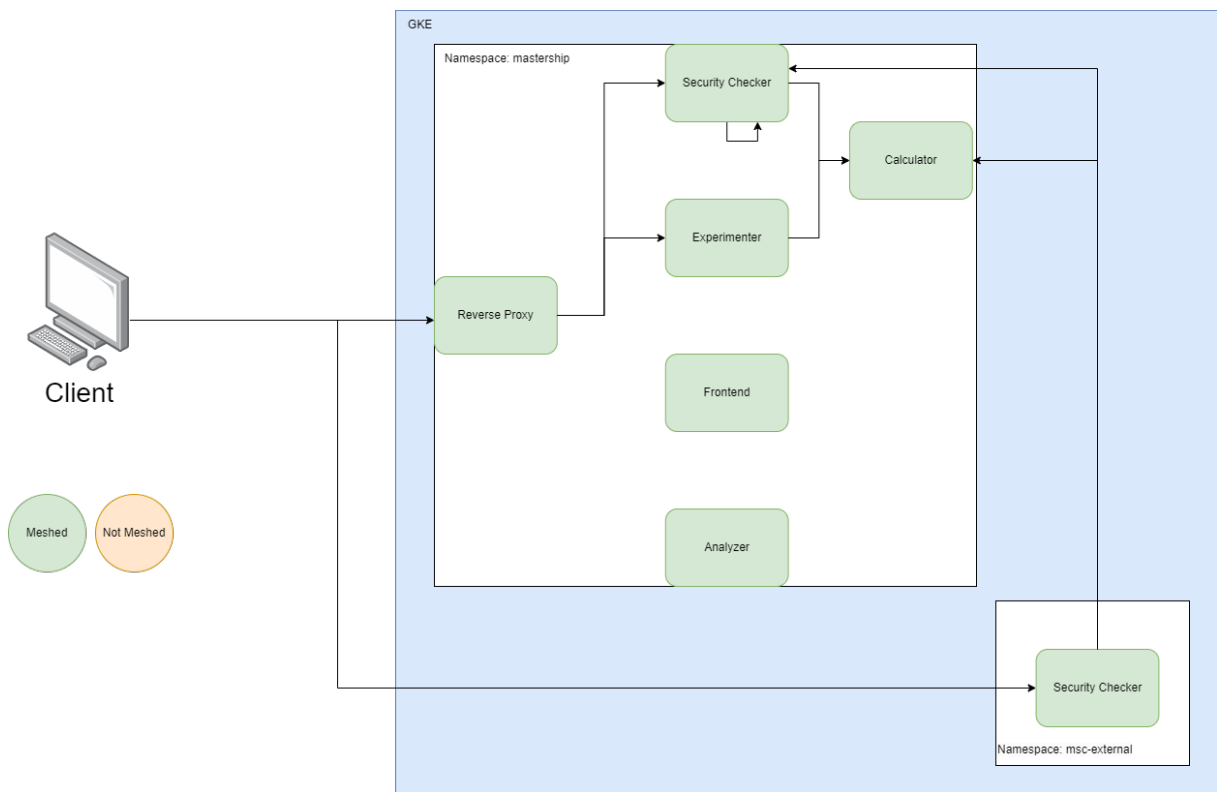
4.3.2.2 Case 2: All services meshed

The case displayed in Figure 15 defines all services as meshed, as it aims to validate the communication to and from meshed services, even from a different namespace, similarly to the case described in Section 4.3.2.1. It includes the same five tests, but in this case, all services are meshed:

1. Request from Experimenter to Calculator: Verify if requests from the Experimenter meshed to the Calculator meshed, originating from the same namespace, are authorized.
2. Request from Security Checker to Calculator: Verify if requests from meshed services to the Calculator meshed, originating from the same namespace, are denied.

3. Request from Security Checker to itself: Verify if requests from meshed services to meshed services, originating from the same namespace, are authorized.
4. Request from external Security Checker to Calculator: Verify if requests from meshed services to the Calculator meshed, originating from a different namespace, are denied.
5. Request from external Security Checker to Security Checker: Verify if requests from meshed services to meshed services, originating from a different namespace, are denied.

Figure 15 – Security case where all services are meshed.



Source: The author.

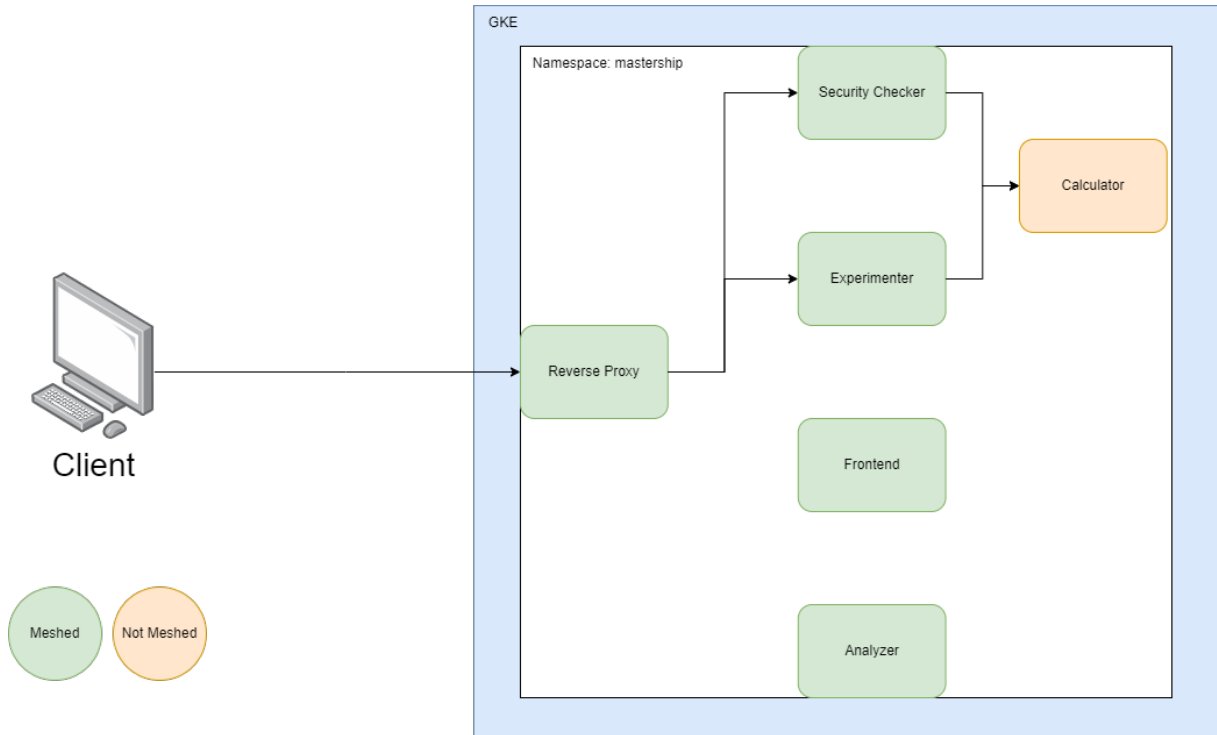
4.3.2.3 Case 3: Calculator not meshed

The case displayed in Figure 16 defines all services on the *mastership* namespace as meshed, except the Calculator. This case intends to verify the communication from meshed to not meshed services. It includes two tests to perform this validation.

1. Request from Experimenter to Calculator: Verify if requests from the Experimenter meshed to the Calculator not meshed, originating from the same namespace, are authorized.

2. Request from Security Checker to Calculator: Verify if requests from meshed services to non-meshed services, originating from the same namespace, are authorized.

Figure 16 – Security case where only the Calculator service is not meshed.



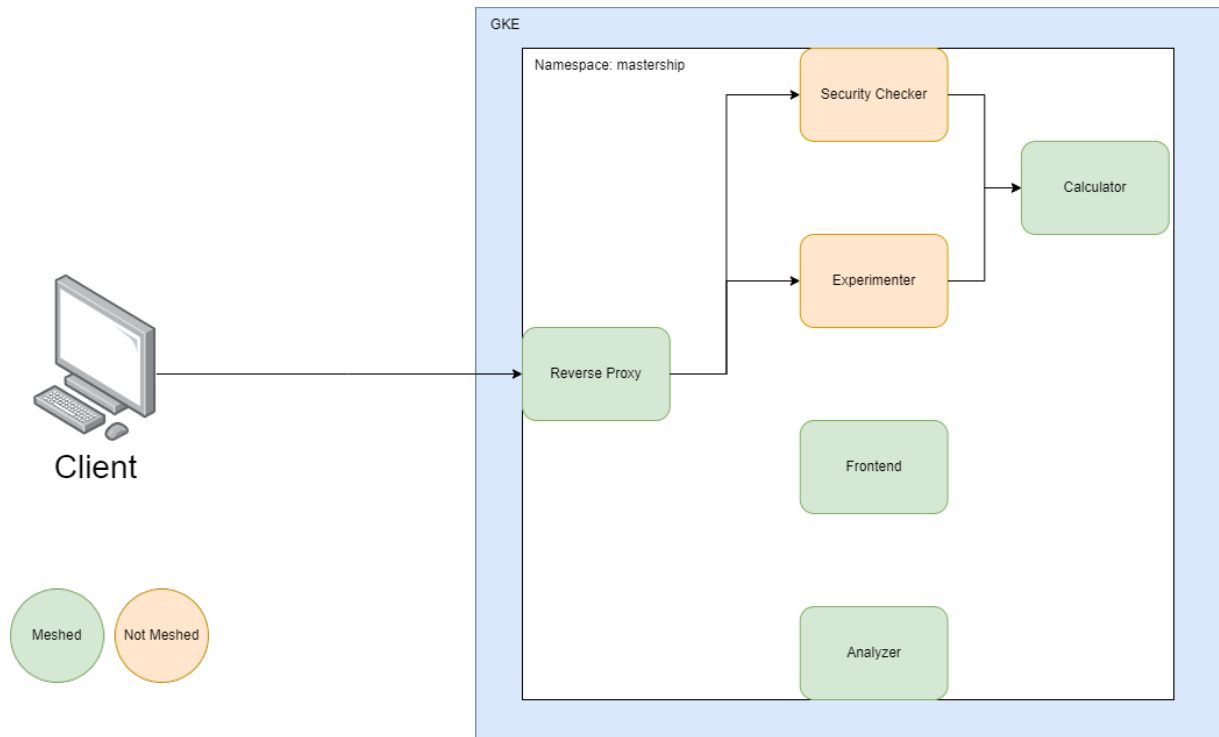
Source: The author.

4.3.2.4 Case 4: Security Checker and Experimenter not meshed

The case displayed in Figure 17 defines all services on the *mastership* namespace as meshed, except the Security Checker and the Experimenter, as it seeks to verify the communication from not meshed to meshed services. It is similar to the case described in Section 4.3.2.3, but the relevant meshed and not meshed services are swapped. It consists of two tests to perform the validation.

1. Request from Experimenter to Calculator: Verify if requests from the Experimenter not meshed to the Calculator meshed, originating from the same namespace, are denied.
2. Request from Security Checker to Calculator: Verify if requests from non-meshed services to meshed services, originating from the same namespace, are denied.

Figure 17 – Security case where only the Security Checker and Experimenter services are not meshed.



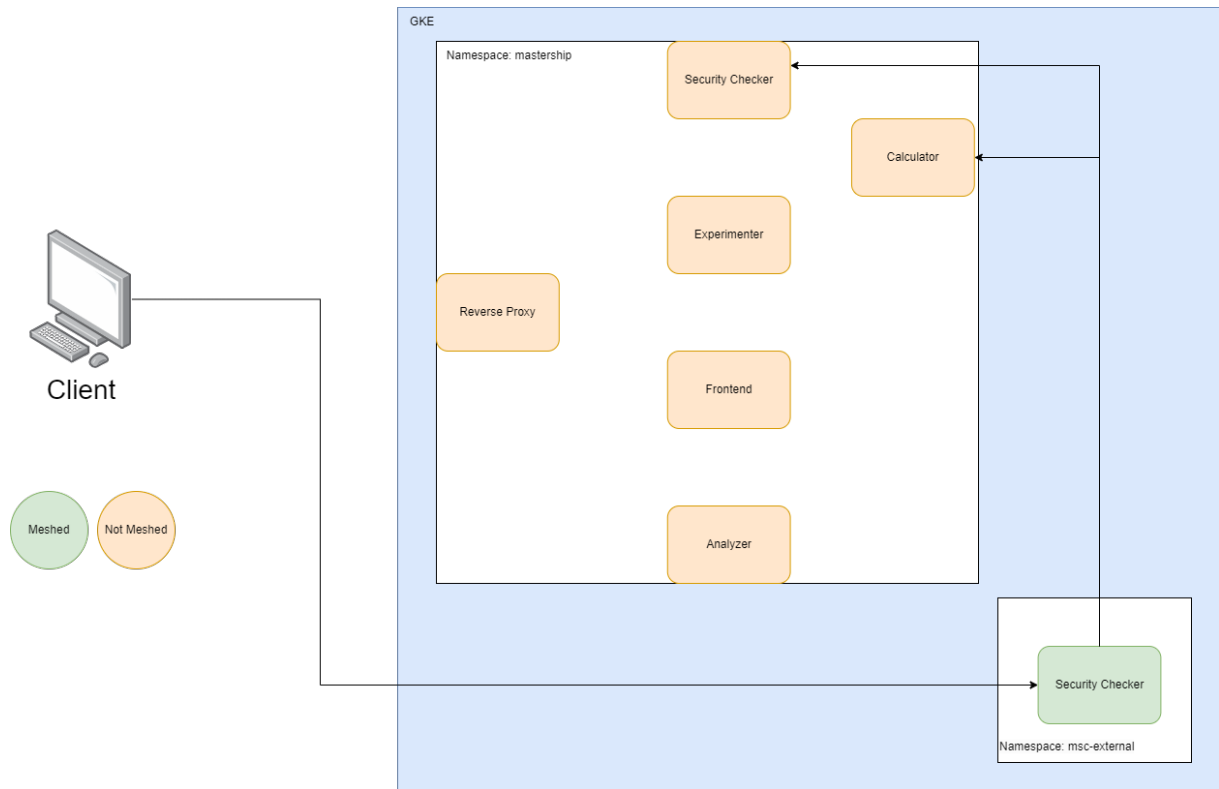
Source: The author.

4.3.2.5 Case 5: External meshed

The case displayed in Figure 18 defines all services on the *mastership* namespace as not meshed and the external Security Checker as meshed. The aim is to validate the communication from meshed to not meshed services but from a different namespace. It defines two tests to perform this validation.

1. Request from external Security Checker to Calculator: Verify if requests from meshed services to the Calculator not meshed, originating from a different namespace, are allowed.
2. Request from external Security Checker to Security Checker: Verify if requests from meshed services to non-meshed services, originating from a different namespace, are allowed.

Figure 18 – Security case where *msc-external* is meshed, but the *mastership* namespace is not.



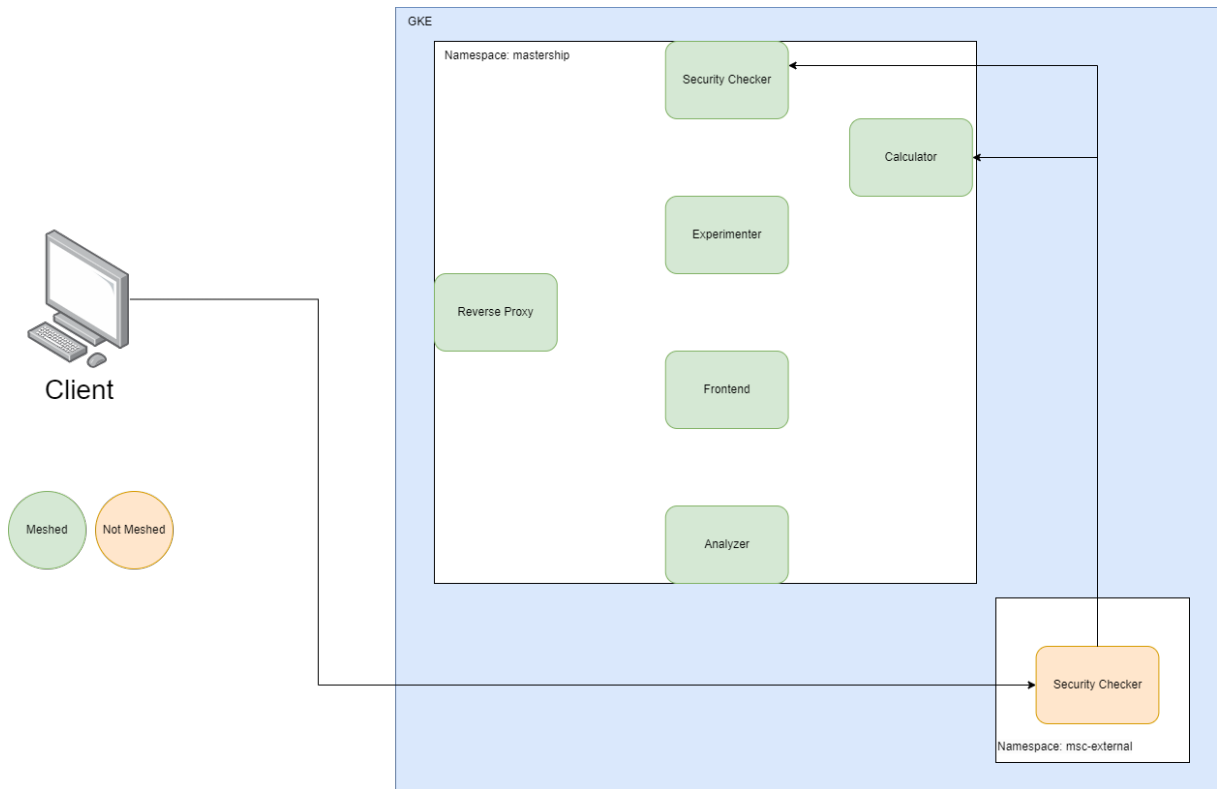
Source: The author.

4.3.2.6 Case 6: External not meshed

The case displayed in Figure 19 defines all services on the *mastership* namespace as meshed, but the external Security Checker as not meshed, as it intends to verify the communication from not meshed to meshed services, but from a distinct namespace. It is similar to the case described in Section 4.3.2.5, but the meshed and not meshed namespaces are swapped. It includes two tests to perform this validation.

1. Request from external Security Checker to Calculator: Verify if requests from non-meshed services to the Calculator meshed, originating from a different namespace, are denied.
2. Request from external Security Checker to Security Checker: Verify if requests from non-meshed services to meshed services, originating from a different namespace, are denied.

Figure 19 – Security case where *msc-external* is not meshed, but the *mastership* namespace is.



Source: The author.

4.4 PERFORMANCE EVALUATION

This research used a performance evaluation with the objectives of comparing the impacts on service-to-service communication performance when using a service mesh and not using a service mesh. Additionally, it also compares different service meshes, Istio and Linkerd, and different security configurations for both service meshes.

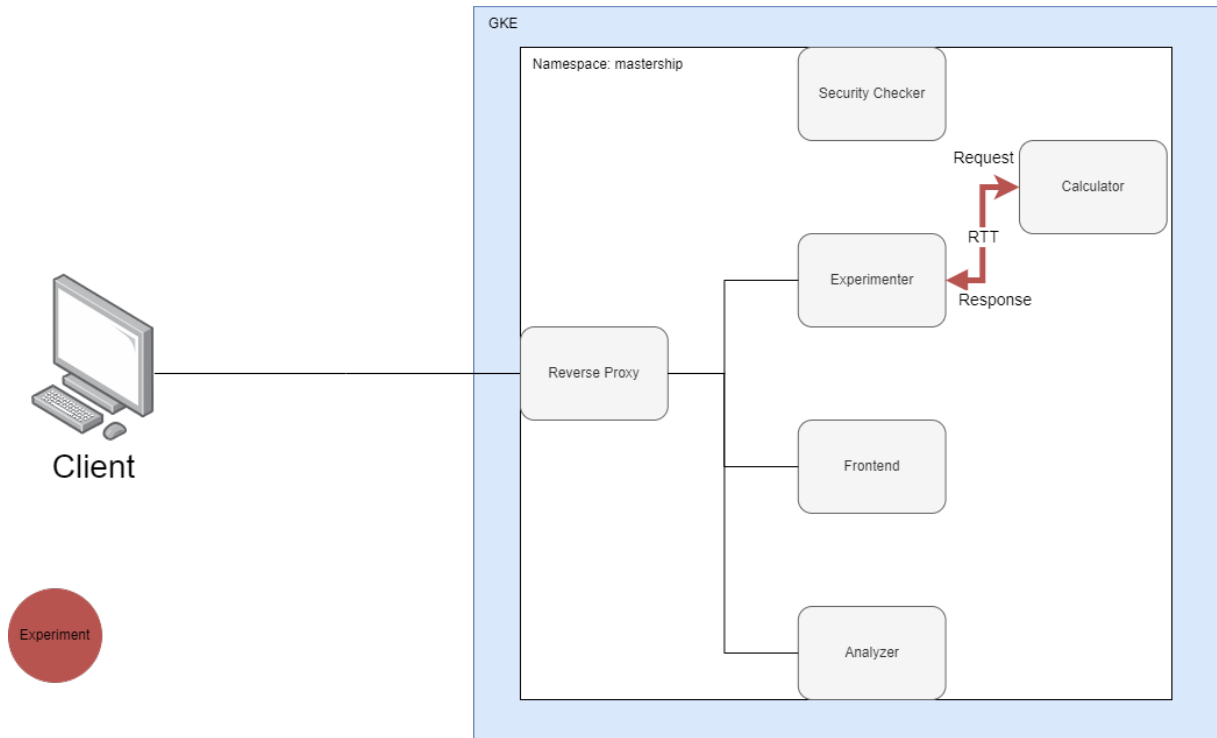
The performance evaluation used the round-trip times (RTTs) identified from requests from the Experimenter to the Calculator service. Figure 20 shows with a red arrow what interval was considered for the RTT. That is, the time necessary for a request from the Experimenter to reach and be processed by the Calculator and travel back to the Experimenter. It also indicates that other services were involved in the experiment, as the Reverse Proxy is the entry point for the application, the Frontend service was used for defining the parameters, and the Analyzer for calculating different metrics.

After calculating the RTTs, the mean, median, and standard deviation were calculated to allow evaluation of the experiment. Additionally, metrics such as RTT90, RTT95, and RTT99 were calculated to be used in addition to candlestick charts to facilitate analysis of

data distribution.

The results are presented in milliseconds and rounded to two decimal places.

Figure 20 – Services used on RTT evaluation.



Source: The author.

To perform the tests, different steps were taken:

1. Preliminary experiments: The preliminary experiments were performed locally with three scenarios: No service mesh, Istio, and Linkerd. No additional configuration was applied to either Istio or Linkerd, as this evaluation aimed to verify the performance of their default settings, building a baseline for the experiments that followed. Section 5.2.1 presents the results of these experiments.
2. Local experiments: Despite being called local experiments, the preliminary experiments were also performed only locally. These experiments differ from the preliminary experiments because the additional security configurations were applied to both Istio and Linkerd to check the impacts when compared to their base performance. It is possible to see the results of the local experiments in Section 5.2.2. These experiments allow for comparing the local performance effects with the cloud performance effects.
3. Local experiments with bigger numbers: These experiments used the same security configurations as the local experiments, but used bigger numbers to verify if increasing

the operands would create different results. It is possible to see the results of the local experiments in Section 5.2.3.

4. Cloud experiments: These are the most relevant experiments to this research. They used the same configurations as the local experiments as it was intended to validate if the performance remained consistent with the local results and to understand which service mesh would be the best regarding performance. Finally, Section 5.2.4 discusses the results of the cloud experiments.

4.4.1 Workload

The parameters used in the performance experiments followed the definitions presented in Table 3.

Table 3 – Workload for performance evaluation.

Requests	Requests per group	Interval per group	Allowed operations	Operands
10000	100	10 ms	sum, subtraction, multiplication, and division	0 to 100

Source: The author.

It declares an evaluation of 10,000 sequential requests. These requests are divided into groups of 100 with a 10ms interval between each group of sequential requests. It also allows all four operations the Calculator service provides (sum, subtraction, multiplication, and division) to be run, each on a single request and equal proportions. Each request sends integer operands from 0 to 100 to the Calculator service. These values are small so the calculated RTT consists mostly of time on the network.

These parameters were selected for a comprehensive evaluation of the system, which might not have been achievable with fewer requests. The groups and intervals try to simulate a more real-world scenario where batches of requests are made. Allowing all operations can also add to realism for a real-world scenario for a calculator, but the operands between 0 and 100 were selected for being small, to not affect significantly the overall RTT, as the focus is on service-to-service communication performance.

4.5 FINAL REMARKS

In this Chapter, both the security and the performance experiments performed during this research were detailed to facilitate replication and to present a guide on how to validate the security configurations and their effects on system performance.

5 RESULTS

This Chapter presents the results found during the experiments. First, the security results will be presented and then the performance results will be discussed.

5.1 SECURITY RESULTS

This Section seeks to show and discuss the results of the security tests from Sections 4.3.1 and 4.3.2.

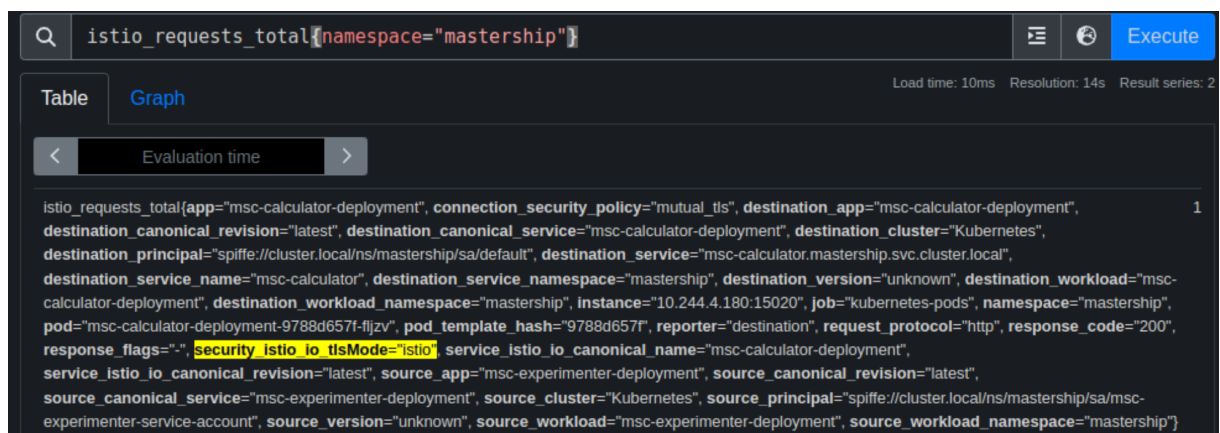
5.1.1 TLS results

This Section aims to present the results found during the TLS experiments.

5.1.1.1 Istio TLS results

In Figure 21 it is possible to see part of the dashboard provided by Prometheus querying for requests on the *mastership* namespace. In this image, the flag *tlsMode* is set as "istio", which indicates that it uses the mutual Transport Layer Security (mTLS) provided by the Istio service mesh to encrypt the communication.

Figure 21 – Confirming Transport Layer Security (TLS) with Istio.



Source: The author.

5.1.1.2 Linkerd TLS results

Part of the logs found when performing the request to the Experimenter service are presented in Figure 22. In it, it is possible to observe traffic incoming and outgoing from the Linkerd proxy on the Experimenter pod. The *tls* flag set to true indicates that the communication is indeed using the Transport Layer Security (TLS) provided by Linkerd.

Figure 22 – Confirming TLS with Linkerd.

```
req id=0:0 proxy=out src=10.244.4.195:46094 dst=10.244.4.197:8000 tls=true :method=POST :authority=msc-experimenter :path=/api/experimenter/experiment
req id=0:0 proxy=in src=10.244.4.195:59134 dst=10.244.4.197:8000 tls=true :method=POST :authority=msc-experimenter :path=/api/experimenter/experiment
```

Source: The author.

5.1.1.3 TLS results observations

The TLS tests have observed that both Istio and Linkerd are applying their respective mTLS flags to the communication of meshed services, this research thus assumes the mTLS is working as expected. It is relevant to note that this observation is based on ten verifications (five locally and five in the cloud) for both Istio and Linkerd, all presenting the same results regarding the setting of the mTLS flag.

5.1.2 HTTP status results

As for the HTTP status results, they are based on five complete samples (three based on local experiments and two from cloud experiments), all observations returning the same values, meaning a total of 55 cases run. The total number of cases run is then defined by Equation 5.1.

$$r = s * (nmc + mc * m) \quad (5.1)$$

- *r*: The total number of cases run.
- *s*: Number of samples.
- *nmc*: Number of non meshed cases.
- *mc*: Number of meshed cases.

- m : Number of meshes.

Considering the cases described in Section 4.3.2, it is then possible to extract the following values:

- s : Five samples were evaluated, then it is defined as 5.
- nmc : Case 1 is the only one without any meshes, then it is defined as 1.
- mc : Cases 2 to 6, then it is defined as 5.
- m : Two meshes were considered, Istio and Linkerd, then it is defined as 2.

Finally, it is then possible to conclude that the total number of cases run (r) is 55, as presented in Equation 5.2.

$$r = 5 * (1 + 5 * 2) = 55 \quad (5.2)$$

Tables 4 to 6 present the results found during the experiments, aggregating the results of the different cases, while Table 7 offers the results in a concise view.

The results displayed in Table 4 refer to requests when both target and source services are not meshed. As expected, it confirms that all requests return an HTTP 200 OK status, meaning the requests were authorized. It should be expected, as no additional security layer was applied.

Table 4 – Security results found when no mesh is applied.

Source/Target	Calculator (Not Meshed)	Security Checker (Not Meshed)
Experimenter (Not Meshed)	200 OK	N/A
Security Checker (Not Meshed)	200 OK	200 OK
External Security Checker (Not Meshed)	200 OK	200 OK

Source: The author.

5.1.2.1 Istio HTTP status results

The results presented in Table 5 refer to requests when target or source services are meshed with Istio. There are some noteworthy results in this evaluation:

1. Requests originating from a not meshed service and targeting a meshed service, resulted in a Connection Error.

2. The exception to the item above is when the source is the Experimenter service as not meshed and the target is the Calculator service as meshed, which was authorized and should not have been.

The investigation of these issues required isolating the additional Istio security configurations that were previously mentioned.

It was observed that the Connection Errors described in item 1 were related to the *STRICT PeerAuthentication* added to Istio, which blocks the non-mTLS traffic from accessing the services. The block on the access was expected, but an HTTP 403 Forbidden status would be more expressive. Still, it accomplishes the desired denial of access.

As for item 2, it was also anticipated an HTTP 403 Forbidden status, but instead, the request was authorized. It occurred because the *AuthorizationPolicy* applied to restrict the access to the meshed Calculator service used the service account of the Experimenter service to allow access, but the *AuthorizationPolicy* supersedes the *STRICT PeerAuthentication*. Therefore, despite not being meshed, the Experimenter is still authorized to access the Calculator. Further studies are required to verify how to prevent this vulnerability.

The other requests were authorized or not as expected.

Table 5 – Security results found when the Istio mesh is applied, the * mark noteworthy results.

Source/Target	Calculator (Meshed)	Calculator (Not Meshed)	Security Checker (Meshed)	Security Checker (Not Meshed)
Experimenter (Meshed)	200 OK	200 OK	N/A	N/A
Experimenter (Not Meshed)	200 OK*	N/A	N/A	N/A
Security Checker (Meshed)	403 Forbidden	200 OK	200 OK	N/A
Security Checker (Not Meshed)	Connection Error*	N/A	N/A	N/A
External Security Checker (Meshed)	403 Forbidden	200 OK	403 Forbidden	200 OK
External Security Checker (Not Meshed)	Connection Error*	N/A	Connection Error*	N/A

Source: The author.

5.1.2.2 Linkerd HTTP status results

The results displayed in Table 6 refer to requests when target or source services are meshed with Linkerd. There are also some noteworthy results in this evaluation:

1. Requests originating from a not meshed service and targeting the meshed Calculator service, resulted in Connection Error.
2. The exception to the above item is when the source is the Experimenter service as not meshed and the target is the Calculator service as meshed, which was authorized and should not have been.

3. The meshed external Security Checker is authorized to perform requests to the meshed Security Checker.

Item 3 is the core difference between the added security configurations to Istio and Linkerd. While a specific *AuthorizationPolicy* was added to Istio to block access from namespaces other than the *mastership*, a *default-inbound-policy* set as *cluster-authenticated* was added to Linkerd, which blocks access from not meshed services, similarly to the *STRICT PeerAuthentication* added to Istio, and blocks access from outside the cluster instead of blocking access from other namespaces, which is the case with Istio.

The investigation of items 1 and 2 involved isolating the additional Linkerd security configurations added by this research.

The investigations have shown that the Connection Errors relate to the HTTP/2 proxy protocol used by the Linkerd server set for the Calculator service. Changing it to HTTP/1 would make these requests return HTTP 403 Forbidden instead of the Connection Errors. The access block would be expected, but an HTTP 403 Forbidden status would be more specific. Still, it accomplishes the desired denial of access, and changing the proxy protocol to an older version to make these edge cases more explicit did not seem the best course of action for this research.

As for the second item, it was also expected an HTTP 403 Forbidden status, but instead, the request was authorized. Similarly to the issue found with Istio, this occurred because the *AuthorizationPolicy* applied to restrict the access to the meshed Calculator service used the service account of the Experimenter service to allow access and this *AuthorizationPolicy* is by-passing the *default-inbound-policy*, therefore, even if not meshed, the Experimenter is authorized to access the Calculator, similarly to Istio. Further investigations are required on how to configure Istio and Linkerd to make them capable of handling this vulnerability

Table 6 – Security results found when the Linkerd mesh is applied, the * mark noteworthy results.

Source/Target	Calculator (Meshed)	Calculator (Not Meshed)	Security Checker (Meshed)	Security Checker (Not Meshed)
Experimenter (Meshed)	200 OK	200 OK	N/A	N/A
Experimenter (Not Meshed)	200 OK*	N/A	N/A	N/A
Security Checker (Meshed)	403 Forbidden	200 OK	200 OK	N/A
Security Checker (Not Meshed)	Connection Error*	N/A	N/A	N/A
External Security Checker (Meshed)	403 Forbidden	200 OK	200 OK*	200 OK
External Security Checker (Not Meshed)	Connection Error*	N/A	403 Forbidden	N/A

Source: The author.

5.1.2.3 Summarizing HTTP status results

Finally, Table 7 takes the results found in Tables 5 and 6 and displays them in a more concise view and shows explicitly whether the results are authorized.

It is necessary to note the difference between Istio and Linkerd regarding the meshed-to-meshed communication on different namespaces, which makes the configuration added to Istio more secure than the one applied to Linkerd.

It is also essential to mention that despite Table 7 showing the communication from not meshed-to-meshed on the same namespace as Unauthorized, there exists the exception described previously regarding the not meshed Experimenter and the meshed Calculator which is authorized in both Istio and Linkerd, due to their authorization policies superseding the denial of access for not meshed services.

Table 7 – Summary of the security results.

Case	Istio	Linkerd
Meshed - Meshed (same namespace)	Authorized	Authorized
Meshed - Not Meshed (same namespace)	Authorized	Authorized
Not Meshed - Meshed (same namespace)	Unauthorized	Unauthorized
Meshed - Meshed (different namespace)	Unauthorized	Authorized
Meshed - Not Meshed (different namespace)	Authorized	Authorized
Not Meshed - Meshed (different namespace)	Unauthorized	Unauthorized

Source: The author.

5.1.2.4 Cloud security differences

No difference was found in the security experiments on the cloud when compared to the local experiments, but it is still relevant to mention a finding:

Section 4.2.3 describes the process of forwarding traffic to a port with Kubernetes on Minikube. On Google Kubernetes Engine (GKE), that would not typically be necessary, as access to an <IP:port> is enabled automatically to the ingress assigned to the Reverse Proxy service. However, as was to be anticipated, by adding the additional security layers to the meshes that block access from not meshed services and from outside the cluster or namespace, based on the service mesh, the access to the Reverse Proxy is also blocked.

Port forwarding can bypass this issue by directly accessing the service enabling both the security and performance experiments to be executed, and it was necessary to use it on the cloud experiments. Even though forwarding is a solution adequate for this research, for a real-world and production-level system, it might not be acceptable and, thus, require additional

tuning on the security policies added to Istio and Linkerd to allow external access to the Reverse Proxy service while keeping the same level of internal security.

5.2 PERFORMANCE RESULTS

This Section presents the results found during the performance experiments.

5.2.1 Preliminary results

The results displayed in Table 8 portray the scenario without a service mesh as the fastest, as was to be expected since no additional layers were added to the application.

Table 8 – Comparison between the different scenarios on the preliminary evaluation.

Mesh	Mean	Median	Standard Deviation	RTT 90	RTT 95	RTT 99
No-Mesh	0.07 ms	0.06 ms	0.03 ms	0.08 ms	0.10 ms	0.19 ms
Istio	0.48 ms	0.37 ms	0.36 ms	0.80 ms	0.93 ms	1.57 ms
Linkerd	0.22 ms	0.19 ms	0.14 ms	0.31 ms	0.46 ms	0.65 ms

Source: The author.

Using the median as a metric, it is possible to see that the no-mesh scenario resulted in a **0.06ms** median round-trip time (RTT), while Linkerd resulted in **0.19ms** and Istio in **0.37ms**. It translates to an increase of approximately 217% for Linkerd and 517% for Istio when compared to the no-mesh scenario.

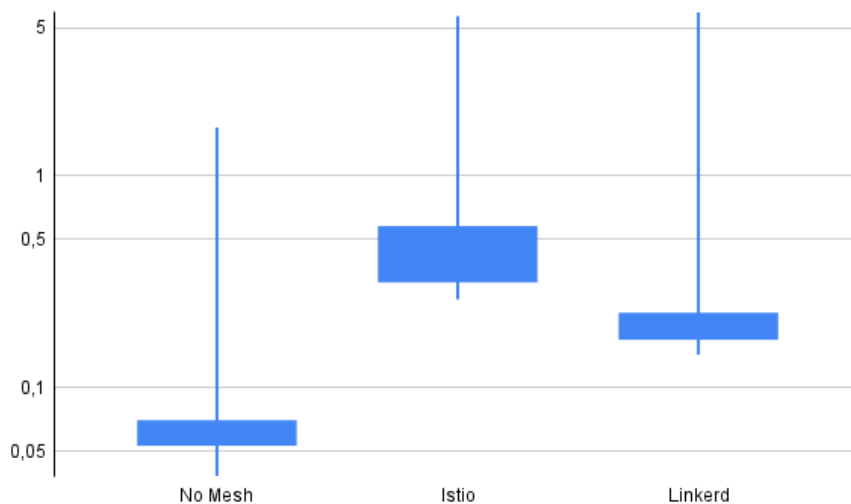
While it was expected that the no-mesh scenario would outperform the service meshes, it is noteworthy how Istio can affect the RTT. The increase in the RTT with Linkerd, while considerable, can be less impactful when compared to the Istio mesh. That is to be expected since one of the objectives of Linkerd is to be lightweight for the application.

The standard deviation is relatively close to the mean and median values. It is also possible to observe that the median value is lower than the mean value. It configures a positively skewed distribution. As presented in the incoming sections, the considerable standard deviation and the positively skewed distribution occur in all scenarios.

It is possible to see a candlestick representation of the RTTs found in the preliminary experiments on the graph displayed by Figure 23. The candlestick representation is relevant when evaluating the RTTs in the applied experiments, as it can help to understand how the values are distributed. Notably, the 25% greater RTTs are considerably outliers, although this

is less prominent due to the scale applied. The first RTT was discarded in all cases as it can be considered as the request that initialized the procedure, often the biggest of the results, not representing the actual results found in the experiments. The data presented on the graph is available in Table 9.

Figure 23 – Candlestick graph for RTTs found on the preliminary results.



Source: The author.

Table 9 – Candlestick graph data of the preliminary results.

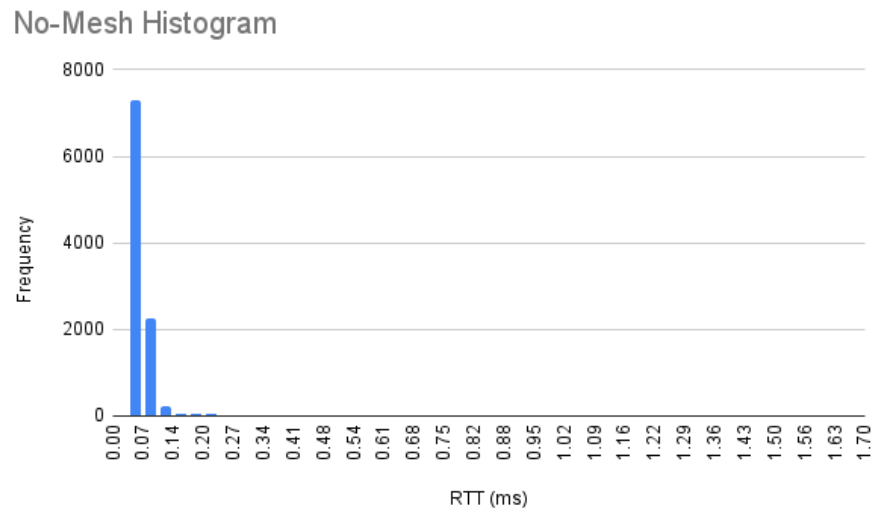
Mesh	Minimum	First Quartile	Median	Third Quartile	Maximum
No-Mesh	0.04 ms	0.05 ms	0.06 ms	0.07 ms	1.69 ms
Istio	0.26 ms	0.32 ms	0.37 ms	0.57 ms	5.68 ms
Linkerd	0.14 ms	0.17 ms	0.19 ms	0.22 ms	5.96 ms

Source: The author.

Comparing the maximum value to the value for the RTT99 shows that the top 1% results significantly differ from the remaining values. It is also a regular occurrence in the different scenarios investigated in this research.

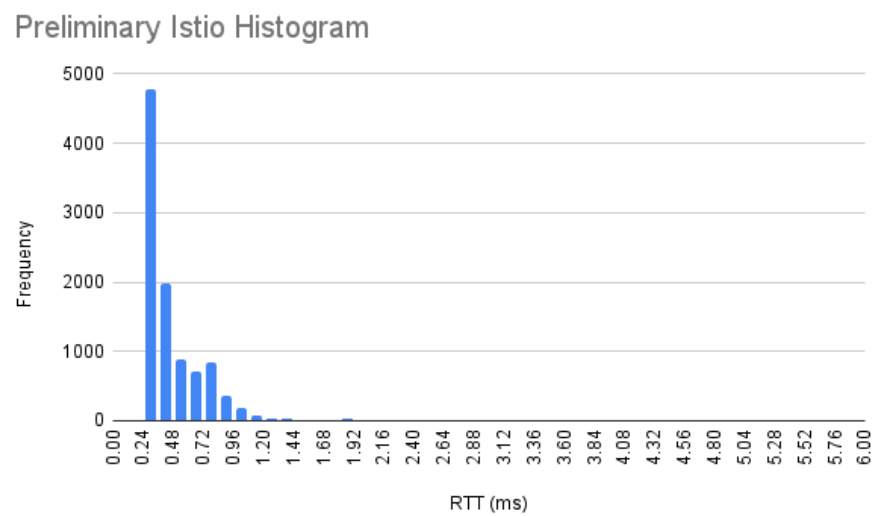
For a more detailed view of the distribution, histograms for the no-mesh, Istio, and Linkerd scenarios are displayed in Figures 24, 25, and 26.

Figure 24 – Histogram for RTTs found on the preliminary and local evaluation for the no-mesh scenario.



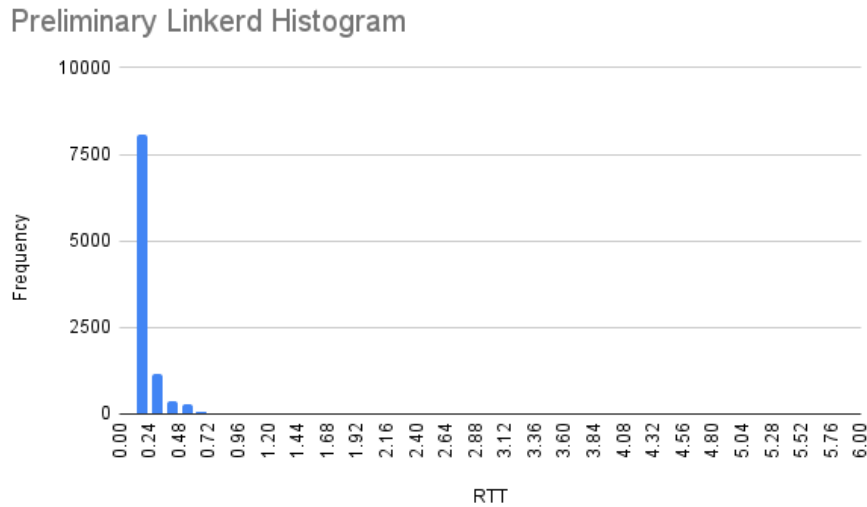
Source: The author.

Figure 25 – Histogram for RTTs found on the preliminary evaluation for the Istio scenario.



Source: The author.

Figure 26 – Histogram for RTTs found on the preliminary evaluation for the Linkerd scenario.



Source: The author.

5.2.2 Local results

For the local experiments, meaning the local experiments with the service meshes using all the added security configurations, the no-mesh results were re-used from the preliminary results, presented in Table 8, as no change was added to the no-mesh structure. That is, the no-mesh scenario for the preliminary evaluation is the same as for the local evaluation. Table 10 presents the metrics found in the local experiments.

Table 10 – Comparison between the different scenarios on the local evaluation.

Mesh	Mean	Median	Standard Deviation	RTT 90	RTT 95	RTT 99
No-Mesh	0.07 ms	0.06 ms	0.03 ms	0.08 ms	0.10 ms	0.19 ms
Istio	0.52 ms	0.44 ms	0.37 ms	0.80 ms	0.90 ms	1.09 ms
Linkerd	0.23 ms	0.21 ms	0.09 ms	0.29 ms	0.33 ms	0.52 ms

Source: The author.

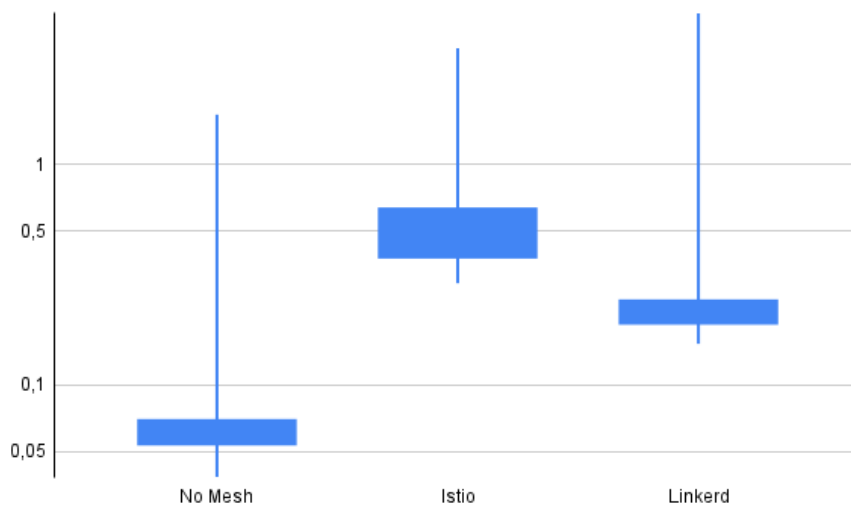
As before, it is possible to see that the no-mesh scenario outperforms the service meshes, as was foreseen. It is also possible to note that for both median and mean, the added security configurations increased the values for the service meshes, compared to their preliminary evaluations. As for the standard deviation, it was reduced for Linkerd which translates to a more consistent RTT.

It is possible to observe that the added configurations impacted **0.07ms** for Istio and **0.02ms** for Linkerd when using the median as a metric and comparing it to their preliminary

results. It might be a slight compromise considering the added security.

The candlestick representation for the results found on the local experiments can be seen in Figure 27, while the data for the chart is present in Table 11. The first RTT was also discarded as in Section 5.2.1. When compared to Figure 23, it is notable that all values are bigger than the preliminary results, except the maximum values.

Figure 27 – Candlestick graph for RTTs found on the local results.



Source: The author.

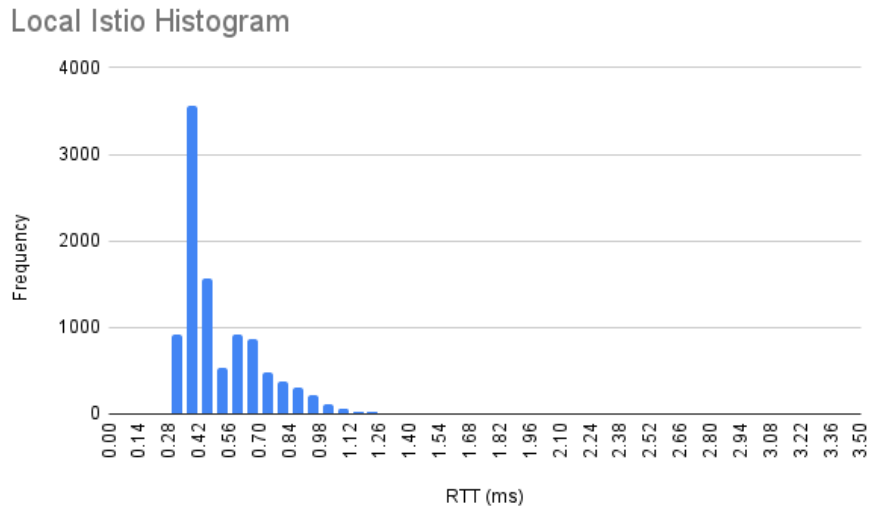
Table 11 – Candlestick graph data of the local results.

Mesh	Minimum	First Quartile	Median	Third Quartile	Maximum
No-Mesh	0.04 ms	0.05 ms	0.06 ms	0.07 ms	1.69 ms
Istio	0.29 ms	0.38 ms	0.44 ms	0.63 ms	3.38 ms
Linkerd	0.15 ms	0.19 ms	0.21 ms	0.24 ms	4.90 ms

Source: The author.

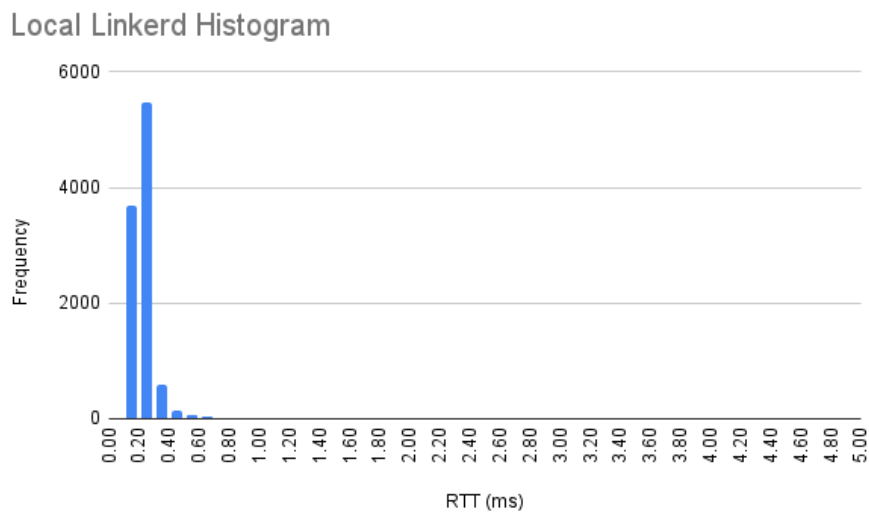
The histograms for the local results are also presented in Figures 24, 28 and 29, for the no-mesh, Istio, and Linkerd, respectively.

Figure 28 – Histogram for RTTs found on the local evaluation for the Istio scenario.



Source: The author.

Figure 29 – Histogram for RTTs found on the local evaluation for the Linkerd scenario.



Source: The author.

5.2.3 Local results with bigger numbers

One question raised during the experiments was: If the workload required additional computation, would the results change significantly? Since the service used for actual operations was the Calculator service, it would qualify for limited options to increase the computation required per operation. To use additional computation, the applied solution was to increase the numbers from 0 to 100, to 0 to 10^{16} .

This Section explains the results found by applying the same workload specification as the other sections but with the bigger numbers described above. Table 12 displays no significant difference when compared to Table 10 for the no-mesh and Linkerd environments. As for Istio, the difference is more prominent.

Table 12 – Comparison between the evaluation of the different scenarios on the local environment with bigger numbers.

Mesh	Mean	Median	Standard Deviation	RTT 90	RTT 95	RTT 99
No-Mesh	0.06 ms	0.06 ms	0.03 ms	0.08 ms	0.09 ms	0.20 ms
Istio	0.72 ms	0.57 ms	0.38 ms	1.12 ms	1.31 ms	2.16 ms
Linkerd	0.23 ms	0.21 ms	0.08 ms	0.28 ms	0.32 ms	0.49 ms

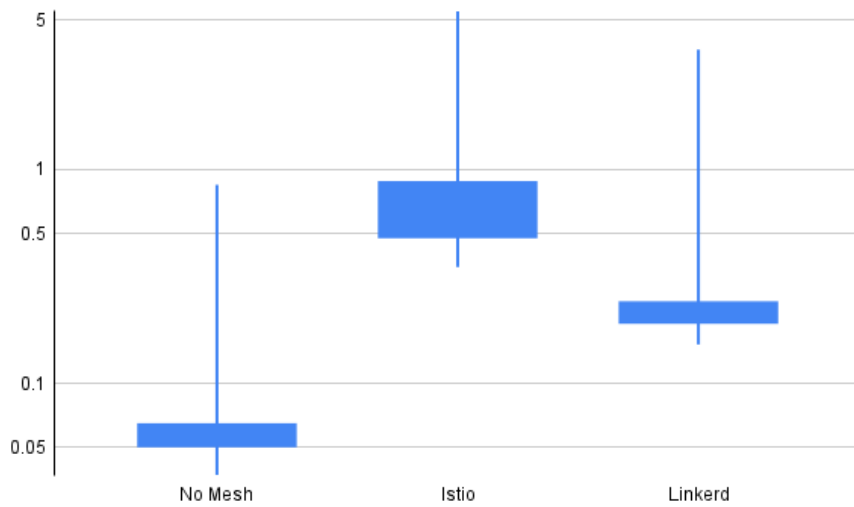
Source: The author.

Using the RTT99 as a metric, the value is almost doubled. Selecting other metrics still demonstrates significant changes. The small difference, mostly of **0.01ms** regarding no-mesh and Linkerd to their previous references, could have been anticipated as changing the Experimenter to send bigger values, might not have been enough to create a significant change, and the difference found can be due to network influences. Notably, this is not the case with the Istio mesh. The standard deviation remained consistent, but there was a significant increase in the mean and median, displaying greater values in general. It could be due to the change in the numbers sent to the Calculator service, but it does not align with the no-mesh and Linkerd scenarios. Further investigations on this end, preferably with more costly operations would be appropriate.

The candlestick representation for the results found on the experiments can be seen in Figure 30, while the data for the chart is present in Table 13. The first RTT was discarded as in the previous sections. When compared to Figure 27, it is remarkable that most values in the no-mesh and Linkerd values remain consistent, while Istio displays a slight increase in the base values. It is possible to see that the maximum values were reduced for the no-mesh and Linkerd scenarios, but the other quartiles remained almost unchanged. It translates to less significant outliers, which demonstrates that the changes in the RTT in the experiments are most likely caused by the network fluctuations, not by the modification in the operation.

The histograms for the cloud results for no-mesh, Istio, and Linkerd are displayed in Figures 31, 32, and 33.

Figure 30 – Candlestick graph for RTTs found on the local environment with greater numbers.



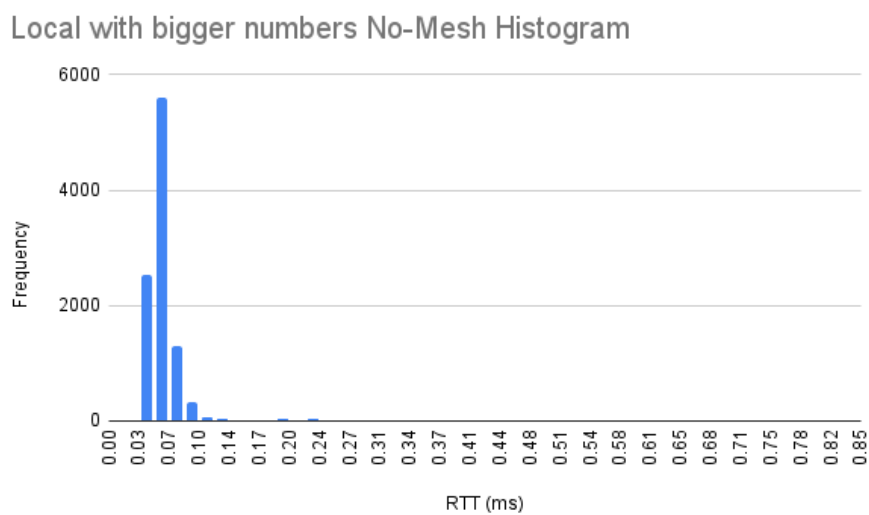
Source: The author.

Table 13 – Candlestick graph data of the results found with the local environment with bigger numbers.

Mesh	Minimum	First Quartile	Median	Third Quartile	Maximum
No-Mesh	0.04 ms	0.05 ms	0.06 ms	0.6 ms	0.85 ms
Istio	0.35 ms	0.49 ms	0.57 ms	0.87 ms	5.54 ms
Linkerd	0.15 ms	0.19 ms	0.21 ms	0.24 ms	3.64 ms

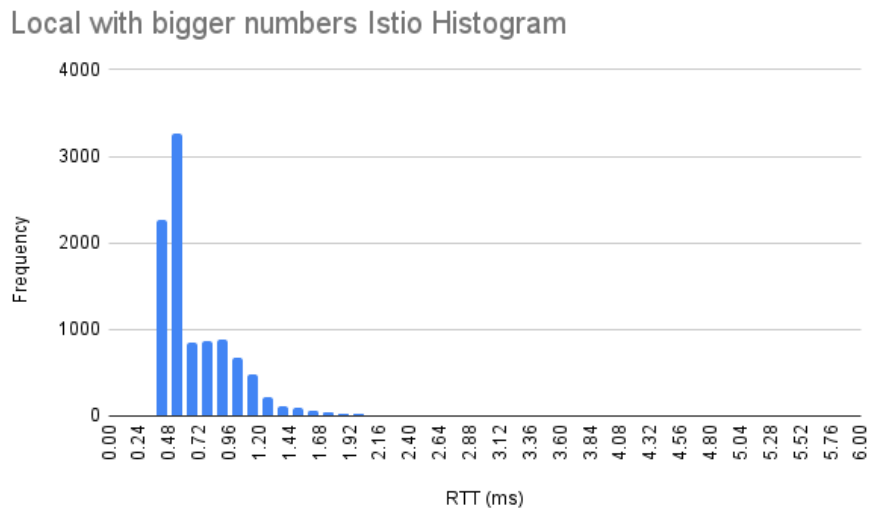
Source: The author.

Figure 31 – Histogram for RTTs found on the local evaluation with bigger numbers for the no-mesh scenario.



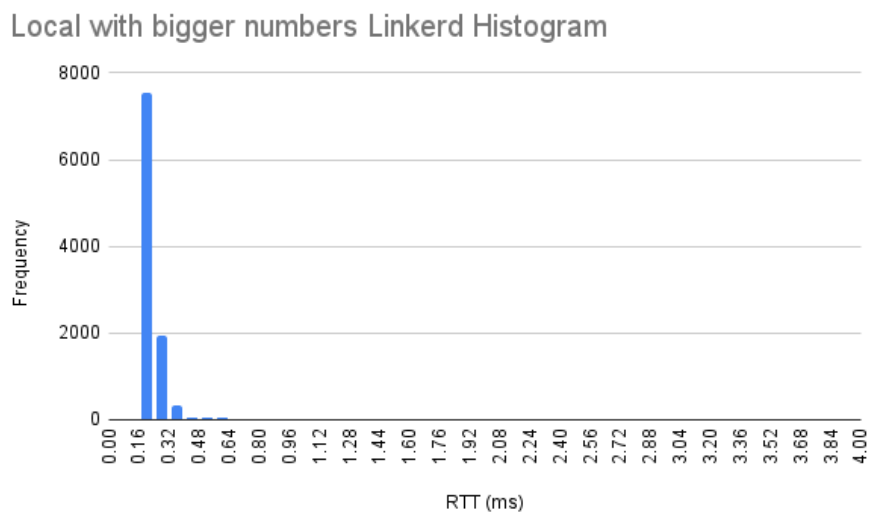
Source: The author.

Figure 32 – Histogram for RTTs found on the local evaluation with bigger numbers for the Istio scenario.



Source: The author.

Figure 33 – Histogram for RTTs found on the local evaluation with bigger numbers for the Linkerd scenario.



Source: The author.

Answering the question raised at the beginning of this section: If the workload required additional computation, would the results change significantly? It can change significantly, but further studies are required for better-elaborated responses.

5.2.4 Cloud results

Ultimately, one of the final objectives of the experiments is to check how the added security would impact a cloud environment. After developing a baseline for the expectations, as

described in Sections 5.2.1 and 5.2.2, it was possible to run the performance experiments on Google Cloud Platform (GCP).

The same metrics used in the previous sections, but applied to the cloud results, are exhibited in Table 14. As was expected, there was a considerable increase in RTT time when compared to Table 10. It is due to how the cloud is configured and how it routes the requests, as opposed to the direct access, possible in a local environment.

Table 14 – Comparison between the different scenarios on the cloud evaluation.

Mesh	Mean	Median	Standard Deviation	RTT 90	RTT 95	RTT 99
No-Mesh	0.57 ms	0.46 ms	0.40 ms	0.89 ms	1.11 ms	1.99 ms
Istio	3.75 ms	3.57 ms	1.33 ms	4.89 ms	5.39 ms	6.84 ms
Linkerd	2.16 ms	2.02 ms	0.67 ms	2.87 ms	3.23 ms	4.65 ms

Source: The author.

Using the median as a metric, it is possible to see that the case without a mesh increased from **0.06ms** to **0.46ms**, an increase of about 667%. The case with an Istio mesh shows an increase of 676%, when compared to the no-mesh scenario, while the case with a Linkerd mesh shows an increase of approximately 339%. Thus, these results also reveal the Linkerd mesh is faster than the Istio mesh, as in the local experiments.

As the mean and median increased when compared to the local results, the standard deviation also increased for all cases, which translates to less stable results, but that can be reasonable, as the average RTT also increased significantly.

The candlestick representation for the cloud results can be seen in Figure 34 while its data is displayed in Table 15. As expected, it shows bigger results in general when compared to Figure 27.

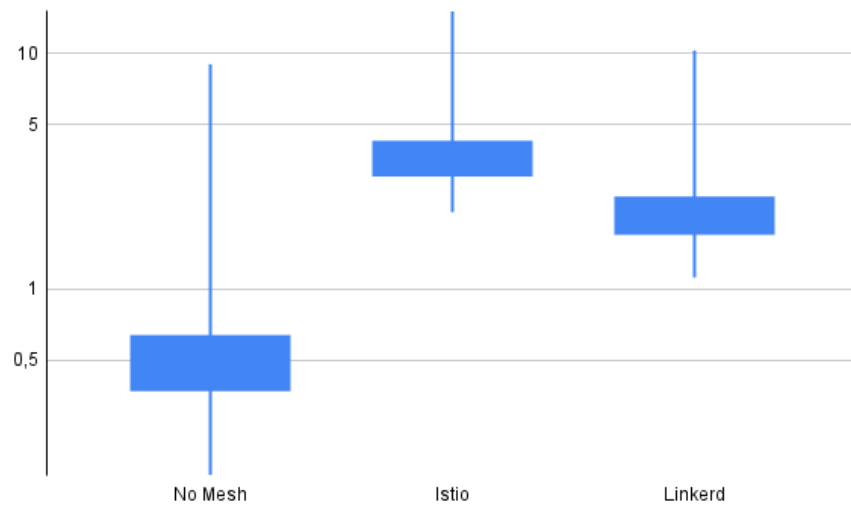
Table 15 – Candlestick graph data of the cloud results.

Mesh	Minimum	First Quartile	Median	Third Quartile	Maximum
No-Mesh	0.16 ms	0.38 ms	0.46 ms	0.63 ms	9.01 ms
Istio	2.13 ms	3.05 ms	3.57 ms	4.20 ms	15.19 ms
Linkerd	1.12 ms	1.73 ms	2.02 ms	2.44 ms	10.29 ms

Source: The author.

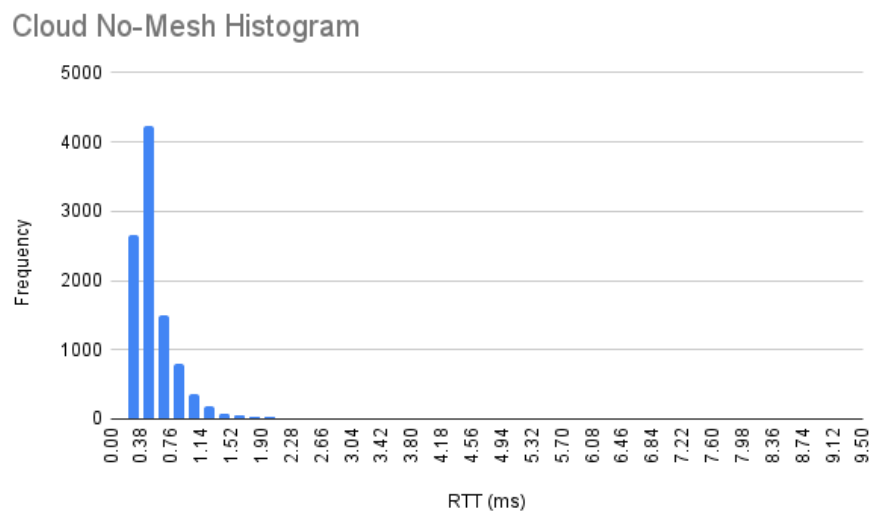
The histograms for the cloud results for no-mesh, Istio, and Linkerd are displayed in Figures 35, 36, and 37.

Figure 34 – Candlestick graph for RTTs found on the cloud results.



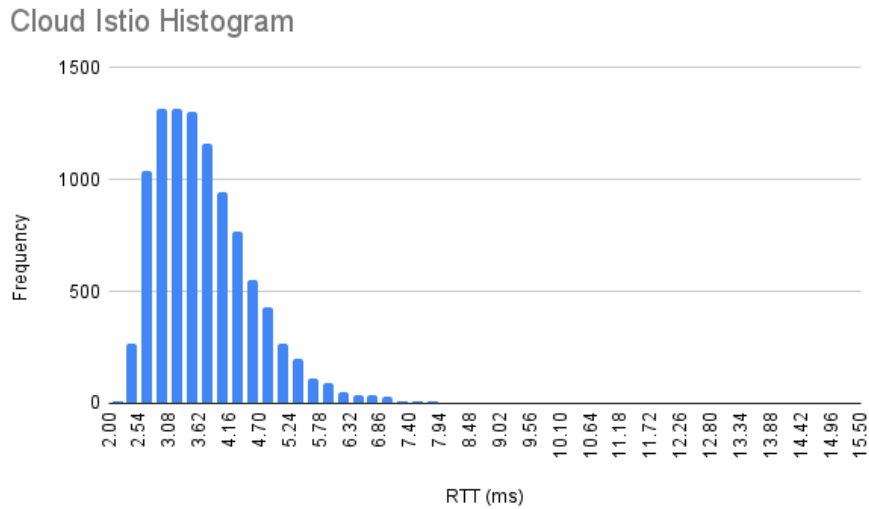
Source: The author.

Figure 35 – Histogram for RTTs found on the cloud evaluation for the no-mesh scenario.



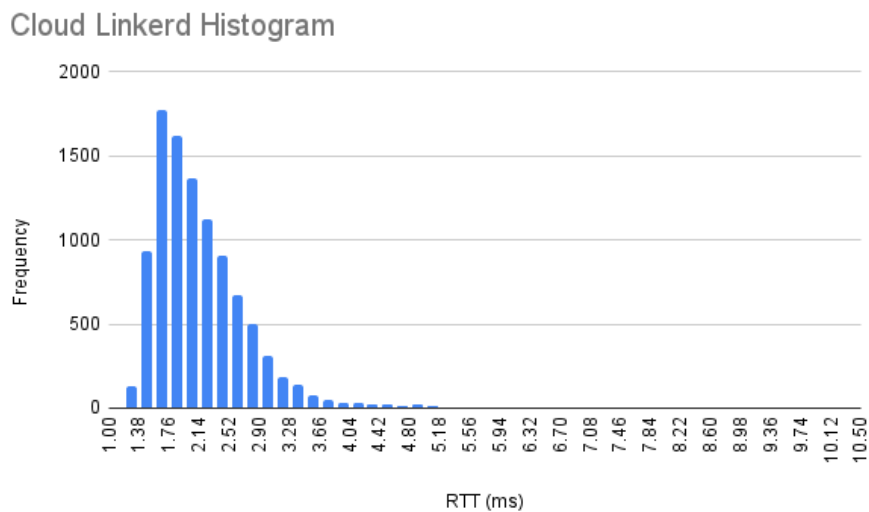
Source: The author.

Figure 36 – Histogram for RTTs found on the cloud evaluation for the Istio scenario.



Source: The author.

Figure 37 – Histogram for RTTs found on the cloud evaluation for the Linkerd scenario.



Source: The author.

5.2.5 Performance observations

All performance experiment cases described in the previous sections characterize a positively skewed distribution, *i.e.*, a distribution with the most values to the left of the mean value, but with the most outlier values to the right. It is expected since most RTTs are too small, leaving no space for them to be outliers to the left. It is the case even discarding the first RTT, which could be more than nine times the maximum without it.

The histograms presented for the distribution of the results in each of the scenarios present skewed distributions, which are non-normal distributions, highlighting the differences in frequency of the bigger values when compared to the smaller ones.

Table 16 compares each scenario and the extreme outliers, here considered as the top 1% of the values. This comparison is made by considering the RTT99 and the maximum values. It is possible to see that in general, Istio produced a smaller relative difference when compared to Linkerd and the no-mesh scenarios, which could achieve more than 800% difference. Additionally, it is also noteworthy that cloud differences were also smaller when compared to their local counterparts, in both cases this could be a result of bigger values in general, thus reducing the difference to the extreme bigger values.

Table 16 – Extreme outliers summary.

Environment	RTT 99	Maximum	Difference
Local - No Mesh	0.19 ms	1.69 ms	789%
Local - Preliminary - Istio	1.57 ms	5.68 ms	262%
Local - Preliminary - Linkerd	0.65 ms	5.96 ms	817%
Local - Istio	1.09 ms	3.38 ms	210%
Local - Linkerd	0.52 ms	4.90 ms	842%
Heavy - Local - No Mesh	0.20 ms	0.85 ms	325%
Heavy - Local - Istio	2.16 ms	5.54 ms	156%
Heavy - Local - Linkerd	0.49 ms	3.64 ms	643%
GKE - No Mesh	1.99 ms	9.01 ms	353%
GKE - Istio	6.84 ms	15.19 ms	122%
GKE - Linkerd	4.65 ms	10.29 ms	121%

Source: The author.

The standard deviation was, in some cases, more than half of the mean RTT, and as was presented in the candlestick graphs, this was primarily due to the outliers to the right, specifically the 25% larger values which were visually far from the remaining 75%, e.g., the Linkerd maximum value, described as **5.96ms**, while the third quartile was described as **0.22ms** in Section 5.2.1. It remains consistent when comparing the maximum to the RTT99, as it can indicate that the top 1% values can be distant from the remaining 99%. Taking the results for Istio in Section 5.2.4, Istio delivers an RTT99 of **6.84ms** and a maximum of **15.19ms**, meaning the top 1% of the values can be more than double the biggest of the remaining 99%.

Considering the skewed distributions, the extreme outliers, and the considerable standard deviations, the median was selected to compare the results. Considering the median as the

metric for comparison, it is evident that the no-mesh outperformed the related meshed scenarios in all cases, while Linkerd outperformed Istio, which was expected due to Linkerd's focus on being lightweight. However, no hypothesis test was applied, which could have increased the statistical validity of the results

5.3 FINAL REMARKS

In this Chapter, the results found during the experiments were reviewed and required further investigations presented appropriately.

The security applied was verified to guarantee the configurations were set correctly. It was then observed that Istio's configuration was more secure than Linkerd's. However, both had the same problem regarding the authorizations with the Calculator service enabling communication from outside the mesh if the service attempting communication used the service account from the Experimenter service. It was also discussed that, for production-level usage, additional effort could be necessary to define the authorization policy for the Reverse Proxy service to enable communication from outside the cluster.

The performance experiments have shown that the Linkerd mesh is faster than Istio and that the difference in the RTTs, while significant when considered the RTTs found in the no-mesh scenarios, can be of a few milliseconds and thus might be an acceptable compromise for the added security.

6 CONCLUSION

This Chapter finalizes the current research, reviewing and summarizing the findings made during the study and the value of these contributions and relations to the research questions. Finally, it will mention the limitations and opportunities for future work.

6.1 RESULTS

This research sought to understand how service meshes can enable Zero Trust approaches to service-to-service communication. It presented tools and configurations provided by the service meshes that can help in applying ZT principles to service-to-service communication, their impact on communication performance, and options for service meshes.

In addition, this research tried to answer the following questions:

1. How can service meshes be applied to enforce Zero Trust principles in service-to-service communication?
2. How does Zero-Trust protection affect service-to-service performance?
3. How do widely adopted service meshes, Istio and Linkerd, compare to each other in terms of security and performance?

This research explained that for the first item, the service meshes provide diverse tools to restrict, monitor, and secure communication between services, such as service credentials, mutual Transport Layer Security (mTLS), authorization policies, and integration with monitoring tools.

To answer the second question, this research investigated the round-trip time (RTT) for the communication of non-meshed services and compared it to the RTT found when using service meshes, finding that applying ZT principles with the service meshes does increase network communication time. It is a compromise between security and performance and the added strain in the communication time might be acceptable depending on the application and the requirements.

Finally, to answer the third question, the research compared two service meshes, Istio and Linkerd, illustrating similarities and differences between their resources for configurations.

The results indicated that Linkerd provides less strain in the time required for communication between the services while applying a similar, though not identical, level of protection. Tables 17 and 18 presents the similarities and differences from the investigated settings for Istio and Linkerd while Table 19 displays their performance overall performance.

Table 17 – Summary of mTLS results.

Mesh	mTLS
Istio	Yes
Linkerd	Yes

Source: The author.

Table 18 – Summary of the authorization results.

Case	Istio	Linkerd
Meshed - Meshed (same namespace)	Authorized	Authorized
Meshed - Not Meshed (same namespace)	Authorized	Authorized
Not Meshed - Meshed (same namespace)	Unauthorized	Unauthorized
Meshed - Meshed (different namespace)	Unauthorized	Authorized
Meshed - Not Meshed (different namespace)	Authorized	Authorized
Not Meshed - Meshed (different namespace)	Unauthorized	Unauthorized

Source: The author.

Table 19 – Comparison between Istio and Linkerd.

Mesh	Median
No-Mesh	0.46 ms
Istio	3.57 ms
Linkerd	2.02 ms

Source: The author.

6.2 CONTRIBUTIONS

As disclosed in Section 2.6, ZT in the cloud integrated with service meshes are under-development fields with few academic works investigating them, especially how to apply and evaluate them. This research presented guidelines for verifying the security configurations are working as expected and evaluating a service mesh performance.

This study contributes to easing this scenario by detailing approaches and compromises required when adding ZT security when using service meshes. Adding security layers would naturally impact overall system performance and is thus a decision that requires sufficient thought. This research could assist in pondering these judgments.

The information presented by this research can simplify the migration to a ZT environment when using service meshes, microservices, and the cloud. It can help to decide which service mesh to select, what settings to apply, and how to verify if the configurations work as intended. These details could help to facilitate studies in this field and the industry to decide on compromises regarding security.

6.3 LIMITATIONS AND FUTURE WORKS

ZT security in the cloud is a profound subject, with diverse applications and drawbacks. This research investigated a limited part of this field and additional studies in it can contribute to the knowledge built here.

Among the limitations of this research, which could lead to interesting future work, the following can be highlighted:

- Simple application: The service used to perform an operation and to have the RTT calculated was simple, offering a low latency, which was intended, but might not be up to common production-level services. A workload including more complex activities, e.g., image rendering, and machine learning, could enable new observations.
- Only two service meshes: This research investigated Istio and Linkerd, but other alternatives are available, such as Cilium [The Cilium Authors, 2023]. Adding more service meshes would enrich the findings of this research, and thus could bring new ideas for future research.
- Limited security configurations: The configurations used to perform the investigations are but a fraction of the available for both Istio and Linkerd. Security policies ranging from request-level, cluster-level, or more could be researched. Additionally, these policies could also be used to restrict or allow access differently for more specific cases and solutions than the ones presented by this research.
- Single cloud usage: This research considered Google Cloud Platform (GCP) only, but there are cloud alternatives such as Microsoft Azure and Amazon Web Services (AWS) that could be investigated in a multi-cloud approach as done by [Rodigari et al., 2021].
- Client-to-service security: Service-to-service communication and security was the target for this research, but service meshes could also be configured to enable ZT for client-

to-service communication. Research focusing on this topic could add to the knowledge constructed by this research and discover relevant findings.

6.4 FINAL REMARKS

This Chapter has presented the results and findings of this research. It has been observed that Linkerd was faster than Istio, in the scenarios considered, with similar contributions to security. Additionally, it disclosed limitations and relevant subjects for future studies that could be based on and facilitated by this research.

REFERENCES

- [ACM, 2013] ACM (2013). A.m. turing award laureates: Leslie lamport, united states – 2013. https://amturing.acm.org/award_winners/lamport_1205376.cfm. Accessed: 2023-12-11.
- [Amaral et al., 2015] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., and Steinder, M. (2015). Performance evaluation of microservices architectures using containers. In *2015 ieee 14th international symposium on network computing and applications*, pages 27–34. IEEE.
- [Ashok et al., 2021] Ashok, S., Godfrey, P. B., and Mittal, R. (2021). Leveraging service meshes as a new network layer. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 229–236.
- [Buoyant, 2023] Buoyant (2023). The service mesh. <https://buoyant.io/service-mesh-manifesto>. Accessed: 2023-05-01.
- [Butcher, 2021] Butcher, Z. (2021). Zero trust architecture.
- [Chandramouli, 2022] Chandramouli, R. (2022). Implementation of devsecops for a microservices-based application with service mesh. *NIST Special Publication*, 800:204C.
- [Chandramouli and Butcher, 2023] Chandramouli, R. and Butcher, Z. (2023). A zero trust architecture model for access control in cloud-native applications in multi-location environments.
- [Chandramouli et al., 2021] Chandramouli, R., Butcher, Z., Chetal, A., et al. (2021). Attribute-based access control for microservices-based applications using a service mesh. *NIST Special Publication*, 800:204B.
- [Cloud Native Computing Foundation, 2022] Cloud Native Computing Foundation (2022). Service meshes are on the rise — but greater understanding and experience are required. Technical report.
- [de Weever and Andreou, 2020] de Weever, C. and Andreou, M. (2020). Zero trust network security model in containerized environments. *University of Amsterdam: Amsterdam, The Netherlands*.

-
- [Dragoni et al., 2017] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216.
- [Envoy Project Authors, 2024] Envoy Project Authors (2024). Envoy. <https://www.envoyproxy.io/>. Accessed: 2024-02-11.
- [Flexera, 2019] Flexera (2019). 2019 state of the cloud report. Technical report.
- [Flexera, 2020] Flexera (2020). 2020 state of the cloud report. Technical report.
- [Flexera, 2021] Flexera (2021). 2021 state of the cloud report. Technical report.
- [Flexera, 2022] Flexera (2022). 2022 state of the cloud report. Technical report.
- [Flexera, 2023] Flexera (2023). 2023 state of the cloud report. Technical report.
- [Google, 2024] Google (2024). What is multicloud? <https://cloud.google.com/learn/what-is-multicloud>. Accessed: 2024-02-11.
- [Kindervag et al., 2010] Kindervag, J., Balaouras, S., et al. (2010). No more chewy centers: Introducing the zero trust model of information security. *Forrester Research*, 3.
- [Koschel et al., 2021] Koschel, A., Bertram, M., Bischof, R., Schulze, K., Schaaf, M., and Astrova, I. (2021). A look at service meshes. In *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pages 1–8. IEEE.
- [Linkerd Authors, 2023a] Linkerd Authors (2023a). Architecture. <https://linkerd.io/2.13/reference/architecture/>. Accessed: 2023-05-08.
- [Linkerd Authors, 2023b] Linkerd Authors (2023b). Viz. <https://linkerd.io/2.14/reference/cli/viz/>. Accessed: 2023-09-05.
- [Linkerd Authors, 2023c] Linkerd Authors (2023c). Why linkerd doesn't use envoy. <https://linkerd.io/2020/12/03/why-linkerd-doesnt-use-envoy/>. Accessed: 2023-05-08.
- [Linkerd Authors, 2023d] Linkerd Authors (2023d). The world's most advanced service mesh. <https://linkerd.io/>. Accessed: 2023-12-22.
- [Lucas Costa, 2024] Lucas Costa (2024). Mastership. <https://github.com/lucas625/Mastership>. Accessed: 2024-02-11.

-
- [Márquez et al., 2018] Márquez, G., Villegas, M. M., and Astudillo, H. (2018). A pattern language for scalable microservices-based systems. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pages 1–7.
- [Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The nist definition of cloud computing. *National Institute of Standards and Technology*.
- [Microsoft, 2021] Microsoft (2021). Evolving zero trust. Technical report.
- [Rodigari et al., 2021] Rodigari, S., O’Shea, D., McCarthy, P., McCarry, M., and McSweeney, S. (2021). Performance analysis of zero-trust multi-cloud. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 730–732. IEEE.
- [Romani Archaman, 2018] Romani Archaman (2018). Singularity 7: Absolute Demonic Front - Babylonia. Fate/Grand Order.
- [Sermoud et al., 2021] Sermoud, G., Machado, L., Sermoud, S., and Lisboa, R. (2021). Os dez mandamentos do zero trust. Technical report.
- [Shoemaker and Estes, 2021] Shoemaker, P. and Estes, C. (2021). Accessible telemetry streams using a zero trust architecture for the flight operations directorate.
- [Stafford, 2020] Stafford, V. (2020). Zero trust architecture. *NIST special publication*, 800:207.
- [Sysdig, 2023] Sysdig (2023). What Is Minikube? <https://sysdig.com/learn-cloud-native/kubernetes-101/what-is-minikube/>. Accessed: 2023-03-01.
- [The Cilium Authors, 2023] The Cilium Authors (2023). Cilium service mesh. <https://cilium.io/use-cases/service-mesh/>. Accessed: 2023-09-15.
- [The Istio Authors, 2023a] The Istio Authors (2023a). The istio service mesh. <https://istio.io/latest/about/service-mesh/>. Accessed: 2023-03-01.
- [The Istio Authors, 2023b] The Istio Authors (2023b). The istio service mesh. <https://istio.io/>. Accessed: 2023-12-22.
- [The Istio Authors, 2023c] The Istio Authors (2023c). Prometheus. <https://istio.io/latest/docs/ops/integrations/prometheus/>. Accessed: 2023-09-05.

- [The Istio Authors, 2023d] The Istio Authors (2023d). Security. <https://istio.io/latest/docs/concepts/security/>. Accessed: 2023-03-01.
- [Thönes, 2015] Thönes, J. (2015). Microservices. *IEEE software*, 32(1):113–116.
- [Vayghan et al., 2019] Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2019). Kubernetes as an availability manager for microservice applications. *arXiv preprint arXiv:1901.04946*.
- [Wireshark Foundation, 2023] Wireshark Foundation (2023). Wireshark. <https://www.wireshark.org/>. Accessed: 2023-09-05.