Enhancing a Micro-Service Application with mTLS

Christian Frank (#473088)

April 5, 2025

Workshop Cyber Defense
Gül Sabab
FOM - Hochschule für Oekonomie & Management
WS 2024

This paper examines mutual TLS for micro-service applications and how to use a service mesh to enhance an existing application with encrypted traffic. We will compare the results using Linkerd and Istio.

Contents

List of Figures

List of Tables

List of Abbreviations

AKS	Azure Kubernetes Service
APA	American Psychological Association
API	Application Programming Interface
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
EU	European Union
IaC	Infrastructure as Code
K8s	Kubernetes
mTLS	Mutual Transport Layer Security
NIST	National Institute of Standards and Technology
NIS	Network and Information Security (Directive)
REST	Representational State Transfer
SSL	Secure Sockets Layer
TLS	Transport Layer Security

1 Introduction

1.1 Cyber Defense

Cyber defense protects computer systems and networks from theft or damage to the hardware, software, or electronic data and disruption or misdirection of their services. It is a subset of cybersecurity that focuses specifically on defending against cyberattacks.[1]

NIS2, which stands for Network and Information Systems Directive II, is the EU's legislative act to strengthen cybersecurity across the European Union. It sets strict requirements for various sectors to improve security for essential entities. These entities include organizations in critical sectors like energy, transport, waste management, healthcare, and digital infrastructure providers.[2]

One important attack vector that cyber defense must consider in this framework is network traffic to and from an application and between different components, which increases significantly when using a microservice architecture.

1.2 Micro-Service Applications

Microservice architecture involves building software applications structured as a collection of small, autonomous services modeled around a business domain.

Compared to a traditional monolithic architecture, function calls within the application can be replaced with API calls across the network. On a Kubernetes platform, this would be network traffic within the cluster.

1.3 TLS and mTLS

Transport Layer Security (TLS) is a cryptographic protocol designed to provide secure communication over a network. It is the successor to SSL and is widely used to secure web traffic, email, and other Internet protocols.

mTLS enhances TLS by adding mutual authentication. This means the client and the server must present digital certificates and verify each other's identities before establishing a secure connection.

[1]See *Gemini (2025)*: What is Cyber Security. [6]
[2]See *NIS 2 Compliant.org (2024)*: Comprehensive Guide to the NIS 2 Directive. [26]

1.4 Kubernetes

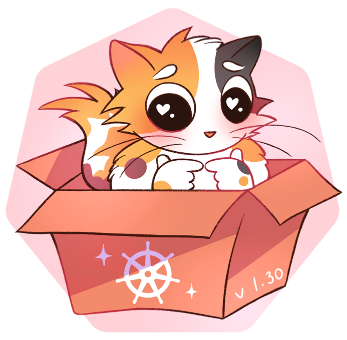
Kubernetes, or K8s, is an open-source system designed to automate deploying, scaling, and managing applications built using containers. Containers package software in a standardized unit that includes all the dependencies it needs to run, such as code, libraries, and settings. This makes them portable and efficient.

Kubernetes helps manage these containers by grouping them logically. This makes it easier to track and manage complex applications with many containers. The original inspiration for Kubernetes came from Google's internal container orchestration system, Borg.[3]

Kubernetes reached the 1.0 milestone in 2015 and was donated to the CNCF in 2016. Its current release is 1.32; we will be using 1.31 in our experiment, but 1.30 was a very special release:

"For the people who built it, for the people who release it, and for the furries who keep all of our clusters online, we present to you Kubernetes v1.30: Uwubernetes, the cutest release to date."[4]

Figure 1: Kubernetes 1.30 Release Logo



1.5 Research Question & Method

This paper will examine whether a service mesh can enhance the encryption of intra-application traffic in a sample micro-service application using mTLS.

To do this, we will perform an Experiment with a Kubernetes cluster and cross-check the findings of two Service-Mesh installations.[5]

[3]See *Gemini (2025)*: What is Kubernetes. [7]
[4]*Dsouza, A. (2024)*: Kubernetes 1.30. [4]
[5]See *Genau, L. (2020)*: Ein Experiment in Deiner Abschlussarbeit Durchführen. [10]

The goal is to establish whether Linkerd or Istio can add mTLS to an existing application and determine which uses fewer resources.

1.6 Gender-neutral Pronouns

Our society is becoming more open, inclusive, and gender-fluid, and now I think it's time to think about using gender-neutral pronouns in scientific texts, too. Two well-known researchers, Abigail C. Saguy and Juliet A. Williams, both from UCLA, propose to use the singular they/them instead: "The universal singular they is inclusive of people who identify as male, female or nonbinary."[6] The aim is to support an inclusive approach to science through gender-neutral language.

I will follow this suggestion in this paper and invite all my readers to do the same for future articles. Thank you!

If you're not sure about the definitions of gender and sex and how to use them, look at the American Psychological Association's definitions.[7]

1.7 Climate Emergency

As Professor Rahmstorf puts it: "Without immediate, decisive climate protection measures, my children currently attending high school could already experience a 3-degree warmer Earth. No one can say exactly what this world would look like—it would be too far outside the entire experience of human history. But almost certainly, this earth would be full of horrors for the people who would have to experience it."[8]

[6] *Saguy, A. (2020)*: Why We Should All Use They/Them Pronouns. [33]
[7] See *APA (2021)*: Definitions Related to Sexual Orientation. [1]
[8] *Rahmstorf, A. (2024)*: Climate and Weather at 3 Degrees More. [29]

2 TLS & mTLS

2.1 TLS

TLS stands for Transport Layer Security.[9] It's a critical internet security protocol that encrypts the connection between a user's computer and their website or service. TLS scrambles the data sent between the computer and the service, turning it into an unreadable code. This prevents hackers from intercepting and stealing information (passwords, credit card details, personal data, etc.). TLS also verifies that the service is legitimate and not a fake one trying to steal information; it ensures that the data sent between computers and services has not been tampered with or altered during transmission.[10]

TLS is the foundation of secure online communication. Without it, data would be vulnerable to interception and theft. TLS helps keep online activity private, preventing eavesdropping and unauthorized access to information. TLS certificates, or SSL certificates, are a sign of confidence. Websites with TLS encryption display a padlock icon in the browser, reassuring users that their connection is secure.

TLS uses a combination of symmetric and asymmetric cryptography. Symmetric cryptography encrypts the data, while asymmetric cryptography authenticates the server and exchanges the symmetric encryption key.[11]

TLS is an essential part of internet security. It helps to protect sensitive information from being intercepted and stolen. SSL was the predecessor to TLS, which is more modern and secure. While the terms are sometimes interchangeable, TLS is the more modern protocol.

TLS is essential for protecting privacy and security online. Its technology allows for secure online transactions, browsing, and communication.

2.2 mTLS

Mutual TLS (mTLS) is a type of authentication that requires both the client and the server to authenticate each other. This contrasts with traditional TLS, which only requires the server to authenticate itself to the client.[12]

mTLS provides additional protection by ensuring that the client and the server are who they claim to be. This can help prevent man-in-the-middle attacks and other security

[9]See *NIST (2011)*: Glossary. [27]
[10]See *Internet Society (2025)*: TLS Basics. [15]
[11]See *ENTRUST (2025)*: What is TLS. [5]
[12]See *Google Cloud (2025)*: Mutual TLS overview. [11]

threats. mTLS can authenticate users and devices, which can be helpful in various situations, such as accessing sensitive data or applications. mTLS can also help protect the confidentiality of user data by ensuring that only authorized parties can access it.

mTLS is very important in securing APIs by ensuring only authorized clients can access them. It can also protect sensitive data, such as financial or medical information, and authenticate users and devices when they access sensitive resources.

mTLS is a valuable security tool for protecting sensitive data and applications. It is beneficial when high levels of security and authentication are required, such as when the NIS2 requirements need to be met.

2.3 Microservice Architecture

Microservice architecture is a software development approach in which an application is structured as a collection of small, autonomous services modeled around a business domain. It can be compared to a bustling city comprising specialized districts (aka services) that work together to make the city function. Each district has its infrastructure, team, and way of doing things, but they all contribute to the overall city experience.[13]

Each microservice is designed to perform a specific business function in a micro-service application. They are small enough to be developed, deployed, and maintained independently. Microservices operate independently. A failure in one service does not bring down the entire application. Microservices can also be updated or deployed without affecting other services.

They are decentralized, with no central database or monolithic code base. Each microservice can choose the technology stack (programming language, database, etc.) best suited for its specific task, allowing for greater flexibility and innovation.

Microservices communicate with each other through well-defined APIs, often using lightweight protocols like REST or message queues. This loose coupling minimizes dependencies between services, making them more independent. Microservices can be deployed and scaled independently. For example, if one service is experiencing high traffic, it can be scaled to increase its capacity without scaling the entire application.

Microservices are a good fit for large, complex applications. They can break down complexity into smaller, manageable parts and application parts that can scale independently. They are also well-suited for applications with varying workloads.

[13]See *Google Cloud (2025)*: What Is Microservices Architecture. [13]

Microservices generally follow an API-first design. They communicate through well-defined APIs, which act as contracts between services and ensure interoperability. A common choice is RESTful APIs using HTTP.

Microservices offer a powerful approach to building complex applications but require careful planning and execution. Understanding the trade-offs and challenges are crucial for successful implementation.

2.4 Service Mesh

A service mesh is an infrastructure layer between application services and the underlying network. It is designed to make communication between microservices easier, more reliable, and secure and has a dedicated traffic management and security layer. It handles all the complexities of inter-service communication, allowing the application code to focus on business logic.[14]

A Service Mesh controls how traffic flows between services. This includes routing, load balancing, retries, timeouts, and fault injection (testing resilience). It can intelligently route requests based on various criteria, such as versioning, A/B testing, or canary deployments. It secures communication between services. This includes mutual TLS (mTLS) authentication, authorization, and encryption, which we will analyze later. With mTLS, it ensures that only authorized services can communicate with each other and that all communication is encrypted.

A Service Mesh also provides insights into the behavior of microservices. This includes metrics, tracing, and logging. It allows for monitoring the health and performance of all services in the mesh and quickly identifying any issues.

A service mesh typically uses a "sidecar" proxy pattern. This means a small proxy (the sidecar) is deployed alongside each microservice instance in the container. All traffic to and from the microservice flows through this sidecar proxy, which is managed by a control plane configuring the mesh and collecting telemetry data.

A service mesh is a powerful tool for managing and securing communication between microservices. It can simplify development, improve security and resilience, and enhance observability. However, it also adds complexity to the application landscape, so it's essential to carefully consider the trade-offs before implementing one.

[14]See *Hamilton, C. (2024)*: What is a service mesh. [14]

2.5 Benefits of a Service Mesh

Microservices often need to communicate with each other, which can involve intricate tasks like service discovery, load balancing, and handling network issues. A service mesh abstracts these complexities away, allowing developers to focus on the core logic of their microservices. It provides a uniform way for services to communicate, regardless of the language or framework in which they are written. This ensures consistency and simplifies development.[15]

A service mesh can enforce mTLS, encrypting communication between services and verifying the identity of both the client and the server. This significantly strengthens security and prevents unauthorized access. It also allows for defining and enforcing security policies across the entire microservices ecosystem from a central point, simplifying security management.

A service mesh provides detailed insights into the behavior of the microservices, including metrics, tracing, and logging. This makes it easier to monitor their health and performance and identify issues.

Service meshes offer features like circuit breaking, retries, and timeouts, which make microservices more resilient to failures. This helps prevent cascading failures and ensures your application remains available even when some services are down.

Thus, a service mesh acts as a dedicated infrastructure layer for the microservices landscape, handling the complexities of inter-service communication, security, observability, and resilience. This allows developers to focus on building great applications while operations teams can effectively manage and monitor the microservices ecosystem.

2.6 Benefits of mTLS

Microservices often communicate over a network, making them vulnerable to interception and eavesdropping. mTLS enforces a "zero trust" model, where every service must authenticate itself regardless of its location within the network. This is critical in a microservice environment where services might be deployed across different environments or clouds.[16]

mTLS verifies the digital certificates of the client and server to ensure that they are who they claim to be. This prevents unauthorized services from impersonating legitimate ones, a key concern in a distributed microservice system. mTLS also encrypts all

[15]See *Gemini (2025)*: What are Microservices. [8]
[16]See *Patil, K. (2023)*: Why Mutual TLS is critical. [28]

communication between services, protecting sensitive data from being intercepted and read. This is paramount in microservices, as data might travel through various network segments.

mTLS establishes a straightforward and automated way for services to trust each other. No complex key management or manual configuration is needed for each service interaction. The service mesh (or other infrastructure) handles certificate distribution and validation. Each service will have a verifiable identity. This simplifies auditing, logging, and access control.

mTLS makes Man-in-the-Middle attacks significantly harder. To intercept communication, an attacker would need to possess the private key of a legitimate service. Also, because each service is authenticated, it's much more challenging for a malicious service to "spoof" or impersonate a legitimate one.

Many industries have strict data security and privacy regulations. mTLS can help organizations meet these requirements by providing strong authentication and encryption of inter-service communication. This is increasingly important as microservices handle increasingly sensitive data.

Microservices' distributed nature increases their attack surface and mTLS helps secure the many communication points between services. Microservices are often deployed and scaled dynamically, and mTLS provides an automated way to handle security in this fluid environments. Microservices rely heavily on each other. If one service is compromised, it can potentially expose others, and mTLS limits a compromised service's "blast radius" by ensuring that communication with other services is always secured.

mTLS is a fundamental security practice for microservice architectures. It provides strong authentication and encryption and simplifies security management, making it an essential tool for protecting sensitive data and ensuring the integrity of inter-service communication. When integrated with a service mesh, mTLS becomes even more powerful and easier to manage.

2.7 Linkerd

Linkerd is a lightweight and straightforward service mesh designed for Kubernetes. It manages and secures communication between microservices, providing a dedicated traffic management and security layer without the complexity associated with service meshes.[17]

Linkerd's minimalist design focuses on providing essential service mesh functionalities

[17]See *Linkerd (2025)*: Why Linkerd. [24]

without unnecessary complexity. This makes installing, configuring, and operating easier than other service meshes. Linkerd is known for its straightforward installation process and user-friendly CLI.[18]

Linkerd uses a purpose-built, ultra-lightweight proxy written in Rust and is designed to have a minimal performance impact on the applications. It aims to be as efficient as possible, adding little overhead to service communication.

Linkerd enables mTLS by default, which encrypts communication between services and ensures that only authorized services can communicate. It promotes a zero-trust security model in which every service must authenticate itself regardless of its location within the network.[19]

Linkerd is a service mesh focusing on simplicity, performance, and security. It's a good option for teams that want a lightweight and easy-to-use service mesh for their Kubernetes deployments.

2.8 Istio

Istio is a popular, open-source service mesh that sits atop a container orchestration platform like Kubernetes. It helps connect, secure, control, and observe microservices; it is a smart infrastructure layer that manages communication between services.[20]

Istio offers fine-grained control over traffic flow between your microservices. This includes Routing, Load Balancing, Fault Injection, and Traffic Splitting.

Istio provides robust security features, such as Mutual TLS (mTLS), Authorization and Authentication. Istio also allows you to define and enforce policies across all microservices, such as Rate Limiting and Quota Management.[21]

Istio uses a "sidecar" proxy pattern. A lightweight proxy (typically Envoy) is deployed alongside each microservice instance. All traffic to and from the microservice flows through this sidecar proxy. Istio's control plane manages these sidecar proxies, configuring them with the desired traffic management, security, and policy rules.

Key Components of Istio:

- Pilot: The primary component for traffic management. It configures the Envoy proxies based on the traffic rules you define

[18]See *Gemini (2025)*: What is Linkerd. [8]
[19]See *Morgan, W. (2024)*: Zero trust network security in Kubernetes. [25]
[20]See *Google Cloud (2025)*: What Is Istio. [12]
[21]See *Istio (2025)*: Security. [19]

- Citadel: Handles security, including certificate management for mTLS
- Galley: Manages Istio's configuration.

Istio is most beneficial for complex microservices architectures and applications that require high levels of security and resilience.

Istio is a powerful and feature-rich service mesh that can significantly simplify the management and security of microservices. However, it also adds complexity to the infrastructure, so it's important to consider whether it's the right solution.

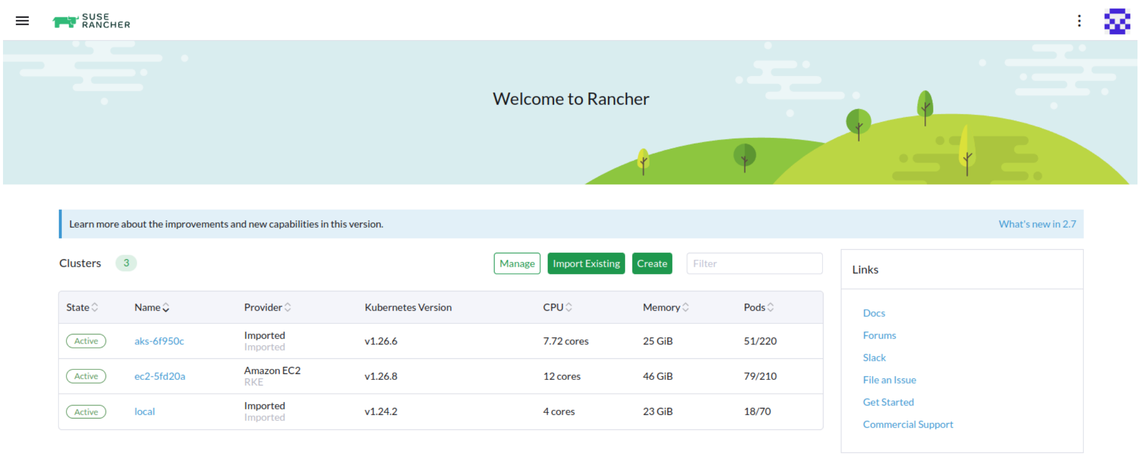
2.9 Rancher

To easily manage the service mesh installation on a Kubernetes cluster, we can turn to SUSE Rancher.

What is Rancher? The Rancher Labs website states it is "[...] a complete software stack for teams adopting containers. It addresses the operational and security challenges of managing multiple Kubernetes clusters while providing DevOps teams with integrated tools for running containerized workloads."[22]

Using a user-friendly GUI, Rancher provides a management platform for centrally managing multiple Kubernetes clusters in Enterprise IT. It also offers application development integration tools and robust enterprise-grade security and governance features. For operations, Rancher provides integrated solutions for logging, monitoring, and auditing, as well as many other features, such as CIS scans or a built-in service mesh.

Figure 2: Rancher Dashboard



Learn more about the improvements and new capabilities in this version. What's new in 2.7

Clusters 3

State	Name	Provider	Kubernetes Version	CPU	Memory	Pods
Active	aks-6f950c	Imported Imported	v1.26.6	7.72 cores	25 GiB	51/220
Active	ec2-5fd20a	Amazon EC2 RKE	v1.26.8	12 cores	46 GiB	79/210
Active	local	Imported Imported	v1.24.2	4 cores	23 GiB	18/70

Links

- Docs
- Forums
- Slack
- File an Issue
- Get Started
- Commercial Support

[22] *Rancher Labs (2025)*: Enterprise Kubernetes Management. [32]

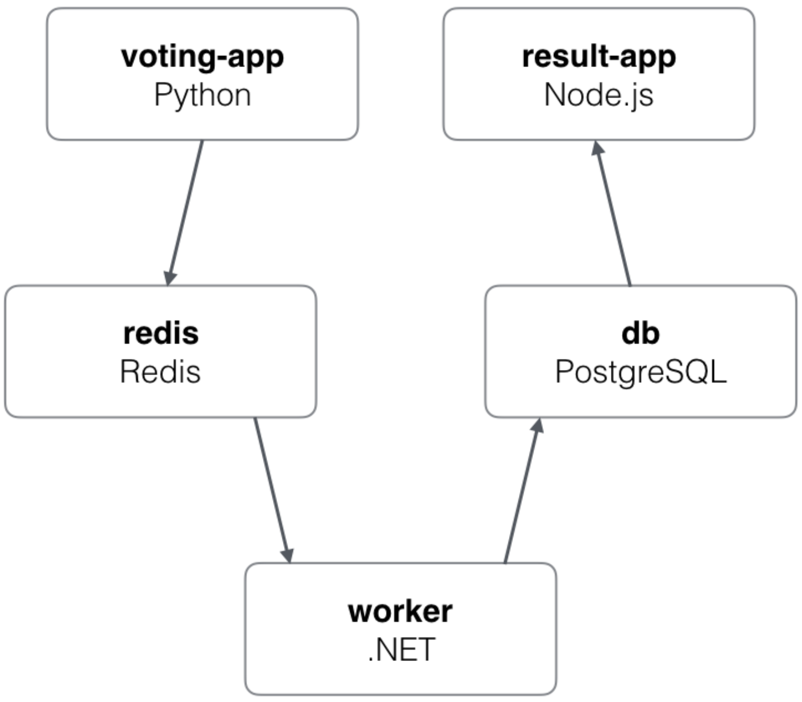
3 mTLS Exploration

3.1 Sample Voting Application

To evaluate mTLS and service mesh, we will use a simple application, the example-voting-app from the official Docker Samples; it's a simple distributed application running across multiple Docker containers, and we will follow Sathish Kumar's excellent post about deploying it on Kubernetes using slightly modified deployment files.[23]

The application has the following architecture:[24]

Figure 3: Application Architecture



It consists of the following components:

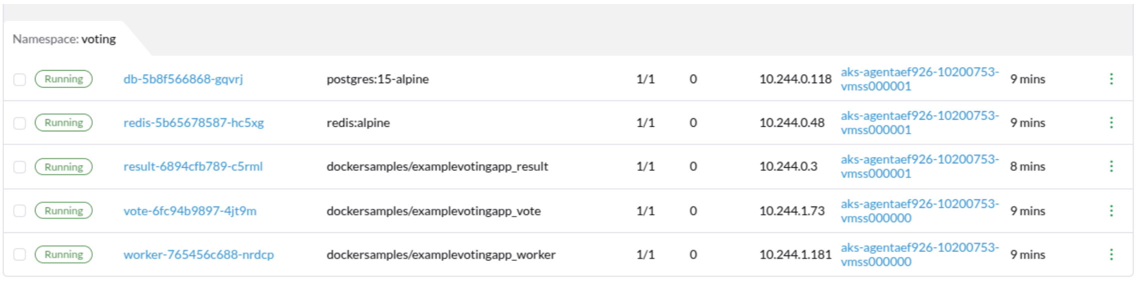
- A front-end web app
- A Redis database collecting new votes

[23]See *Kumar, S. (2021)*: Deploying a sample microservices app with Kubernetes. [21]
[24]See *Kumar, S. (2021)*: Ibid. [21]

- A worker consuming votes and storing them
- A Postgres database
- A Node.js web app showing the results[25]

After deploying the application, these components will run inside the Kubernetes cluster, each as an individual pod:

Figure 4: Voting Application Components



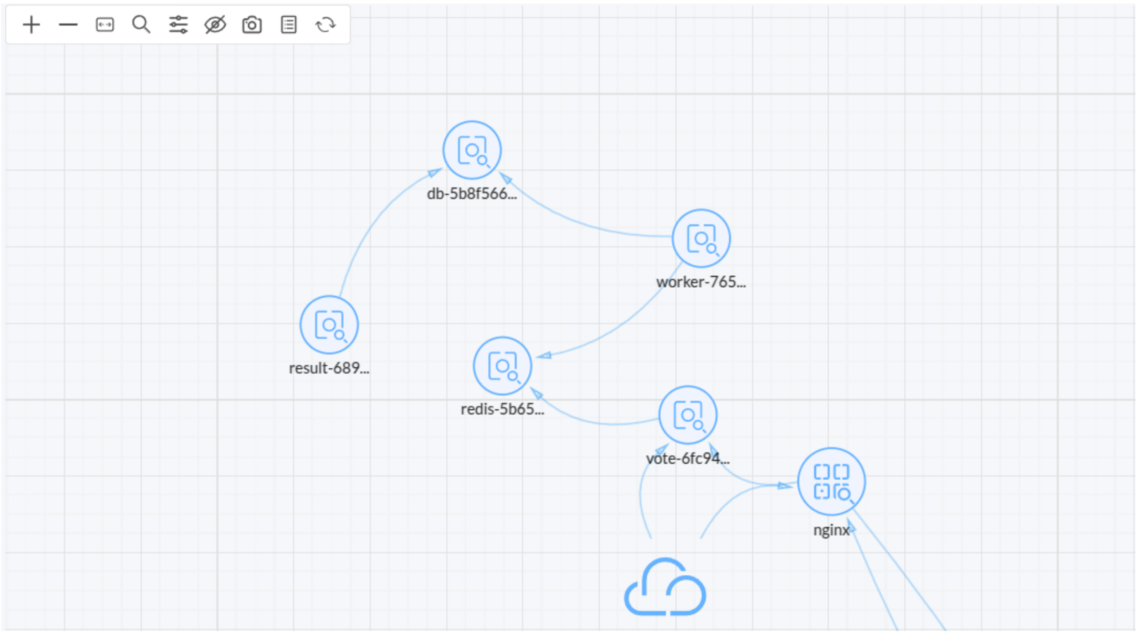
Namespace: voting

Running	db-5b8f566868-gqvrj	postgres:15-alpine	1/1	0	10.244.0.118	aks-agentaef926-10200753-vmss000001	9 mins	⋮
Running	redis-5b65678587-hc5xg	redis:alpine	1/1	0	10.244.0.48	aks-agentaef926-10200753-vmss000001	9 mins	⋮
Running	result-6894cfb789-c5rml	dockersamples/examplevotingapp_result	1/1	0	10.244.0.3	aks-agentaef926-10200753-vmss000001	8 mins	⋮
Running	vote-6fc94b9897-4jt9m	dockersamples/examplevotingapp_vote	1/1	0	10.244.1.73	aks-agentaef926-10200753-vmss000000	9 mins	⋮
Running	worker-765456c688-nrdcp	dockersamples/examplevotingapp_worker	1/1	0	10.244.1.181	aks-agentaef926-10200753-vmss000000	9 mins	⋮

The YAML files used for the application deployment are on the paper's Github.

Using Rancher's security component Neuvector, we can examine the network connections of the application components:

Figure 5: Voting Application Connections



[25]See *Docker (2024)*: Example Voting App. [3]

- Outside communication will reach the vote front-end, initially through the AKS ingress controller
- The vote component will connect to the Redis in-memory database
- The worker component will connect to both the Redis and the Postgres database
- The result component will read from the Postgres database

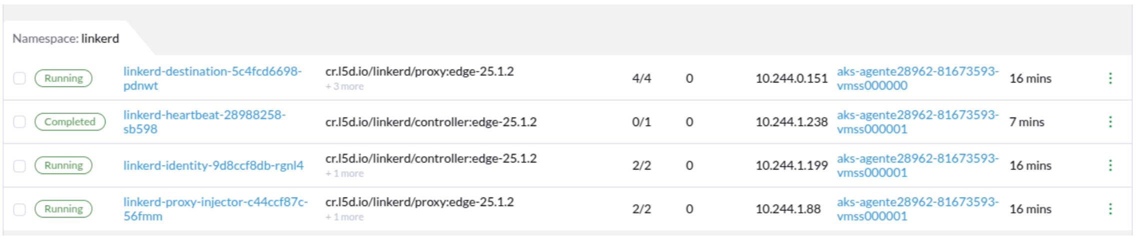
The arrows indicate the direction in which the connections were established. Without any installed service mesh, all connections are unencrypted. The observed communication relationships match the application documentation.

3.2 Linkerd

To install Linkerd edge-25.1.2, we follow the instructions provided by SUSE[26] and the reference documentation provided by Linkerd.[27]

After installation, the Linkerd control plane adds four pods to the running cluster.

Figure 6: Linkerd Components



Namespace: linkerd

☐	Running	linkerd-destination-5c4fcd6698-pdnwt	cr.l5d.io/linkerd/proxy:edge-25.1.2 + 3 more	4/4	0	10.244.0.151	aks-agente28962-81673593-vmss000000	16 mins	⋮
☐	Completed	linkerd-heartbeat-28988258-sb598	cr.l5d.io/linkerd/controller:edge-25.1.2	0/1	0	10.244.1.238	aks-agente28962-81673593-vmss000001	7 mins	⋮
☐	Running	linkerd-identity-9d8ccf8db-rgnl4	cr.l5d.io/linkerd/controller:edge-25.1.2 + 1 more	2/2	0	10.244.1.199	aks-agente28962-81673593-vmss000001	16 mins	⋮
☐	Running	linkerd-proxy-injector-c44ccf87c-56fmm	cr.l5d.io/linkerd/proxy:edge-25.1.2 + 1 more	2/2	0	10.244.1.88	aks-agente28962-81673593-vmss000001	16 mins	⋮

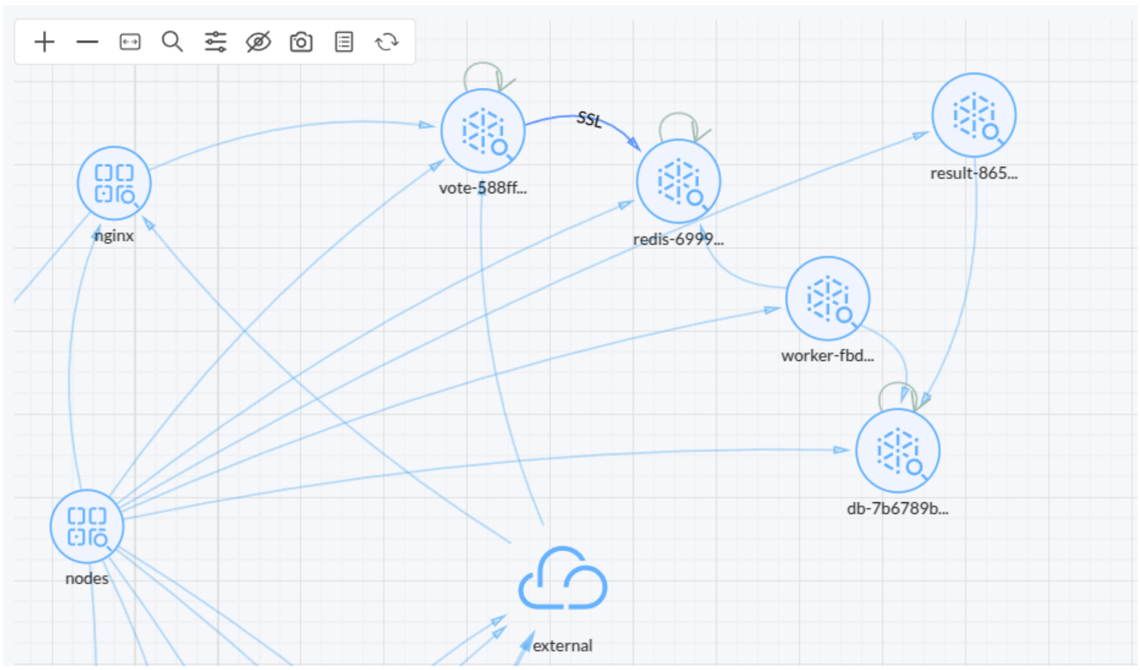
Using the Linkerd CLI, we inject our voting application with the service mesh information.

```
kubectl get -n voting deploy -o yaml \
| linkerd inject - \
| kubectl apply -f -
```

Linkerd will then add a proxy sidecar to each application pod to enable the service mesh functionality and encrypt the application traffic.

[26]See *Dayley, B. (2021)*: End-to-end Encryption with Linkerd. [2]
[27]See *Linkerd (2025)*: Getting Started. [22]

Figure 7: Linkerd Application Connections



The network diagram shows that the traffic between the vote and the Redis component is now encrypted with TLS; this is also true for all the other network connections within the application. The changed deployment icon indicates the added sidecar container and the new network connection inside the pod.

In the default installation, only the traffic is encrypted. Linkerd offers the capability to encrypt the traffic and add authentication and authorization using Linkerd policies.[28]

This paper will only look at mTLS encryption and leave authentication for future investigations.

3.3 Istio

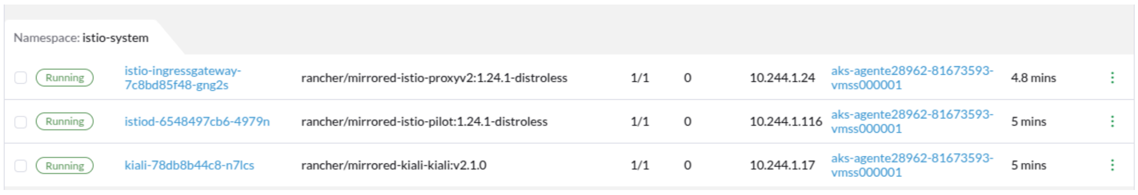
To install Istio 1.24.1, we again follow the instructions provided by SUSE.[29] As with Linkerd, enabling mTLS for applications in the mesh will be automatic.[30]

Istio requires the Rancher Monitoring application or any other Prometheus deployment to be installed on the cluster first.[31]

[28]See *Linkerd (2025)*: Restricting Access To Services. [23]
[29]See *Rancher Labs (2025)*: Enable Istio in the Cluster. [30]
[30]See *Istio (2025)*: mTLS in Istio. [18]
[31]See *Rancher Labs (2025)*: Enable Monitoring. [31]

After installation, the Istio adds three pods to the running cluster.

Figure 8: Istio Components



Namespace: istio-system

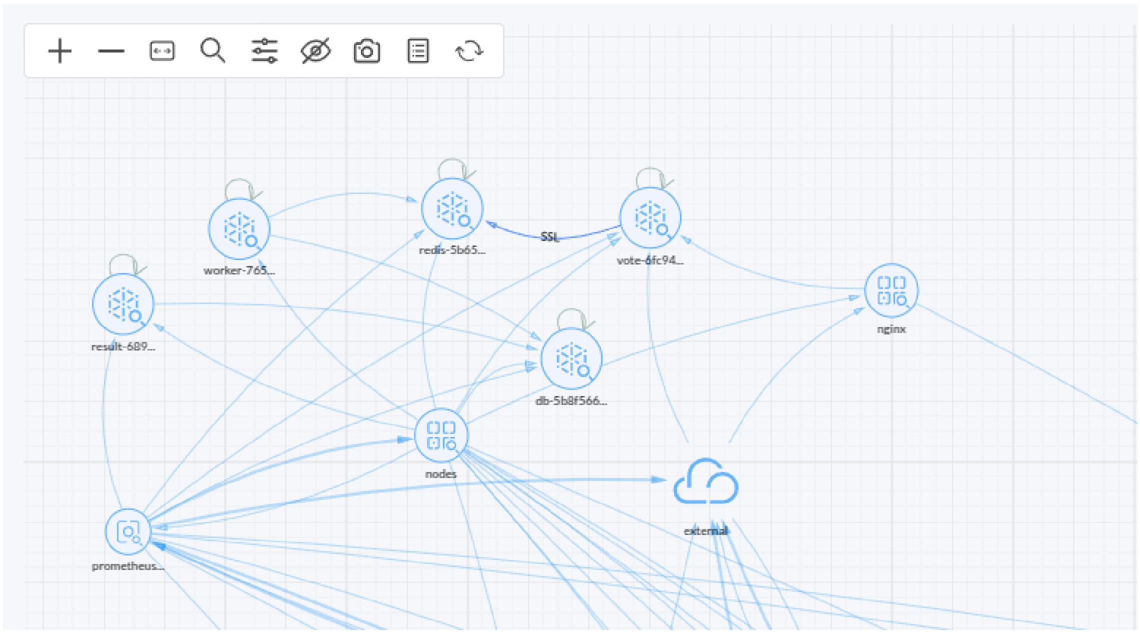
☐	Running	istio-ingressgateway-7c8bd85f48-gng2s	rancher/mirrored-istio-proxyv2:1.24.1-distroless	1/1	0	10.244.1.24	aks-agente28962-81673593-vmss000001	4.8 mins	⋮
☐	Running	istiod-6548497cb6-4979n	rancher/mirrored-istio-pilot:1.24.1-distroless	1/1	0	10.244.1.116	aks-agente28962-81673593-vmss000001	5 mins	⋮
☐	Running	kiali-78db8b44c8-n7lcs	rancher/mirrored-kiali-kiali:v2.1.0	1/1	0	10.244.1.17	aks-agente28962-81673593-vmss000001	5 mins	⋮

For Istio, we use the auto-inject feature on the application namespace to inject our voting application with the service mesh information.

```
"kubernetes_namespace": {
  "name": "voting",
  "labels": {
    "field.cattle.io/projectId": "p-g89q2",
    "istio-injection": "enabled",
    "kubernetes.io/metadata.name": "voting"
  }
}
```

Istio will then add a proxy sidecar to each application pod during deployment in this namespace to enable the service mesh functionality and encrypt the application traffic.

Figure 9: Istio Application Connections



As with Linkerd, the network diagram now shows that the traffic between the vote and the Redis component is encrypted with TLS. Compared to Linkerd, there are a number of additional connections due to the requirement for metrics collection with Prometheus.

Istio, like Linkerd, also offers the option to authenticate connections.

4 mTLS Analysis

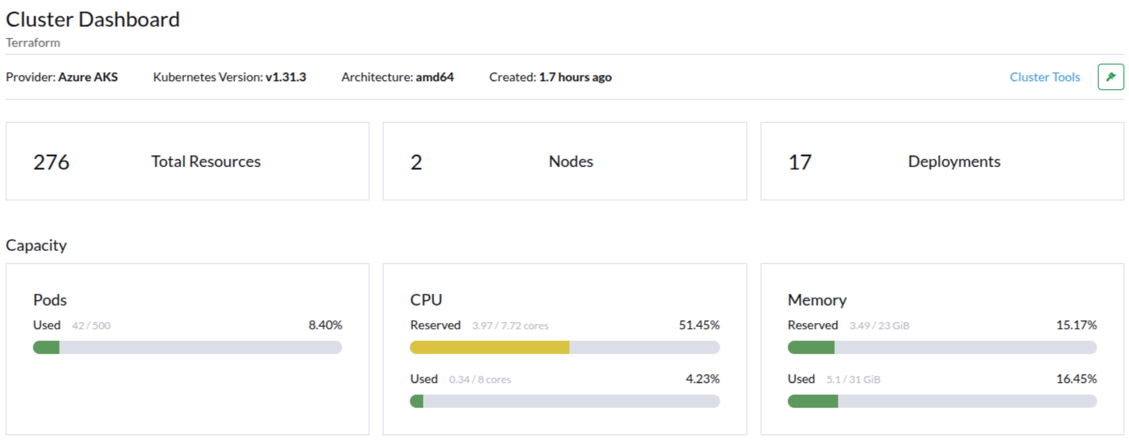
4.1 Resource Consumption

To evaluate the two service meshes, we will look at the following Kubernetes metrics:[32]

- Number of pods
- CPU reserved
- CPU used
- Memory reserved
- Memory used

We will take the values from the Rancher cluster dashboard for each experiment iteration.

Figure 10: Cluster Dashboard



Cluster Dashboard
Terraform

Provider: **Azure AKS** Kubernetes Version: **v1.31.3** Architecture: **amd64** Created: **1.7 hours ago** Cluster Tools

276 Total Resources
2 Nodes
17 Deployments

Capacity

Pods
Used 42 / 500 8.40%

CPU
Reserved 3.97 / 7.72 cores 51.45%
Used 0.34 / 8 cores 4.23%

Memory
Reserved 3.49 / 23 GiB 15.17%
Used 5.1 / 31 GiB 16.45%

Here are the tabulated results from the experiment:

Table 1: Resource Consumption

	Pods	CPU Rsvd	CPU Used	Memory Rsvd	Memory Used
Idle	37	3.97 cores	0.32 cores	3.49 GB	4.78 GB
App Only	42	3.97 cores	0.34 cores	3.49 GB	5.1 GB
Linkerd	59	3.97 cores	0.61 cores	3.49 GB	13 GB
Istio	61	5.93 cores	1.02 cores	7.14 GB	13 GB

[32]See *Kubernetes (2023)*: Resource Pipeline. [20]

4.2 Installation Time

Installing the two service meshes was not very time-consuming. We measured the time from the beginning of the first command until after the Helm chart was finished installing the software.

Table 2: Installation Time

	Time
Linkerd	5 min
Istio	20 min

Istio requires several prerequisites, such as a running monitoring stack, which makes the installation significantly longer.

4.3 Enabling mTLS

The documentation made enabling mTLS for our sample application straightforward, either from the CLI or through auto-injection. We measured the time it took to enable mTLS on the installed sample application.

Table 3: Enabling mTLS

	Time
Linkerd	5 min
Istio	5 min

There was no difference in time or effort between Istio and Linkerd regarding enabling mTLS for our sample application.

4.4 Linkerd and Istio Evaluation

From our experiment, we can derive the following conclusions:

- Istio uses more resources and takes longer to deploy
- Istio and Linkerd both offer automatic traffic encryption

Further findings include:

- Istio enables Metrics by default and includes its own observability console, Kiali.[33]
- Istio uses the Envoy proxy, a well-established industry standard

[33]See *Istio (2025)*: Kiali. [17]

- Istio is available for Kubernetes clusters but can support virtual machines and bare metal environments
- Linkerd uses its own linkerd-2 proxy
- Linkerd focuses on Kubernetes clusters only

Istio offers very powerful traffic management capabilities, allowing for fine-grained control over how traffic flows through the service mesh.

Linkerd, on the other hand, is designed to be easier to install, configure, and manage than Istio. Its resource consumption above shows it's more lightweight and has a more negligible performance overhead.

In summary, Istio offers a robust and comprehensive solution for complex microservices architectures, while Linkerd prioritizes simplicity and ease of use for less demanding environments. For a complex microservices architecture with demanding traffic management and security requirements, Istio might be the better choice.

Both service meshes handle application traffic encryption equally well.

4.5 Outlook

So far, we've only looked at application traffic encryption. Both service meshes also allow authentication policies to restrict traffic between applications and arrive more at a zero-trust network security model.[34]

Other options to explore in a future paper include Istio's Ingress and Egress gateways, which allow network policies to extend across clusters and include non-Kubernetes resources. Similarly to Istio, Linkerd also offers a multi-cloud feature.

Both service mesh installations that we have looked at in this paper use the sidecar injection pattern. Istio now offers with Ambient mode an option to run a service mesh without sidecars, which could be worthwhile to explore, too.[35]

[34]See *Gemini (2025)*: What is Zero-Trust Networking. [9]
[35]See *Istio (2024)*: Istio's Ambient Mode. [16]

5 Summary

There are two ways to enable mTLS for an application. One is refactoring the application and including mutual encryption in all API calls. The other is to use a Service Mesh, such as Linkerd or Istio, to encrypt the intra- and inter-application traffic.

In this paper, we used a sample application with Istio and Linkerd. We found that both service meshes provide the desired functionality, and we added mTLS to the application without requiring application modification.

Linkerd uses less overhead than Istio, but both service meshes can easily enhance a microservice application with mutual TLS.

Other service meshes are available, most notably the Cilium Service Mesh and Consul by Hashicorp.

We can conclude that using a service mesh is a viable alternative to refactoring the code to encrypt application traffic. Linkerd and Istio can add mTLS to an existing application, and we found that Linkerd uses fewer resources.

The Terraform plan files for the Kubernetes cluster used in this paper are on my GitHub.

Happy Ranching!

References

[1] APA. (2021) Definitions related to sexual orientation. [Access 2021-04-06]. [Online]. Available: https://www.apa.org/pi/lgbt/resources/sexuality-definitions.pdf

[2] B. Dayley. (2021) End-to-end encryption with linkerd. [Access 2025-02-08]. [Online]. Available: https://www.suse.com/c/end-to-end-encryption-for-your-rancher-cluster-with-linkerd/

[3] Docker. (2024) Example voting app. [Access 2025-02-08]. [Online]. Available: https://github.com/dockersamples/example-voting-app

[4] A. Dsouza and K. Martin. (2024) Kubernetes v1.30. [Access 2025-02-08]. [Online]. Available: https://kubernetes.io/blog/2024/04/17/kubernetes-v1-30-release/

[5] ENTRUST. (2025) What is tls. [Access 2025-02-08]. [Online]. Available: https://www.entrust.com/resources/learn/what-is-tls

[6] Gemini. (2025) What is cyber security. [Access 2025-02-08]. [Online]. Available: https://gemini.google.com

[7] Gemini. (2025) What is kubernetes. [Access 2025-02-08]. [Online]. Available: https://gemini.google.com

[8] Gemini. (2025) What is linkerd. [Access 2025-02-08]. [Online]. Available: https://gemini.google.com

[9] Gemini. (2025) What is zero-trust networking. [Access 2025-02-11]. [Online]. Available: https://gemini.google.com

[10] L. Genau. (2022) Ein experiment in deiner abschlussarbeit durchführen. [Access 2025-02-08]. [Online]. Available: https://www.scribbr.de/methodik/experiment/

[11] Google Cloud. (2025) Mutual tls overview. [Access 2025-02-08]. [Online]. Available: https://cloud.google.com/load-balancing/docs/mtls

[12] Google Cloud. (2025) What is istio. [Access 2025-02-08]. [Online]. Available: https://cloud.google.com/learn/what-is-istio

[13] Google Cloud. (2025) What is microservices architecture. [Access 2025-02-08]. [Online]. Available: https://cloud.google.com/learn/what-is-microservices-architecture

[14] C. Hamilton. (2024) What is a service mesh. [Access 2025-02-08]. [Online]. Available: https://www.dynatrace.com/news/blog/what-is-a-service-mesh/

[15] Internet Society. (2025) Tls basics. [Access 2025-02-08]. [Online]. Available: https://www.internetsociety.org/deploy360/tls/basics/

[16] Istio. (2025) Istio's ambient mode. [Access 2025-02-11]. [Online]. Available: https://istio.io/latest/blog/2024/ambient-reaches-ga/

[17] Istio. (2025) Kiali. [Access 2025-02-11]. [Online]. Available: https://istio.io/latest/docs/ops/integrations/kiali/

[18] Istio. (2025) mtls in istio. [Access 2025-02-11]. [Online]. Available: https://istio.io/latest/blog/2023/secure-apps-with-istio/#mtls-in-istio

[19] Istio. (2025) Security. [Access 2025-02-08]. [Online]. Available: https://istio.io/latest/docs/concepts/security/

[20] Kubernetes. (2025) Resource pipeline. [Access 2025-02-10]. [Online]. Available: https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/

[21] S. Kumar. (2021) Deploying a sample microservices app with kubernetes. [Access 2025-02-08]. [Online]. Available: https://www.sysspace.net/post/deploying-docker-voting-app-with-kubernetes

[22] Linkerd. (2025) Getting started. [Access 2025-02-11]. [Online]. Available: https://linkerd.io/2.17/getting-started/

[23] Linkerd. (2025) Restricting access to services. [Access 2025-02-11]. [Online]. Available: https://linkerd.io/2.17/tasks/restricting-access/

[24] Linkerd. (2025) Why linkerd. [Access 2025-02-08]. [Online]. Available: https://linkerd.io/

[25] W. Morgan. (2024) Zero trust network security in kubernetes. [Access 2025-02-08]. [Online]. Available: https://buoyant.io/zero-trust-in-kubernetes-with-linkerd

[26] NIS 2 Compliant.org. (2024) Comprehensive guide to the nis 2 directive. [Access 2025-02-08]. [Online]. Available: https://nis2compliant.org/

[27] NIST. (2011) Glossary. [Access 2025-02-08]. [Online]. Available: https://csrc.nist.gov/glossary/term/tls

[28] K. Patil. (2023) Why mutual tls is critical. [Access 2025-02-08]. [Online].

Available: https://www.appviewx.com/blogs/
why-mutual-tls-mtls-is-critical-for-securing-microservices-communications-in-a-service-mesh/

[29] S. Rahmstorf, *Climate and Weather at 3 Degrees More*. Cham: Springer Nature Switzerland, 2024, pp. 3–17.

[30] Rancher Labs. (2025) Enable istio in the cluster. [Access 2025-02-11]. [Online]. Available: https://ranchermanager.docs.rancher.com/how-to-guides/advanced-user-guides/istio-setup-guide/enable-istio-in-cluster

[31] Rancher Labs. (2025) Enable monitoring. [Access 2025-02-11]. [Online]. Available: https://ranchermanager.docs.rancher.com/how-to-guides/advanced-user-guides/monitoring-alerting-guides/enable-monitoring

[32] Rancher Labs. (2025) Enterprise kubernetes management. [Access 2025-02-08]. [Online]. Available: https://rancher.com/

[33] A. Saguy and J. Williams. (2020) Why we should all use they/them pronouns. [Access 2025-02-08]. [Online]. Available: https://blogs.scientificamerican.com/voices/why-we-should-all-use-they-them-pronouns/