Universitat Oberta de Catalunya

uoc.edu

Implementación de protocolos seguros en arquitecturas Service Mesh con Istio

Joseba García Velasco
Máster Universitario en Ciberseguridad y Privacidad
M1.847 - Sistemas de autenticación y autorización_MISTIC

Nombre Consultor/a: Angel Linares Zapater
Nombre Profesor/a responsable de la asignatura: Víctor García Font

Fecha Entrega: 31 de mayo de 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación de protocolos seguros en arquitecturas Service Mesh con Istio*
Nombre del autor:	*Joseba García Velasco*
Nombre del consultor/a:	*Ángel Linares Zapater*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	05/2022
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*M1.847 - Sistemas de autenticación y autorización_MISTIC*
Idioma del trabajo:	*Castellano*
Palabras clave	*Service Mesh, Istio, mTLS, https, autenticación, autoridad certificadora, CA, autorización, kubernetes, contenedores, microservicios, segmentación*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Este TFM persigue mostrar una solución a la necesidad de diseñar aplicaciones de forma que se puedan separar las funciones de implementación de la lógica de negocio (qué funcionalidad tiene que cubrir la aplicación) de la lógica de implementación de servicios de seguridad (qué controles de seguridad tengo que dotar a la solución). Istio aborda esta solución en arquitecturas basadas en microservicios sobre Kubernetes.

Para ello, se aborda la implementación de una solución basada en una arquitectura de microservicios apoyada en una solución que añade controles de seguridad. A esta solución se le dotará de capacidades de seguridad, orientadas al control del tráfico de comunicaciones entre los servicios. Concretamente la solución tendrá implantadas capacidades propias de la herramienta Istio, sobre una plataforma de Kubernetes. Estas capacidades están basadas en controles de seguridad de autenticación y autorización, que tienen una base en el protocolo mTLS. Concretamente estas funciones a implantar serán la habilitación del protocolo mTLS en el tráfico específico entre los servicios de la solución y la habilitación de tráfico http sobre este protocolo mTLS, consiguiendo de esta forma dotar de un protocolo de comunicaciones seguro en la arquitectura.

En la fase de implementación se presenta una aplicación web con una serie de microservicios sobre los que se implantarán las capacidades de mTLS y comunicación vía http.

Abstract (in English, 250 words or less):

This final work of head seeks to show a solution to the need to design applications in such a way that the implementation functions of the business logic can be separated (what functionality the application must cover) from the security service implementation logic (what security controls I must equip the solution). Istio approaches this solution in architectures based on microservices on top of Kubernetes.

To do this, the implementation of a solution based on a microservices architecture supported by a solution that adds security controls is addressed. This solution will be provided with security capabilities, aimed at controlling communications traffic between services. Specifically, the solution will have the capabilities of the Istio tool implemented, on a Kubernetes platform. These capabilities are based on authentication and authorization security controls, which are based on the mTLS protocol. Specifically, these functions to be implemented will be the enabling of the mTLS protocol in the specific traffic between the services of the solution and the enabling of http traffic on this mTLS protocol, thus achieving a secure communications protocol in the architecture.

In the implementation phase, a web application is presented with a series of microservices on which the capabilities of mTLS and communication via http will be implemented.

Índice

Lista de ilustraciones

Lista de tablas

1. Introducción

Todo tipo de plataforma o arquitectura tiene un **riesgo inherente** frente a vulnerabilidades. En este trabajo se mostrarán las buenas prácticas que pueden ser utilizadas en **arquitecturas basadas en microservicios o malladas** y cómo estas medidas de seguridad y buenas prácticas **mitigan** posibles riesgos y vulnerabilidades.

Tomando como base este tipo de arquitecturas y bajo la premisa de la necesidad de utilizar **protocolos de comunicación seguros** en estas arquitecturas, el presente TFM tiene por objeto **implantar una solución** que utiliza protocolos seguros en la comunicación entre servicios en arquitecturas malladas (Service Mesh) utilizando **Istio**.

Para ello, el TFM hace una revisión de los diferentes conceptos necesarios para **entender la solución desde el punto de vista conceptual**, cuáles son las **soluciones que Istio propone para implementar este tipo de arquitecturas** y por último utilizamos una **solución práctica de estos conceptos sobre un aplicativo** web con microservicios, sobre la cual habremos habilitado **comunicaciones basadas en mTLS** y habilitaremos el **protocolo https** entre los servicios implementados.

1.1. Contexto y justificación del trabajo.

Partiendo del enunciado de este estudio, "**uso de certificados digitales para la autenticación y autorización de APIs**", este TFM pretende dar una visión completa de cómo implementar protocolos seguros de comunicaciones en arquitecturas de microservicios basadas en contenedores. Para ello, **utilizaremos Istio como herramienta que ofrece diferentes capacidades**, entre ellas la posibilidad de implementar soluciones basadas en protocolos de comunicaciones seguros.

Uno de los problemas a los que se enfrenta una compañía, no es sólo conocer los activos que tiene que proteger, sino además cómo estos se comportan, quién puede acceder a ellos y dónde tienen que acceder. Algo así como definir previamente las **aplicaciones que pueden ser accedidas y quién y qué puede acceder a ellas**.

Tradicionalmente este análisis y definición de políticas de acceso se ha venido haciendo a través del tráfico que se permitía realizar a través del perímetro de la compañía con el exterior y dentro de la compañía. Esta labor se ha venido realizando con elementos de electrónica de red, que efectivamente seguirán haciendo su trabajo de análisis de tráfico y definición de políticas sobre dicho tráfico de red.

Sin embargo, en este TFM se aborda un acercamiento a la problemática de la **gestión y protección de accesos desde el punto de vista de la definición y diseño de las arquitecturas implementadas en un activo cualquiera**. Para ello analizaremos las arquitecturas basadas en microservicios (introducimos un grado de detalle mayor a las APIs y a las arquitecturas intentando abarcar un análisis más general que pueda abarcar este tipo de arquitecturas) y las soluciones que ofrecen las arquitecturas de referencia en el Cloud.

Este punto inicial del TFM nos debe permitir analizar cómo los certificados digitales definen la autenticación y autorización del cliente o la entidad que requiere el acceso al servicio publicado (API, contenedor…).

1.2. Objetivos del trabajo.

Las capacidades de diseño y desarrollo sobre arquitecturas malladas (Service Mesh) o arquitecturas basadas en microservicios, nos ofrecen posibilidades desde el punto de vista de ciberseguridad que dotan a los activos y plataformas de una capa de seguridad que antes no podía desarrollarse en otro tipo de arquitecturas.

En este proyecto, presentamos los **riesgos y vulnerabilidades** a los que se puede enfrentar cualquier aplicación y desde el punto de vista de la definición de medidas de seguridad, introducimos **Istio y sus mejores prácticas** para la mitigación de esos riesgos.

Por tanto, los objetivos del TFM se pueden definir en los siguientes puntos:

- Como en todo proyecto de ciberseguridad, la aplicación de estas capacidades de seguridad, deberá mitigar una serie de **riesgos y vulnerabilidades**.
- Introducción breve a **conceptos clave**.
    - Arquitecturas de microservicios.
    - Definición de contenedores, APIs, servicios web…
    - Gestión de identidades.
    - Arquitecturas Service Mesh.
- **Analizar las capacidades que ofrece Istio** en arquitecturas basadas en microservicios para dotarlas de estas capacidades de seguridad.
- **Desarrollo práctico de una solución basada en microservicios** en la que se implementen **protocolos seguros de comunicaciones (mTLS)** con Istio sobre una aplicación.

El objetivo principal del TFM es presentar Istio de forma práctica sobre un aplicativo y cómo sus capacidades de seguridad pueden mitigar riesgos y vulnerabilidad concretos, aplicando una capa de seguridad más sobre las que habitualmente se pueden realizar en cualquier sistema a implantar.

1.3. Enfoque y método seguido.

El objetivo último de un sistema de gestión de seguridad de la información es **proteger los activos de información y los procesos de negocio** que tienen base sobre estos activos. Es decir, el sistema de gestión de seguridad de la información debe estar **alineado con los objetivos de negocio**.

Para ello, se debe poder **valorar la criticidad de los activos** en base a las dimensiones que se estimen oportunas según la metodología que tomemos como referencia. Éste será el primer paso previo al análisis de riesgos que pueden estar amenazando nuestros procesos de negocio y sus activos de información.

En base al cálculo del riesgo, se deberán definir cuáles son las **medidas técnicas de ciberseguridad** que podrán ser adoptadas con el objetivo de mitigar o reducir el **riesgo intrínseco** calculado previamente. Al riesgo obtenido después de la aplicación de estas medidas técnicas, lo llamaremos **riesgo residual**. Además, estas medidas técnicas, nacerán de la definición de unas **políticas y normativas** que podrán tener su origen en diferentes estándares y/o normativas de aplicación legal según el sector en el que opere el negocio.

Por último, se deberá definir un **plan de acción, que implemente dichas medidas** y conduzca a la empresa a **reducir el nivel de riesgo y a mejorar su madurez**, mediante procesos de **monitorización continua y gestión del cambio**.

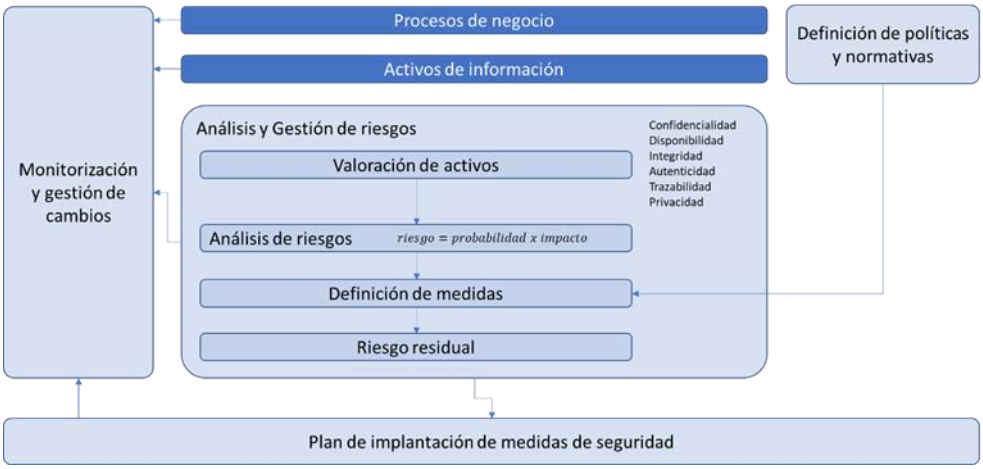


Ilustración 1: Metodología de gestión del riesgo.

Si bien no aplicaremos una metodología de cálculo del riesgo, no es objeto del TFM, sí que mencionamos **cuál es la metodología de trabajo que nos ha llevado a las conclusiones del mismo**.

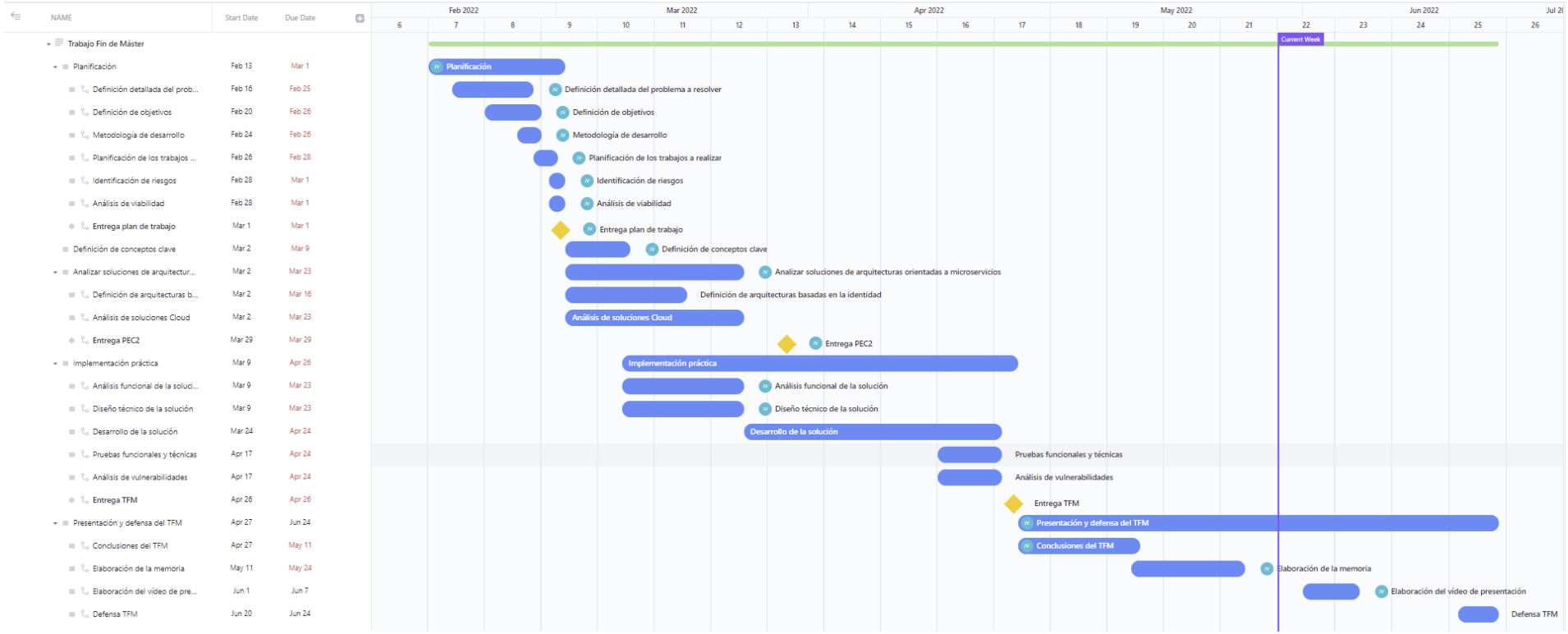
De este modo, se presenta un análisis de riesgos de forma que se identifican, los posibles **riesgos y vulnerabilidades**. Y sobre estos riesgos y vulnerabilidades se presentan las **mejores prácticas de desarrollo y medidas de seguridad con la plataforma Istio**.

De esta forma, siguiendo los objetivos que se han definido en el apartado anterior, teniendo en cuenta que se quieren cubrir tres ámbitos en base a la arquitectura de microservicios y cómo proteger los accesos a éstas, siguiendo con la metodología de análisis de riesgos, **se analizan las soluciones y medidas técnicas de ciberseguridad** basándonos en los siguientes puntos:

- **Análisis de las arquitecturas basadas en microservicios** y cómo Istio puede dotar de capacidades de seguridad a estas arquitecturas.
- **Integración de la funcionalidad de comunicaciones** basadas en identificación, autenticación y autorización basadas en Istio.
- **Implementación práctica de la solución**. En realidad, ésta será la parte que más peso en esfuerzo llevará en la ejecución del proyecto.

1.4. Planificación del trabajo

Habida cuenta de los objetivos marcados y las diferentes tareas descritas en el apartado de metodología, este sería el listado y planificación de tareas a realizar en la ejecución del TFM.



1.5. Estado del arte

Istio, tal y como veremos en el apartado dedicado a la herramienta, nos permite implementar todas sus capacidades en cualquier herramienta que permita su instalación basada en Kubernetes.

Revisando las diferentes opciones que pueden darse en las arquitecturas Cloud existentes en el mercado, se puede observar cómo hay opciones tecnológicas similares quizás en su definición, pero parece a priori necesario realizar un análisis de las posibles soluciones que se ofrecen en el mercado.

No es objetivo de este TFM analizar en profundidad cómo se puede implementar Istio y sus capacidades en cada una de las soluciones de mercado en Cloud, si bien sí que se implementará Istio en una de ellas para demostrar que la solución es viable en arquitecturas Cloud (ver apartado 6).

En el desarrollo del TFM, se ha podido implementar una solución de Istio sobre una plataforma Minikube en local sobre una máquina virtual en Ubuntu (ver apartado 5.2). Siguiendo los mismos pasos, se ha implementado la misma solución sobre GCP (Google Cloud Platform – ver apartado 6).

En este apartado, hacemos un breve resumen de las diferentes soluciones que existen en el mercado para la gestión de arquitecturas de microservicios. Hemos optado por trabajar con Istio, al ser ésta una plataforma de código abierto y que permite optimizar el uso y los recursos de una manera más didáctica.

A continuación, se reflejan las diferentes opciones, dando un pequeño resumen de cada una de ellas.

- Istio
- Google API Gateway
- Azure API Management
- Amazon API Gateway

En realidad, los tres últimos, responden a la gestión de APIs en arquitecturas implementadas en los propios cloud. El acercamiento en Istio a la solución pasa por un desarrollo y configuración sobre cualquier diseño de microservicios.

1.5.1. Istio

Desarrollaremos específicamente un apartado dedicado a **Istio y todas las capacidades y funciones que se pueden implementar en el ámbito de seguridad** (ver apartado 4), detallando aquellas que hemos marcado como objetivo en este TFM.

Istio es una solución de código abierto que aporta soluciones de **gestión de tráfico** entre servicios en una **arquitectura basada en kubernetes**. Esta solución nos permitirá implementar **gestión de tráfico basado en mTLS de forma que podremos controlar dicho tráfico con protocolos seguros**.

Además, **Istio es implementable en soluciones Cloud**. No es objeto del TFM entrar en detalle en cómo se debería desplegar estas soluciones en los diferentes Cloud, pero sí que podemos hacer un breve resumen en este apartado sobre el estado del arte de éstas y en los resultados definitivos del TFM hacer entrega de la solución en uno de ellos, concretamente GCP (Google Cloud Platform).

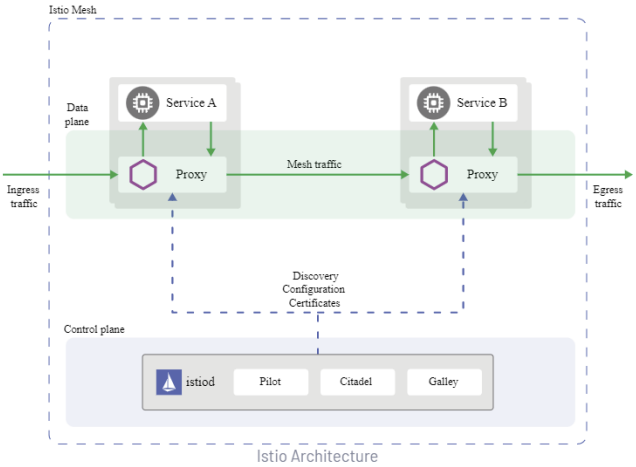


Ilustración 2: Arquitectura Istio

1.5.2. Google API Gateway

API Gateway es el **módulo que permite la conexión segura** con los desarrollos que se realicen en el ámbito Cloud de Google (Google Cloud Platform Services)[1] "*API Gateway enables you to provide secure access to your services through a well-defined REST API that is consistent across all of your services, regardless of service implementation*"

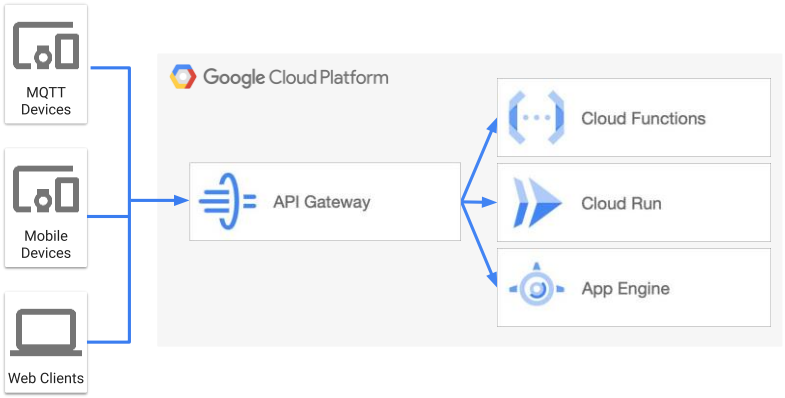


Ilustración 3: API Gateway

1.5.3. Azure API Management

Del mismo modo que en las otras herramientas objeto de estudio, **Azure API Management permite crear un enlace con las diferentes API publicadas**, de forma que se puedan integrar diferentes API en diferentes Cloud[2].

[1] *About API Gateway.* [en línea] [fecha de consulta: 1 de marzo de 2022]. Disponible en: https://cloud.google.com/api-gateway/docs/about-api-gateway

[2] *Acerca de API Management.* [en línea] [fecha de consulta: 1 de marzo de 2022]. Disponible en: https://docs.microsoft.com/es-es/azure/api-management/api-management-key-concepts

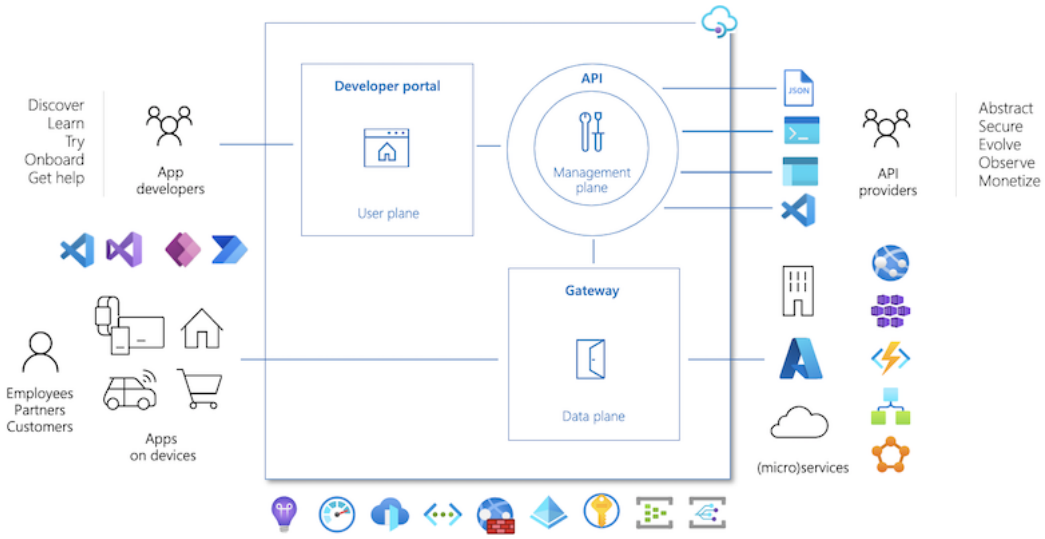


Ilustración 4: Azure API Management

1.5.4. Amazon API Gateway

Se trata de una misma solución, en este caso para entornos AWS, donde Amazon API Gateway "*es un servicio de AWS para la creación, la publicación, el mantenimiento, el monitoreo y la protección de las API REST, HTTP y de WebSocket a cualquier escala. Los desarrolladores de API pueden crear API que obtengan acceso a AWS o a otros servicios web, así como los datos almacenados en la nube de AWS*"[3]

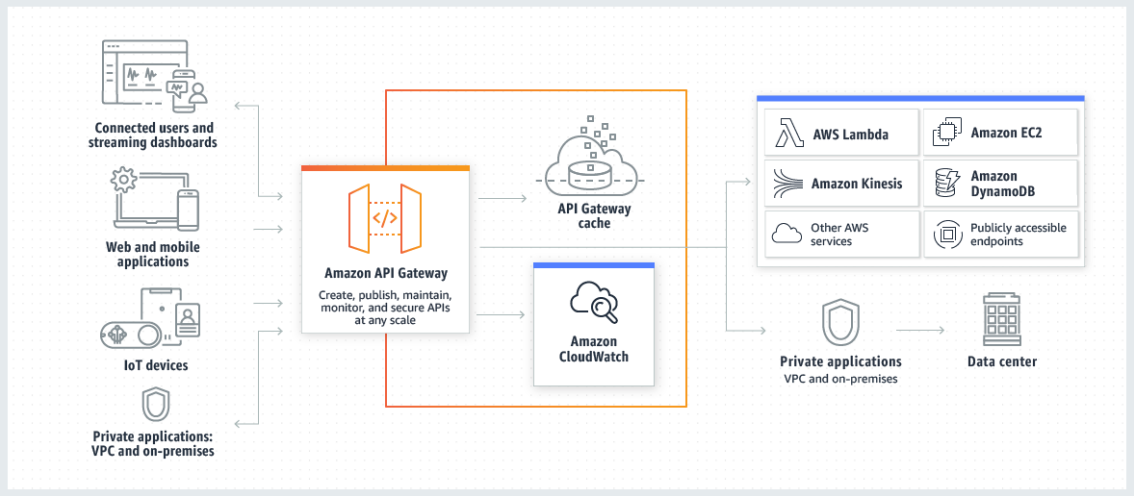


Ilustración 5: Amazon API Gateway

1.6. Breve sumario de productos obtenidos

En este TFM se tiene un **análisis completo y detallado** de cómo implementar **medidas técnicas de ciberseguridad** en una solución basada en microservicios y cuáles son las **capacidades que ofrece Istio para su implementación**. Resulta una completa **guía de implementación de Istio** basada en el análisis de riesgos y la **implementación de soluciones de seguridad que mitiguen dichos riesgos**.

[3] *¿Qué es Amazon API Gateway?*. [en línea] [fecha de consulta: 1 de marzo de 2022]. Disponible en: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/welcome.html

No existe una entrega de software de desarrollo, ya que la implementación de Istio está basada en productos de código abierto, accesibles vía código abierto.

1.7. Breve descripción de los otros capítulos de la memoria

La memoria de este TFM se estructura en los siguientes apartados:

- **Introducción**: detalle de los objetivos del TFM y del trabajo de análisis realizado.
- **Análisis de riesgos**: basamos todo el análisis y conclusiones en un análisis de riesgos y vulnerabilidades, poniendo en valor las medidas técnicas de seguridad que se implementarán como mitigación de estas vulnerabilidades.
- **Conceptos base**: breve resumen sobre conceptos base.
- **Istio**: definición de la plataforma y de sus capacidades.
- **Aplicación BookInfo**: piedra angular del TFM. Se hace una introducción funcional del aplicativo sobre el que se implementarán las medidas. A continuación, se describe la guía de implementación de las capacidades de Istio sobre el aplicativo.
- **Implantación de Istio en GCP**: a modo de demostración de las capacidad de Istio a ser implementada en cualquier Cloud, se presenta la implementación de Istio en Google Cloud Platform.

2. Análisis de riesgos

Uno de los objetivos de este TFM es **dotar de medidas de seguridad** a una plataforma frente a **riesgos y vulnerabilidades**. El objetivo de todo proyecto de ciberseguridad es medir el riesgo de un activo, entender sus vulnerabilidades y mitigar dicho riesgo mediante la aplicación de capacidades de seguridad sobre la plataforma.

En este apartado **identificaremos los riesgos y las medidas técnicas** que deberían aplicarse para mitigar dichos riesgos. Haremos una breve introducción al concepto de riesgo, definiremos las vulnerabilidades que típicamente se pueden encontrar en este tipo de plataformas y definiremos las medidas técnicas que se pueden adoptar para mitigar dichos riesgos.

Podría entenderse el concepto de riesgo sobre un activo, siguiendo la ilustración de abajo. De esta forma, podemos entender que las **amenazas aprovechan vulnerabilidades que explotan para generar un impacto** (normalmente negativo) sobre los activos. Tanto las **amenazas, como las vulnerabilidades y el impacto que éstas ocasionen, incrementan el riesgo**. En la gráfica, aumentan el círculo en rojo el riesgo. Mientras que **las medidas técnicas compensatorias, disminuyen en riesgo reduciendo también la probabilidad de las amenazas**.

Ilustración 6: Visión conceptual de la gestión del riesgo.

Para la definición de la metodología de análisis del riesgo, sin entrar en el cálculo, seguiremos la siguiente ilustración.

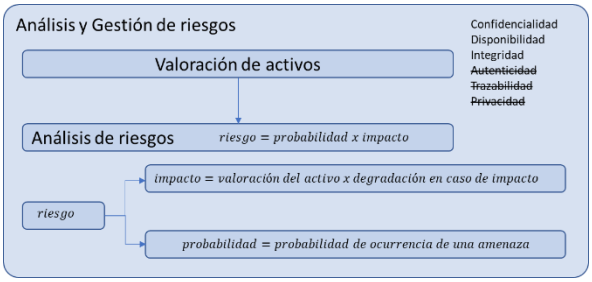


Ilustración 7: Metodología propia

Nos interesa la etapa de definición de medidas compensatorias en base a la definición de las vulnerabilidades. En cualquier caso, conceptualmente, se debe tener claro que la aplicación de un análisis de riesgos es pieza clave para entender qué medidas compensatorias deben implementarse. En la Ilustración 6, estas medidas compensatorias están representadas por las medidas técnicas que disminuyen el riesgo y el impacto de posibles vulnerabilidades.

En la definición de los riesgos, hemos **dividido estos en dos tipos de riesgos**. Los riesgos **inherentes a la ejecución del proyecto** y los **riesgos propios de la implementación de cualquier proyecto de aplicación** (hemos denominado vulnerabilidades) que con la aplicación de Istio procuramos mitigar.

2.1. Riesgos inherentes a la ejecución del proyecto

En sucesivas fases de entregas del proyecto se ha ido profundizando en el análisis de riesgos, si bien desde el inicio del mismo ya se intuyen riesgos ligados a la tecnología y a la consecución de objetivos. Tal y como se ha explicado en los apartados anteriores, se está queriendo conjugar varios elementos tecnológicos:

- Arquitecturas orientadas a microservicios.
- Publicación de API sobre estas arquitecturas.
- Soluciones de seguridad sobre la publicación de estas APIs.
- Acceso seguro mediante certificados sobre estas APIs.

Por tanto, a priori se podrían definir los siguientes riesgos ante los que nos encontramos, como riesgos inherentes de la tecnología y el contexto propios del proyecto y los podemos enumerar en la siguiente tabla:

Id.	Definición del riesgo	Mitigación
R0	Aplicación práctica de la tecnología	Análisis previo de las soluciones
R1	Integración de diferentes soluciones	Definir un caso práctico con una única solución que permita llegar a las conclusiones objetivo del TFM
R3	Integración con un IdP	Veremos en el desarrollo del TFM la gestión de certificados autogenerados con Istio.
R4	Implementación práctica	Se ha optado por una implementación en local, tal y como se detalla en el apartado 5.2.

Tabla 1: Análisis de riesgos inherentes.

2.2. Análisis de vulnerabilidades

Avanzamos en el análisis de riesgos y damos una visión de las amenazas y riesgos que se pretenden cubrir con la implantación de Istio en cualquier solución basada en Kubernetes.

Id.	Definición del riesgo	Mitigación
R5	Movimiento lateral	Aplicación de mTLS
R6	Man in the middle	Aplicación de mTLS y apertura de tráfico http
R7	Extracción de información	Capacidad de observación y mitigación de amenazas
R8	Suplantación de indentidad	Implantación de certificados.

Tabla 2: Análisis de vulnerabilidades

La aplicación de medidas técnicas basadas en la implementación de protocolos seguros, deben mitigar los riesgos identificados.

2.3. Beneficios de las arquitecturas Service Mesh

Tal y como veremos a lo largo del documento, las arquitecturas Service Mesh ofrecen una serie de **funciones y capacidades que aportan beneficios** claros sobre otras arquitecturas, todo ello basado en la **segmentación de funciones** a la hora de implementar este tipo de arquitecturas.

Si bien se entra más en detalle en el apartado 4, cuando hablamos de segmentación de funciones de una arquitectura, estamos hablando de diferentes **capacidades que dicha arquitectura implementa**, diferenciándolas de **la implementación puramente funcional del aplicativo** que se soporta sobre dicha arquitectura.

Adelantándonos al detalle que se define más adelante en el documento, una implantación funcional del aplicativo, por ejemplo, puede tratarse de información sobre un libro que se muestra en pantalla. La implementación de acceso al servicio que nos devuelve esta información debe estar separada, tanto en código como en implementación de la parte funcional.

Revisando la Ilustración 8 (se tiene más detalle en el apartado 4), se observa cómo están separadas las funciones de Istio Mesh, sobre las funciones propias de los servicios implementados (representado en la ilustración como Service A y Service B).

Este entramado arquitectónico cubre funciones como las detalladas a continuación:

- Seguridad
- Gestión del tráfico
- Capacidad de observación, protección contra amenazas.

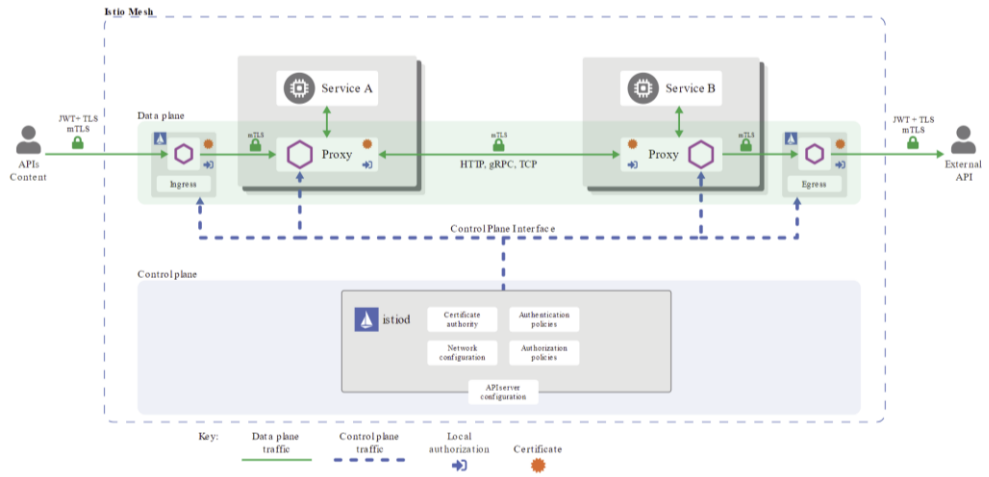


Ilustración 8: Funciones de una arquitectura Service Mesh

Son estas funciones, las que, desarrolladas e implementadas sobre arquitecturas basadas en microservicios, **mitigarán los riesgos y dotarán a la aplicación y a la plataforma implantada de una capa adicional de seguridad**. Y esta aplicación de seguridad además está **separada funcional y técnicamente** de la implementación propia de funcionalidad del aplicativo, dando respuesta a la separación y segmentación de funciones.

3. Breve introducción a conceptos clave

Durante el desarrollo de este TFM se trabajará con diferentes estándares. El TFM será fiel al título del trabajo del área "uso de **certificados digitales** para la **autenticación y autorización** de **APIs**". En este apartado empezaremos por definir los conceptos básicos que dirigen el trabajo.

3.1. ¿Qué es un API?

Para saber qué es un API, nos vamos a basar en lo que dice el OAI (The Open API Initiative) sobre lo que es un API. "*An **Application Programming Interface (API)** defines the allowed interactions **between two pieces of software**, just like a User Interface defines the ways in which a user can interact with a program.*"[4]

Se trata de una definición muy básica, si bien ya nos está dando directrices sobre lo que se basará el TFM. Un API es un **interfaz (Interface)**, por tanto, **contiene una definición** de cómo se puede **llamar a dicho código** "envuelto" en el API para ser ejecutado. Decimos que dicho código que contiene el API **se ejecuta por una aplicación de un tercero (Application)** que **programa (Programming)** dicha llamada al API.

Por tanto, un API hace disponible un trozo de código en tiempo de ejecución para que un tercero pueda ejecutarlo, "*…defines the allowed interactions **between two pieces of software**…*"

Lógicamente esta definición se "alarga" bastante más, ya que un API tiene más características necesarias para entender qué tipos de arquitecturas se pueden definir a través de un buen diseño.

3.2. Gestión de identidades y accesos

Nos basaremos en una solución que nos permita definir la identidad (quién o qué, **autenticación**) tiene acceso (está autorizado, **autorización**) a un elemento software (en nuestro caso a una API). Esta gestión de autenticación y autorización, se realiza con un gestor de identidades y accesos, conocido como **IAM (Identity Access Management)**. La definición podría ser la siguiente… "*Identity and access management (IAM) is a **framework of business processes, policies and technologies** that facilitates the management of electronic or **digital identities**.*"[5]

Se trata por tanto de una plataforma con una serie de funciones que facilita la gestión de **la identidad y la definición de políticas** para **gestionar los accesos de esas identidades**.

Un API contiene el código a ejecutar y la gestión de IAM definirá quién puede hacer la llamada a ese API para ejecutarla. **En IAM se define quién puede acceder a qué**.

3.3. Certificados digitales

Esta sección supone la piedra angular de este trabajo ya que el "certificado digital" será el elemento que dotará de autenticación al proceso. Explicaremos varios conceptos basándonos en el protocolo TLS y en su definición.

---

[4] *What is an API?.* [en línea] [fecha de consulta: 28 de marzo de 2022]. Disponible en: https://github.com/OAI/Documentation/blob/main/introduction.md

[5] *What is identity and access management? Guide to IAM* [en línea] [fecha de consulta: 28 de marzo de 2022]. Disponible en: https://www.techtarget.com/searchsecurity/definition/identity-access-management-IAM-system

3.3.1. TLS 1.3 – Tansport Layer Protocol versión 1.3[6]

Tomamos como base la definición del protocolo TLS para entender cómo tiene que trabajar la comunicación entre dos instancias para que ésta sea segura. "*The primary goal of TLS is to **provide a secure channel between two communicating peers***"

El protocolo TLS define tres características básicas a cubrir en la comunicación entre dos elementos:

- **Autenticación**: el protocolo define la necesidad de que la parte servidora esté siempre autenticada y la parte cliente lo esté opcionalmente. En cualquier caso, esta capacidad de autenticación en nuestro caso está cubierta por la capacidad de **certificación**, lo que se describe en el protocolo como "*…asymmetric cryptography … or a symmetric pre-shared key*"
- **Confidencialidad**: la información que se comparte entre las dos partes sólo es visible por estas dos partes. "*Data sent over the channel after establishment is only visible to the endpoints*".
- **Integridad**: se asegura la integridad del envío y recepción de la información, de modo que ésta no se altera. "*Data sent over the channel after establishment cannot be modified by attackers without detection*".

Estos tres principios son los que deberán asegurarse en la implementación del TFM, en este caso aplicados a las API. En nuestro caso nos centraremos en la primera capacidad de autenticación.

Esta implementación se basa en dos protocolos:

- Handshake protocol: "***authenticates the communicating parties**, negotiates cryptographic modes and parameters, and establishes shared keying material*". Cuando se produce la comunicación entre las partes, el primer intercambio de información se produce para autenticar dicha comunicación (ver Ilustración 9: Protocolos TLS)
- Record protocol: "*…that **uses the parameters established by the handshake protocol** to protect traffic between the communicating peers*"

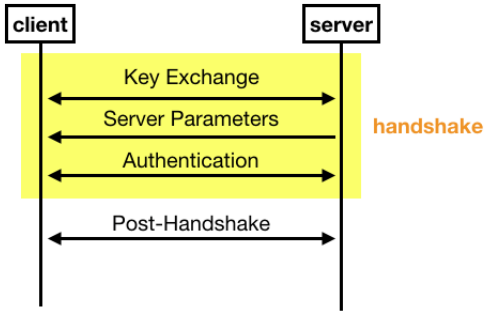


Ilustración 9: Protocolos TLS

3.3.2. CA – Certification Authority

Los certificados están emitidos por una autoridad certificadora acreditada que es reconocida como Autoridad de Certificación (AC o CA en inglés Certification Authority). TLS utiliza certificados X.590v3 (según RFC 5280[7]).

[6] *Especificación TLS 1.3.* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://www.rfc-editor.org/rfc/rfc8446.txt

[7] *Especificación X.590v3* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://datatracker.ietf.org/doc/rfc5280/

3.4. Arquitecturas de microservicios

Cuando se diseña una aplicación se puede optar por diferentes tipos de arquitecturas. En cualquier caso, una arquitectura basada en microservicios busca que el desarrollo de este software sea independiente en su ejecución y el diseño de los servicios funcionales atienda a las necesidades del proceso de negocio de forma segmentada. Dicho de otra forma, los requerimientos funcionales se atienden en servicios identificados de forma muy concreta y el diseño de la arquitectura atienda a estos servicios (microservicios) para que su ejecución sea independiente.

Por ejemplo. Un sistema en el que se quiere conocer el nombre y apellidos de una persona y dirección postal y de correo electrónico a través de su DNI. Habitualmente este tipo de funciones se podrían llegar a desarrollar de forma monolítica y obtener el resultado en una única llamada.

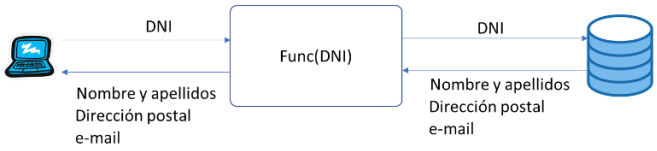


Ilustración 10: Función monolítica

En una arquitectura con microservicios, estas funciones se segmentan y nacen de esta forma los microservicios, orientados en este caso a dar respuesta a la necesidad concreta funcional. En este caso, podemos dividir (segmentar) la primera función en tres microservicios, que devolverán la información segmentada, tal y como se muestra en el siguiente diagrama.

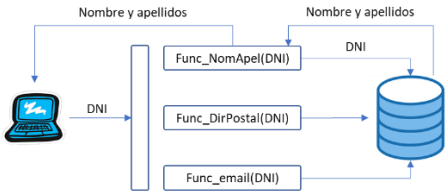


Ilustración 11: Arquitecturas de microservicios

3.4.1. Service mesh

Hasta el momento estamos hablando de una comunicación en una arquitectura cliente-servidor. En realidad, lo que este TFM quiere aportar es qué ocurre en arquitecturas dónde la comunicación se puede gestionar entre diferentes elementos y donde éstos pueden ser llamados tanto como servidores o que pueden ser también clientes que realizan una llamada a un servidor.

Service mesh o malla de servicios, dota de una capacidad de interacción entre los diferentes servicios, de modo que se puede diseñar una arquitectura basada en microservicios, donde estos se apoyan unos en otros para retornar la información necesaria al usuario de la aplicación.

Siguiendo con el ejemplo anterior, podríamos pensar que tuviéramos un servicio que devuelve el password de una persona a partir de su email. Por tanto, podríamos tener una arquitectura tal y como sigue en la siguiente ilustración.

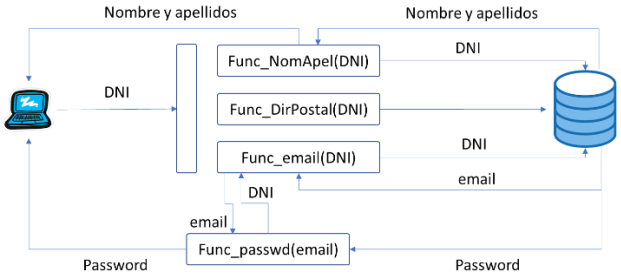


Ilustración 12: Service Mesh

Una arquitectura mallada con servicios Service Mesh ofrece soluciones a tres tareas básicas:

- Observar, recoger toda la telemetría de comportamiento de la plataforma.
- Gestión del tráfico, configuración del tráfico de datos.
- Seguridad, configuración de medidas de seguridad, apoyadas en la telemetría y la gestión de tráfico.

Habitualmente, antes de existir este tipo de arquitecturas, estas tareas recaían en los propios elementos de la plataforma, de forma que unas veces este tipo de programación residía en el propio desarrollo de las aplicaciones (sea del tipo que fueran, monolíticas, cliente-servidor, orientadas a servicios, microservicios…) y otras recaía en elementos de red destinados a proteger o gestionar el tráfico. En cualquier caso, en el ámbito de ciberseguridad, una arquitectura mallada basada en servicios viene a complementar como una capa más de seguridad (seguridad por capas y/o seguridad en profundidad) de forma que, si bien existen otras capacidades habilitadas por este tipo de plataformas, nos centraremos en la seguridad y la gestión del tráfico.

Conceptualmente, y a modo de resumen de cómo funciona una arquitectura Service Mesh, estamos separando la lógica de negocio a implementar en el software y que debe cubrir las necesidades funcionales del aplicativo, de la lógica de control y seguridad que también debe ser implementadas en la arquitectura. La palabra clave es segmentación, concretamente segmentación de funciones, de forma que en la misma arquitectura tenemos implementadas ambas, función de negocio y función de seguridad, con dos capas bien diferenciadas, lo cual nos aporta beneficios[8] sustanciales.

Esto se consigue implementando una lógica de seguridad sobre la capa de lógica de negocio de los servicios. El siguiente diagrama lo explica conceptualmente.

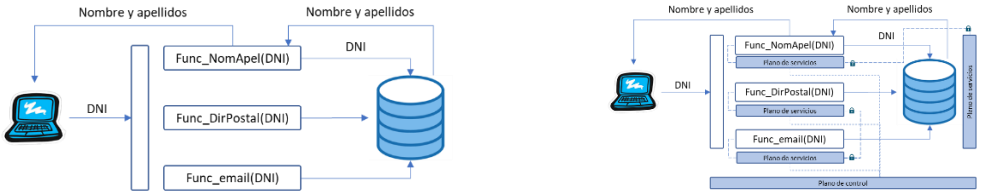


Ilustración 13: Microservicios y microservicios con Service Mesh

En la parte izquierda tenemos una arquitectura de microservicios, dónde hablamos exclusivamente de los servicios y sus funciones de negocio. En la parte de derecha, hemos segmentado e implementado la función de seguridad (representada por las cajas azules), donde se puede observar la capacidad de control sobre los servicios (tráfico, operación y seguridad) y los diferentes flujos de información que se generan en la arquitectura.

[8] Ver apartado 2.3 sobre los beneficios de este tipo de arquitecturas.

4. Istio

Istio[9] es una plataforma OpenSource que ofrece capacidades de desarrollo basadas en Kubernetes y que permite gestionar arquitecturas diseñadas en microservicios adoptando arquitecturas Service Mesh.

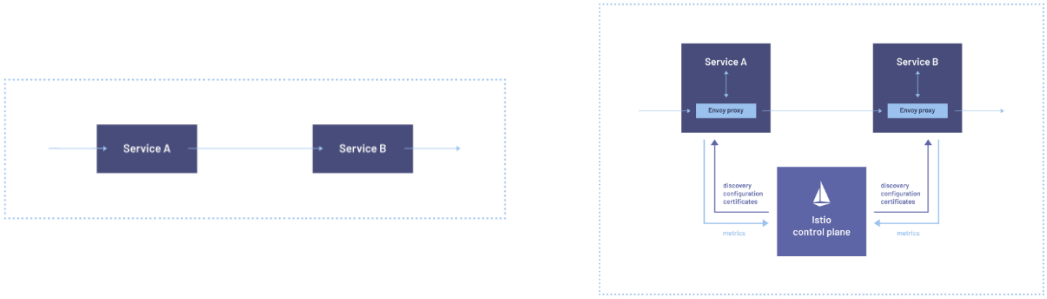


Ilustración 14: Qué es Istio?

Tal y como hemos visto en el apartado introductorio anterior a las arquitecturas Service Mesh, Istio desarrolla dos planos que implementan todas sus capacidades. A partir de este punto utilizamos la terminología propia de Istio.

- **Data plane** (plano de servicios), que desarrolla toda la funcionalidad propia de los servicios y los datos. En la Ilustración 14, en la parte derecha, se trata de la parte de que se adhiere (sidecar) a los servicios y que está representada con las cajas de Envoy Proxy. Estas cajas recogen la información de configuración que llega del plano de control (Control Plane) y gobiernan el comportamiento de los servicios.
- **Control plane** (plano de control), responsable de la definición y configuración del comportamiento de la plataforma, envía la información necesaria de control al data plane y recoge toda la información de telemetría de comportamiento de la plataforma.

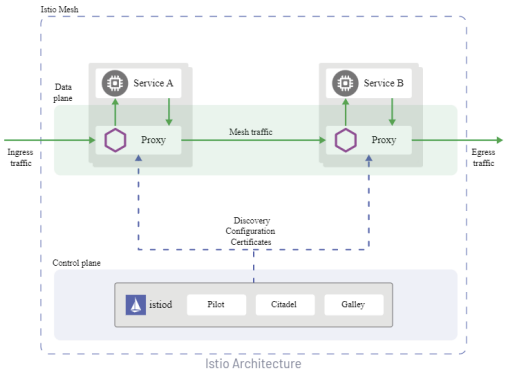


Ilustración 15: Arquitectura Istio

En la Ilustración 15 se representa un mayor detalle de la arquitectura de Istio, donde se pueden ver los dos principales componentes de Istio, representados por la tecnología que los implementa. Envoy para el Data Plane e Istiod para el Control Planer.

Tal y como se ha comentado en el apartado anterior, esta implementación ofrece la posibilidad de ejecutar las siguientes tareas:

- **Observability**. El plano de control es capaz de recibir toda la telemetría generada en el tráfico que se produce entre los servicios. Esta telemetría permite tomar

[9] *What is Istio* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://istio.io/latest/about/service-mesh/

decisiones de eficiencia en el mantenimiento de la plataforma, pero también permite definir casos de uso que permitan proteger la misma.
- **Traffic management**. Gestión del tráfico a través de reglas configuradas en el plano de control, de modo que se puede administrar el tráfico a través de las API de los servicios.
- **Security**. Istio ofrece funciones de seguridad que dotan capacidades de gestión de accesos y definición de políticas basadas en modelos de ZTNA. Estas son las capacidades que ofrece Istio:
    - Identidad
    - Definición de políticas
    - Mutual TLS
    - Autenticación
    - Autorización
    - Auditoria

En el objeto de este TFM, nos centraremos en las capacidades de Security que ofrece Istio, ya que diseñaremos una solución basada en microservicios, donde implementaremos las capacidades de mTLS (mutual TLS) entre los servicios. Para ello deberemos entender algunos conceptos propios de la solución de Istio y que nos servirán de introducción al objetivo del TFM.

4.1. Arquitectura global de seguridad

Istio provee diferentes compenentes dentro de Istiod orientados a las funcionalidades de seguridad. De esta forma y tal y como se presenta en la siguiente ilustración, Istiod tiene las siguientes capacidades, tal y como se muestran en Ilustración 19:

- Identidad y servicio de CA (Certificate Authority)
- Configuración de políticas de seguridad:
    - Políticas de autenticación.
    - Políticas de autorización.

4.1.1. Identidad y servicio de CA[10]

Introducidos los conceptos básicos en los capítulos anteriores y entendiendo el concepto de diseño de las arquitecturas malladas Service Mesh, entendemos que en toda comunicación entre dos elementos software, éstos deberán identificarse y para ello intercambiarán sus credenciales. La identidad y la gestión de credenciales vienen a definir y resolver este problema.

En este caso la arquitectura desarrollada permitirá la certificación de los servicios (API – microservicios) a través de una entidad certificadora autorizada. "*Istio securely provisions strong identities to every workload with X.509 certificates. Istio agents, running alongside each Envoy proxy, work together with istiod to automate key and certificate rotation at scale.*"

[10] *Identity and certificate management* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://istio.io/latest/docs/concepts/security/#pki

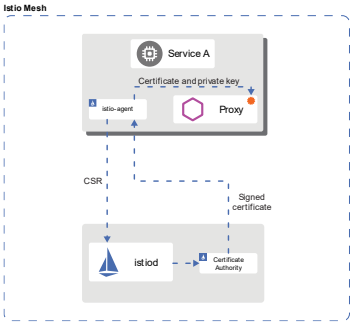


Ilustración 16: Autoridad certificadora

En la Ilustración 16 se observa cómo los servicios pueden obtener un certificado a través de la funcionalidad de Certificate Authority. Básicamente, los servicios de la arquitectura se basan en esta funcionalidad para poder identificarse a través de un certificado interno autogenerado mediante el módulo Certificate Authority de Istiod.

4.1.2. Autenticación

En Istio se utilizan dos tipos de autenticación. En este TFM nos basaremos en la autenticación de servicios. Los dos tipos de autenticación que utiliza Istio son:

- **Peer authentication**, basada en el protocolo mTLS (mutual TLS, ver apartado 4.2) y que ofrece la autenticación entre servicios.
- **Request authentication**, que se utiliza para la autenticación de usuarios.

Partimos del hecho de que en la autenticación estamos verificando la identidad, en este caso del servicio o del usuario.

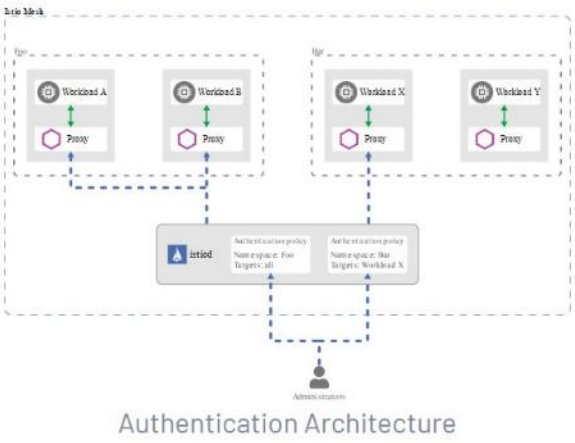


Ilustración 17: Arquitectura de autenticación

4.1.3. Autorización

Tal y como ya estamos viendo en la definición de la arquitectura, Istio provee también la capacidad de autorización de tráfico entre los servicios de una arquitectura Service Mesh, basándose en la configuración que llega a los servicios proxy de cada uno de los servicios.

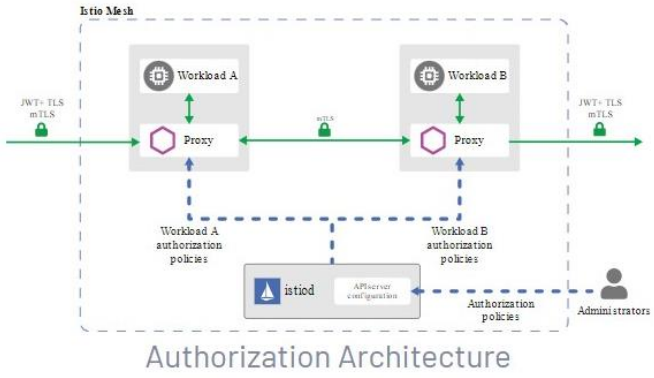


Ilustración 18: Arquitectura de autorización.

A modo de ejemplo y tal y como se muestra en la Ilustración 18 de forma conceptual, nuevamente es el plano de control desde donde se configuran los controles de acceso que autorizan las conexiones entre los servicios que conforman el servicio mallado.

4.2. Mutual TLS

La capacidad que ofrece Istio para proteger las comunicaciones se basa en el protocolo mTLS. Istio, básicamente, tuneliza las comunicaciones entre los diferentes servicios basándose en las capacidades de autenticación, autorización y cifrado de información según protocolo mTLS ("*Secure service-to-service communication in a cluster with TLS encryption, strong identity-based authentication and authorization*").

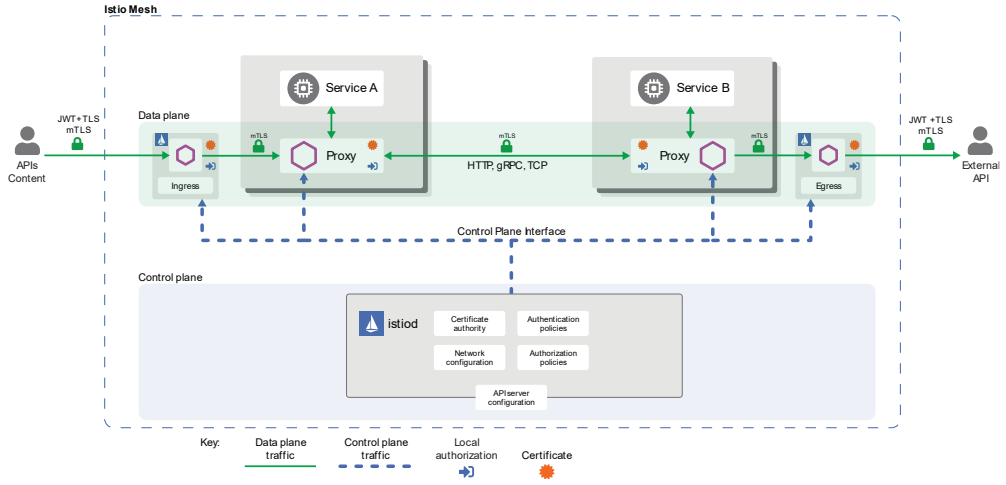


Ilustración 19: Gestión de la seguridad en Istio

Tal y como hemos visto en las arquitecturas (y conceptos) de autenticación y autorización, las configuraciones se guardan y gestionan desde el plano de control (control plane) de istiod. Estas configuraciones gobiernan las entradas y salidas de los servicios en el plano de datos (data plane).

5. Aplicación BookInfo

Detallamos funcional y técnicamente qué hace la aplicación basada en kubernetes sobre la que implantaremos las capacidades de seguridad. Además, tal y como hemos definido en apartados previos, instalaremos la solución en una de las herramientas Cloud que hemos analizado en el estado del arte, concretamente GCP (Google Cloud Platform), además de la instalación que haremos en local sobre una máquina virtual Debian sobre una solución de miniKube.

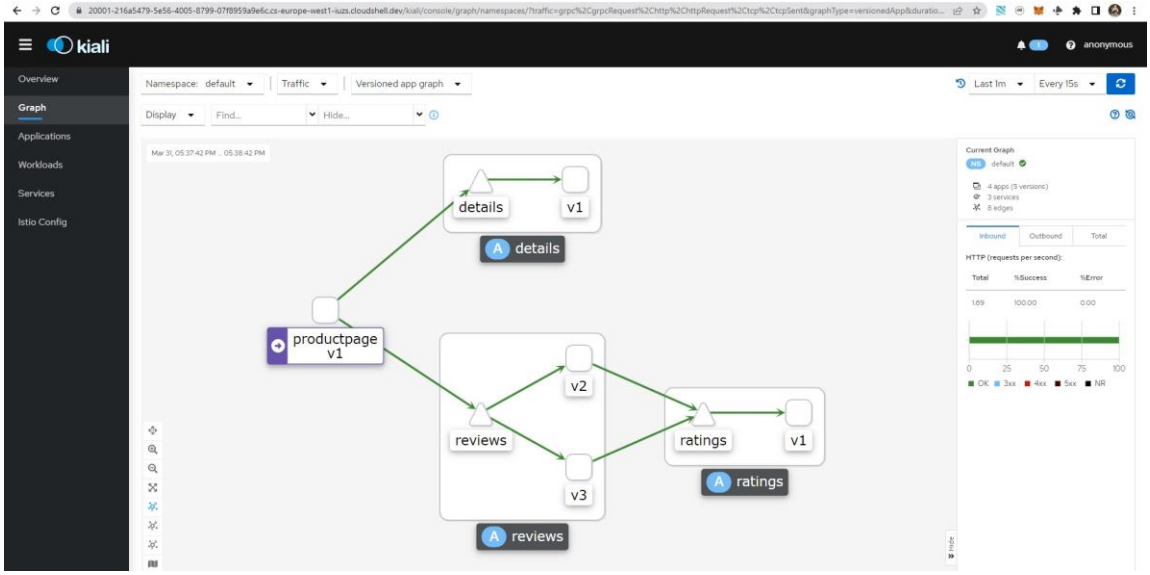


Ilustración 20: Arquitectura BookInfo

La Ilustración 20 muestra el detalle de la solución sobre la que implementaremos los controles de seguridad basados en mTLS y tráfico http. Básicamente es una aplicación html, que tiene embebidos cuatro servicios. El detalle de esta ilustración nos muestra en tiempo de ejecución el detalle de las comunicaciones que se dan entre los diferentes servicios implementados.

5.1. Visión funcional de la aplicación

Tal y como comentábamos la aplicación tiene cuatro microservicios con la siguiente funcionalidad:

- **Productpage**: página **html principal** que realiza la llamada a los microservicios de "details" y "reviews". Es la página que tiene embebida toda la funcionalidad, y en la que **se muestran los datos accedidos desde los microservicios** a los que se hace la llamada. (ver Ilustración 21)
- **Details**: microservicio con el **detalle de información de los libros** que se muestran en la página html. Este microservicio sólo puede recibir llamadas desde el microservicio de productpage. (parte izquierda de la Ilustración 21)
- **Reviews**: microservicio **con diferentes versiones implementadas** (ver Ilustración 22). Si bien no forma parte del alcance de este TFM, Istio ofrece **capacidades de gestión de tráfico en la arquitectura** de forma que puede gestionar tráfico entre diferentes versiones de software desplegado en la arquitectura. Funcionalmente describe el **detalle** de diferentes **referencias** a los libros. Las versiones implementadas pueden o no llamar al microservicio "ratings", según la versión implementada.
- **Ratings**: es un microservicio que **sólo puede ser accedido desde el microservicio "reviews"** en sus versiones de desarrollo 2 o 3. El rating muestra

un valor del libro con un **gráfico de estrellas**. Tanto el microservicio "ratings" como el microservicio "reviews", están en la parte derecha de la página principal.

La aplicación es **multilenguaje** y cada uno de los servicios está embebido en diferentes **contenedores**, de forma que nuevamente hacemos hincapié en la capacidad de **separación de funciones de la lógica de negocio implementada en cada uno de los servicios y la capacidad de gestión de las funciones de seguridad**.

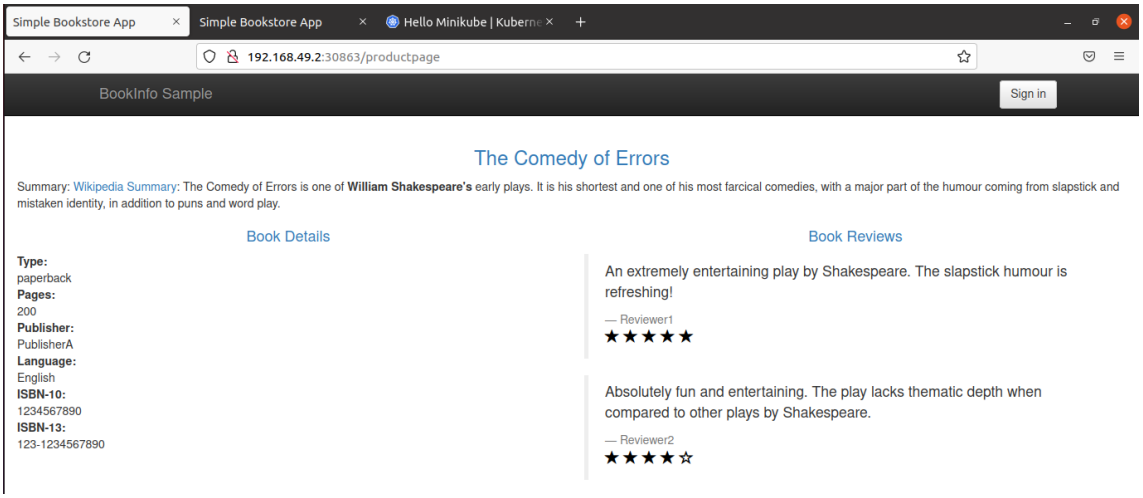


Ilustración 21: Página principal productpage

En el siguiente apartado se explica la implementación de la solución y cómo éste se publica a través de una dirección IP para su acceso. En la Ilustración 21, se visualiza un comportamiento de la aplicación en la que se está ejecutando la versión del microservicio "reviews". Tal y como se observa en la Ilustración 22, el flujo de llamadas que puede gestionar Istio dentro del aplicativo, justamene la versión 2 de este micorservicio es el que hace la llamada para pintar las estrellas en negro.

La versión 1 del microservicio, no tiene opción de llamada al microservicio "ratings" y por tanto no tiene implementada una valoración gráfica con estrellas. Y la versión 3, hace la llamada obteniendo el valor del rating del libro y pintando dicha valoración con estrellas rojas.

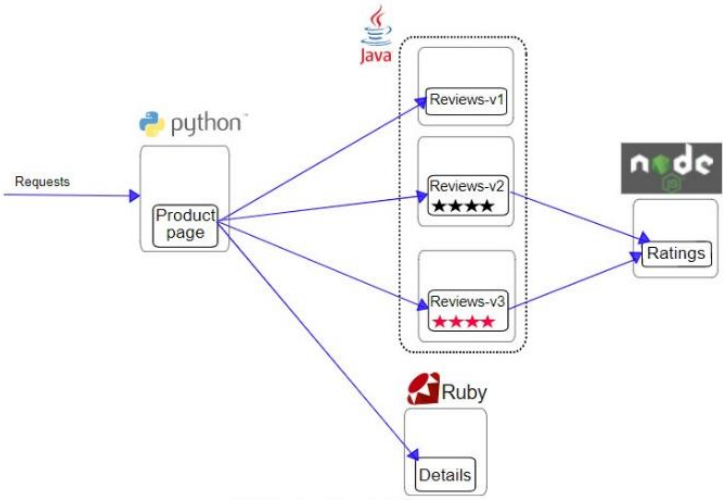


Ilustración 22: Flujo de llamadas de la aplicación BookInfo

Los siguientes apartados desarrollan la implantación de las diferentes funcionalidades de seguridad, sobre el aplicativo aquí detallado. De este modo, se presenta la

implantación de la herramienta Istio, del aplicativo BookInfo y de las funciones de seguridad mencionadas.

5.2. Visión técnica

Presentamos en este apartado, las necesidades y especificaciones necesarias para la instalación y publicación de la aplicación BookInfo. Básicamente se han seguido los pasos definidos por Istio para poder analizar y trabajar con su herramienta[11].

Se detallan por tanto los siguientes apartados:

- Instalación de Istio
- Instalación de la aplicación en kubernetes
- Publicación de la aplicación
- Visualización del dashboard

Para la instalación de este aplicativo, he utilizado una **máquina virtual Ubuntu** con **tres procesadores y 8182 MB de memoria**. Teniendo en cuenta que tendremos instalado **Minikube** dentro de la máquina, quizás se pueda realizar este mismo trabajo con algún recurso algo más holgado, más procesador y más memoria…

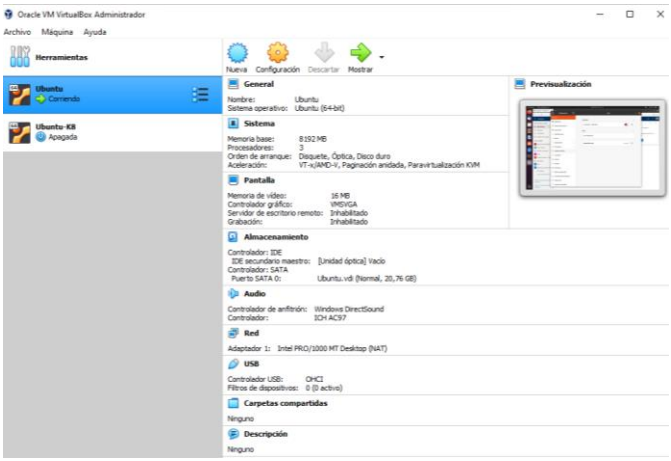


Ilustración 23: Características de la máquina virtual.

Durante el proceso de instalación se revisará todo aquel **software base** necesario para que el aplicativo funcione correctamente. Para ello, el usuario utilizado (jgarciavelasco) en todo el proceso tiene **permisos de administrador** sobre la máquina.

Minikube forma parte del software base implantado y para poder trabajar con el aplicativo, deberemos tenerlo instalado. Gran parte de los comandos utilizados son propios de Kubernetes.

En cualquier caso, es necesario poder iniciar Minikube y trabajar con él.

El proceso se define paso a paso, de modo que se puede replicar con las mismas opciones descritas en el TFM.

[11] *Getting started* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://istio.io/latest/docs/setup/getting-started/

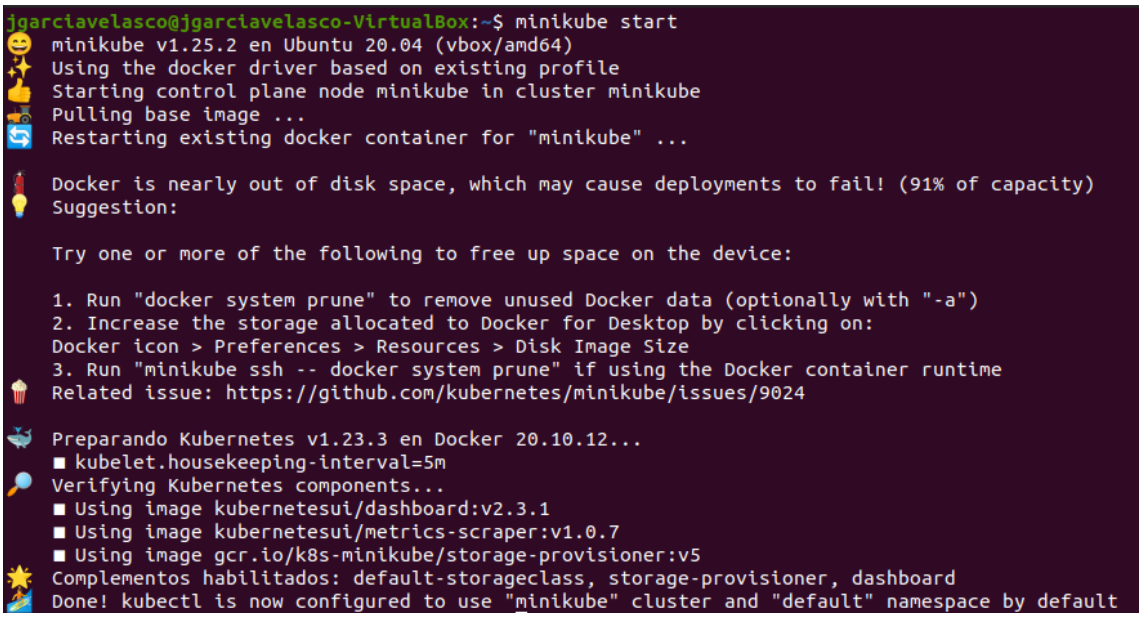


Ilustración 24: Minikube start

5.2.1. Instalación de Istio

El **proceso de instalación** se puede realizar de diferentes formas, si bien aquí describiremos el proceso que comienza con la descarga de la última versión disponible y su instalación en la máquina servidora (en nuestro caso, la máquina virtual Ubuntu).

Utilizaremos el siguiente comando para realizar la **descarga de Istio**.

$$\text{curl -L } https://istio.io/downloadIstio \mid sh -$$

Con este comando nos descargamos la última versión de Istio, que tal y como podemos ver en la Ilustración 25 es la **versión istio-1.13.4**. Durante la realización del TFM, también he trabajado con la versión anterior, si bien para el objeto del TFM no hay diferencias de funcionalidad y no altera los resultados del mismo.

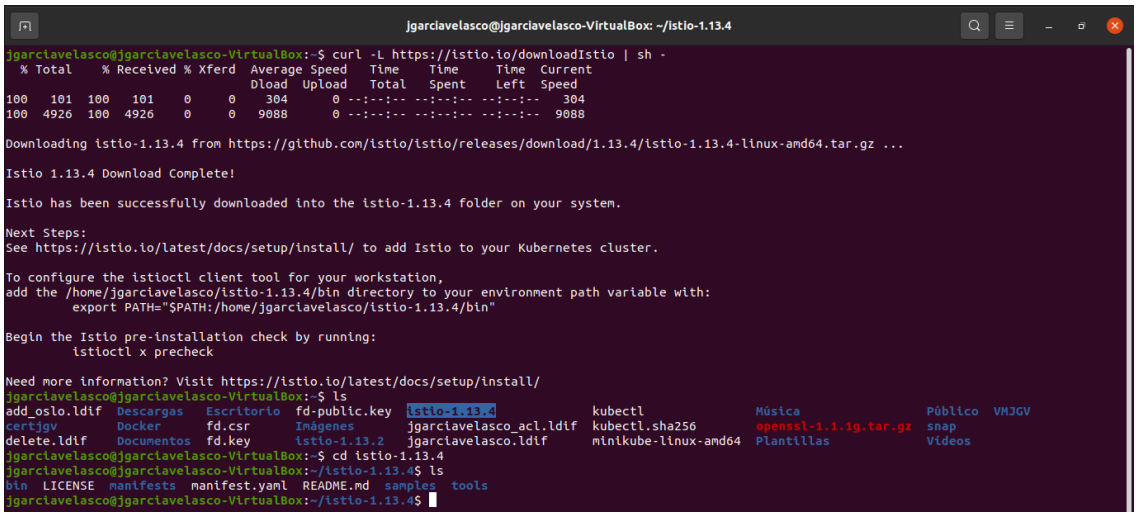


Ilustración 25: Download de Istio

Tal y como se puede ver en la descarga de la herramienta, tenemos en el directorio de la misma diferente información, entre las cuales tenemos algunos ejemplos de

base para poder probar la herramienta (/samples) y el ejecutable del cliente Istion en la carpeta /bin.

Añadimos este ejecutable al path de Linux `export PATH=$PWD/bin:$PATH` y realizamos la instalación de Istio con un perfil demo `istioctl install --set profile=demo -y`

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ export PATH=$PWD/bin:$PATH
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ istioctl install --set profile=demo -y
WARNING: Istio control planes installed: 1.13.2.
WARNING: A newer installed version of Istio has been detected. Running this command will overwrite it.
✔ Istio core installed
✔ Istiod installed
✔ Ingress gateways installed
✔ Egress gateways installed
✔ Installation complete
Making this installation the default for injection and validation.

Thank you for installing Istio 1.13.  Please take a few minutes to tell us about your install/upgrade experience!  https://forms.gle/pzWZpAvMVBecaQ9h9
```

Ilustración 26: Instalación de Istio

Tal y como se ha comentado anteriormente, se detecta una versión ya instalada de Istio. En cualquier caso, ésta se sobrescribe y se finaliza la instalación.

5.2.2. Instalación de la aplicación en kubernetes

Durante el desarrollo de la práctica se ha comentado que Istio puede trabajar en entornos basados en arquitecturas de microservicios sobre contenedores.

Hacemos el **despliegue del aplicativo sobre una estructura de MiniKube** con el siguiente comando:

`kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml`

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
service/details unchanged
serviceaccount/bookinfo-details unchanged
deployment.apps/details-v1 unchanged
service/ratings unchanged
serviceaccount/bookinfo-ratings unchanged
deployment.apps/ratings-v1 unchanged
service/reviews unchanged
serviceaccount/bookinfo-reviews unchanged
deployment.apps/reviews-v1 unchanged
deployment.apps/reviews-v2 unchanged
deployment.apps/reviews-v3 unchanged
service/productpage unchanged
serviceaccount/bookinfo-productpage unchanged
deployment.apps/productpage-v1 unchanged
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl get services
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
details          ClusterIP   10.106.152.31    <none>        9080/TCP         11h
hello-minikube   NodePort    10.108.14.73     <none>        8080:32174/TCP   42d
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP          47d
productpage      ClusterIP   10.99.41.143     <none>        9080/TCP         11h
ratings          ClusterIP   10.105.105.227   <none>        9080/TCP         11h
reviews          ClusterIP   10.99.231.199    <none>        9080/TCP         11h
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl get pods
NAME                              READY   STATUS    RESTARTS      AGE
details-v1-5498c86cf5-wkmmm       2/2     Running   0             11h
hello-minikube-7bc9d7884c-gscd7   2/2     Running   4 (22d ago)   42d
productpage-v1-65b75f6885-rkvnr   2/2     Running   0             11h
ratings-v1-b477cf6cf-k5w25        2/2     Running   0             11h
reviews-v1-79d546878f-trlln       2/2     Running   0             11h
reviews-v2-548c57f459-zsnrb       2/2     Running   0             11h
reviews-v3-6dd79655b9-b5c5q       2/2     Running   0             11h
```

Ilustración 27: Instalación de la aplicación BookInfo sobre kubernetes

Se muestra en la Ilustración 28, la misma instalación que en la ilustración previa, ya que se puede apreciar en la anterior que la instalación ya estaba realizada con una versión anterior y así se corrobora en esta ilustración.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
deployment.apps/productpage-v1 created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ kubectl get pods
NAME                              READY   STATUS    RESTARTS      AGE
details-v1-5498c86cf5-wkmmm       1/2     Running   0             8s
hello-minikube-7bc9d7884c-gscd7   2/2     Running   4 (21d ago)   41d
productpage-v1-65b75f6885-rkvnr   1/2     Running   0             8s
ratings-v1-b477cf6cf-k5w25        2/2     Running   0             8s
reviews-v1-79d546878f-trlln       1/2     Running   0             8s
reviews-v2-548c57f459-zsnrb       1/2     Running   0             8s
reviews-v3-6dd79655b9-b5c5q       1/2     Running   0             8s
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ kubectl get services
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
details          ClusterIP   10.106.152.31    <none>        9080/TCP         23s
hello-minikube   NodePort    10.108.14.73     <none>        8080:32174/TCP   41d
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP          46d
productpage      ClusterIP   10.99.41.143     <none>        9080/TCP         23s
ratings          ClusterIP   10.105.105.227   <none>        9080/TCP         23s
reviews          ClusterIP   10.99.231.199    <none>        9080/TCP         23s
```

Ilustración 28: Instalación de BookInfo

Se observa que se han generado los servicios y los pods que atienden a estos servicios, tal y como se había detallado en el apartado (ver apartado 5.1) de definición funcional del aplicativo.

5.2.3. Publicación de la aplicación

Ahora mismo tenemos una aplicación que no se puede ver desde ningún navegador, no está publicada. Tendremos que hacer que la aplicación se publique y sea visible a través de alguna dirección IP

Lo primero que hacemos es **abrir el tráfico de la aplicación** a través de un **Gateway** con el siguiente comando:

kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
gateway.networking.istio.io/bookinfo-gateway created
virtualservice.networking.istio.io/bookinfo created
```

Ilustración 29: Apertura de tráfico a través de un Gateway

Una vez tenemos el Gateway abierto, debemos publicar el aplicativo en una **dirección IP y un puerto determinado**, de forma que podamos llamar al aplicativo e interactuar con él.

Los siguientes comandos ejecutarán la definición de una IP y puerto de acceso a la aplicación, según el siguiente orden de ejecución:

Se definen las variables INGRESS_PORT y SECURE_INGRESS_PORT

export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')

export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')

En la Ilustración 30 se observa como toman estos valores donde INGRESS_PORT=30863 y SECURE_INGRESS_PORT=30479

Por último, abrimos un tunnel con el comando *minikube tunnel*, en una terminal diferenciada, ya que ésta estará corriendo sobre el servidor como un proceso demonio para garantizar el acceso al aplicativo.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{
.spec.ports[?(@.name=="http2")].nodePort}')
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o json
path='{.spec.ports[?(@.name=="https")].nodePort}')
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ echo "$INGRESS_PORT"
30863
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ echo "$SECURE_INGRESS_PORT"
30479
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ export INGRESS_HOST=$(minikube ip)
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ echo "$INGRESS_HOST"
192.168.49.2
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.2$ minikube tunnel
[sudo] contraseña para jgarciavelasco:
Status:
        machine: minikube
        pid: 14575
        route: 10.96.0.0/12 -> 192.168.49.2
        minikube: Running
        services: [istio-ingressgateway]
    errors:
                minikube: no errors
                router: no errors
                loadbalancer emulator: no errors
```

Ilustración 30: Apertura de una IP y puerto

En la Ilustración 30, se puede observar cómo al abrir el tunnel, se ejecuta un proceso (pid 14575) en la máquina servidora, donde se produce un enrutado al servidor Minikube con el servicio de Kubernetes en la dirección IP 10.96.0.0/12 y que lo enruta a la dirección IP que obtendremos en los siguientes pasos.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ echo "$GATEWAY_URL"
192.168.49.2:30863
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ echo "http://$GATEWAY_URL/productpage"
http://192.168.49.2:30863/productpage
```

Ilustración 31: Obtenemos la dirección IP de acceso al aplicativo.

5.2.4. ¿Qué tenemos ahora mismo?

Hasta este punto, el aplicativo está funcionando, pero no lo hace en modo Kubernetes, sino que ya está instalado conjuntamente con Istio.

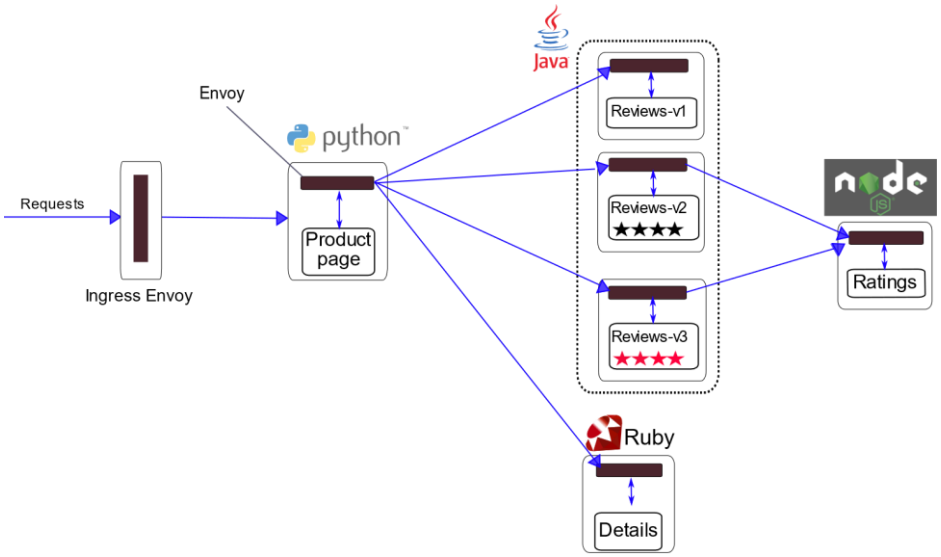


Ilustración 32: Aplicación con el Data Plane (Envoy)

La ilustración complementa la misma que hemos visto en el apartado de explicación funcional del aplicativo. En esta nueva ilustración se añade el adjunto "Envoy" al microservicio. Este adjunto forma parte del Data Plane, que ya hemos visto también en la parte de definición de la arquitectura.

Tiene lógica que esto sea así, ya que de esta forma estamos conformando la arquitectura, que a continuación volvemos a representar y que de alguna forma completa la explicación de cómo una arquitectura de microservicios se puede segmentar en sus funciones frente a la aplicación de medidas técnicas de seguridad.

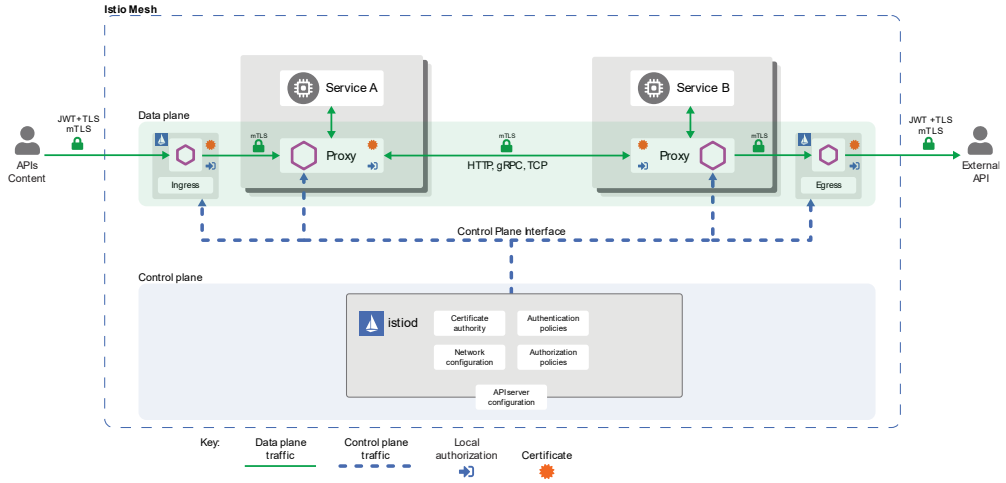


Ilustración 33: Arquitectura Istio

Los servicios implementados en la aplicación tienen ahora los dos planos de gestión de Istio.

5.2.5. Dashboard

Conceptualmente se entiende que efectivamente se cumple lo que teóricamente ya habíamos apuntado. Los planos de Istio están implementados, pero no los vemos, el comportamiento de la aplicación es el mismo que sin Istio (salvo un apunte que ahora entramos a detallar).

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f samples/addons
serviceaccount/grafana unchanged
configmap/grafana unchanged
service/grafana unchanged
deployment.apps/grafana configured
configmap/istio-grafana-dashboards configured
configmap/istio-services-grafana-dashboards configured
deployment.apps/jaeger unchanged
service/tracing unchanged
service/zipkin unchanged
service/jaeger-collector unchanged
serviceaccount/kiali unchanged
configmap/kiali unchanged
clusterrole.rbac.authorization.k8s.io/kiali-viewer unchanged
clusterrole.rbac.authorization.k8s.io/kiali unchanged
clusterrolebinding.rbac.authorization.k8s.io/kiali unchanged
role.rbac.authorization.k8s.io/kiali-controlplane unchanged
rolebinding.rbac.authorization.k8s.io/kiali-controlplane unchanged
service/kiali unchanged
deployment.apps/kiali unchanged
serviceaccount/prometheus unchanged
configmap/prometheus unchanged
clusterrole.rbac.authorization.k8s.io/prometheus unchanged
clusterrolebinding.rbac.authorization.k8s.io/prometheus unchanged
service/prometheus unchanged
deployment.apps/prometheus configured
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl rollout status deployment/kiali -n istio-system
deployment "kiali" successfully rolled out
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ istioctl dashboard kiali
http://localhost:20001/kiali
```

Ilustración 34: Instalación del DashBoard

Mediante el siguiente comando, instalamos los addons del Dashboard Kiali[12], que se apoya en diferentes aplicaciones como Grafana, Jaeger y Prometheus.

kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.13/samples/addons/kiali.yaml

[12] *Software Kiali* [en línea] [fecha de consulta: 29 de marzo de 2022]. Disponible en: https://github.com/istio/istio/tree/release-1.13/samples/addons

Una vez instalado Kiali y sus addons, ejecutamos Kiali siguiendo los pasos que se muestran en la Ilustración 34. Como se puede observar, en el último paso se hace una llamada al dashboard, lo cual nos muestra la siguiente pantalla.

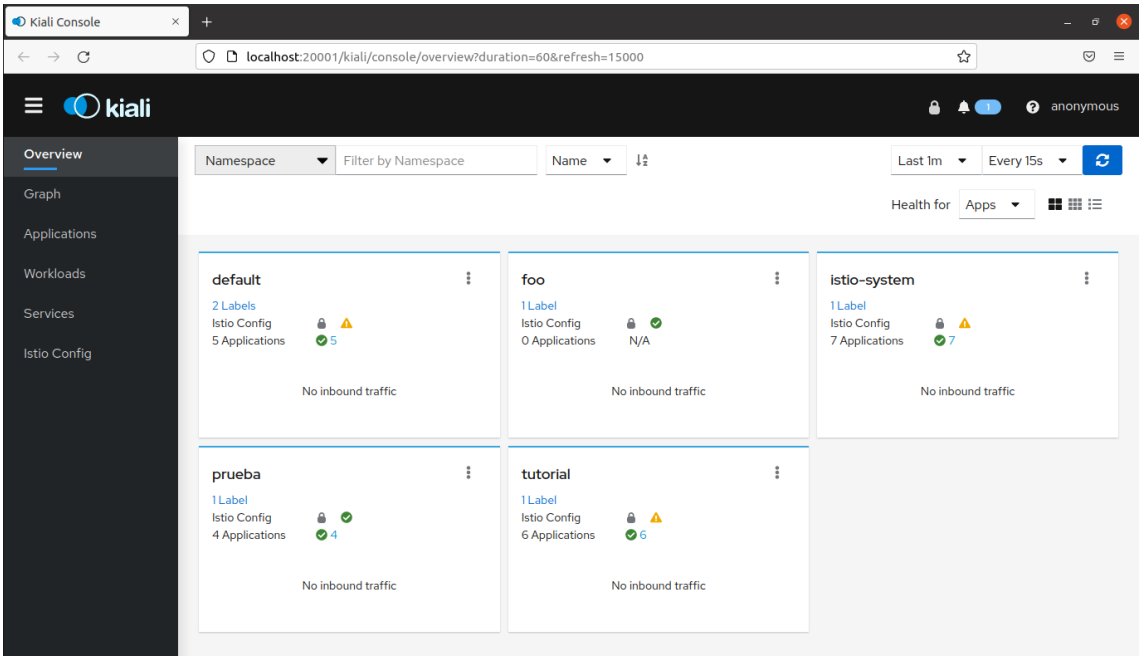


Ilustración 35: DashBoard Kiali

Con este Dashboard podremos ver en la práctica cómo funciona Istio y cómo efectivamente tenemos un aplicativo configurado con Istio. Como se puede observar tenemos varias aplicaciones en vuelo, pero la nuestra es la que está bajo el label default. Recordemos que en la fase de instalación hemos dotado de un namespace by default.

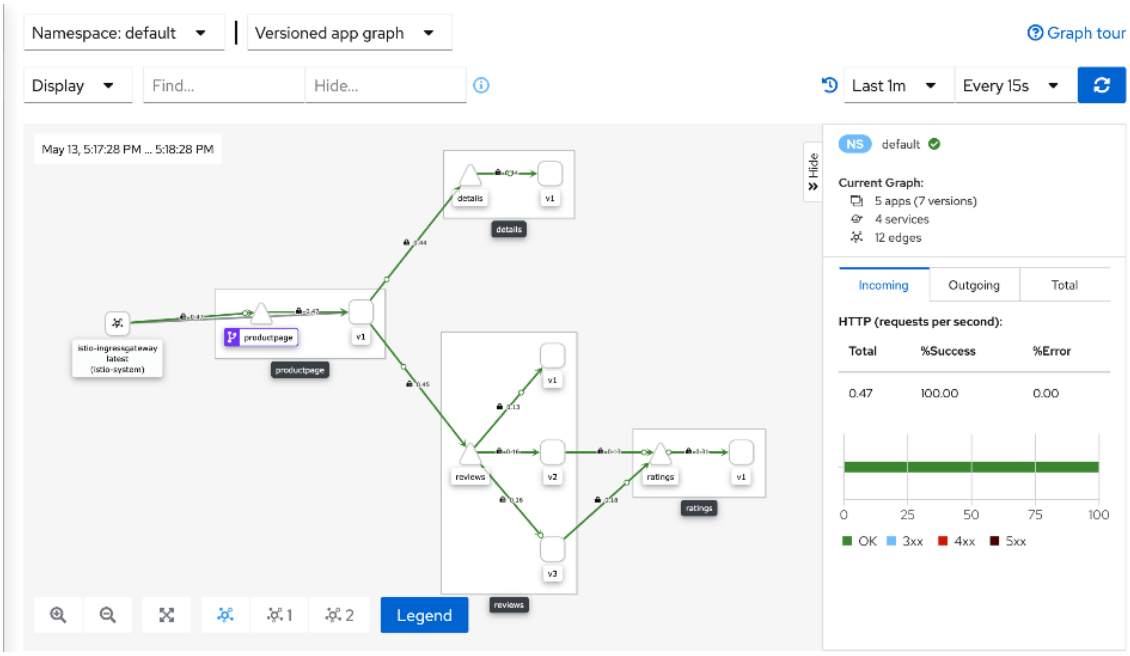


Ilustración 36: Monitorización con Dashboard Kiali.

Si bien ahora entramos en algunos conceptos de mayor detalle, en el dashboard podemos revisar algunas configuraciones, como por ejemplo la habilitación de tráfico TLS.

En la Ilustración 35, se observan diferentes candados, tanto a nivel de namespace, como a nivel de Istio Mesh (parte de arriba). Por defecto, la configuración completa de la plataforma instalada conlleva la instalación y gestión de tráfico con mTLS. Esto es lo que marca el dashboard cuando entramos a monitorizar la plataforma.

5.3. Implantación de mTLS

Aunque según hemos visto en el apartado anterior, esta capacidad ya viene instalada por defecto cuando tenemos la aplicación en Istio, en este apartado definimos el detalle de las diferentes configuraciones que se pueden realizar y cómo ejecutarlas en la plataforma.

5.3.1. Arquitectura de autenticación

La siguiente ilustración muestra la arquitectura de autenticación, nuevamente apoyándonos en ambos planos (data y control). Las políticas definidas se implementan sobre el plano de control y es éste el que las implementa en los diferentes proxy (Envoy) de los difentes microservicios.

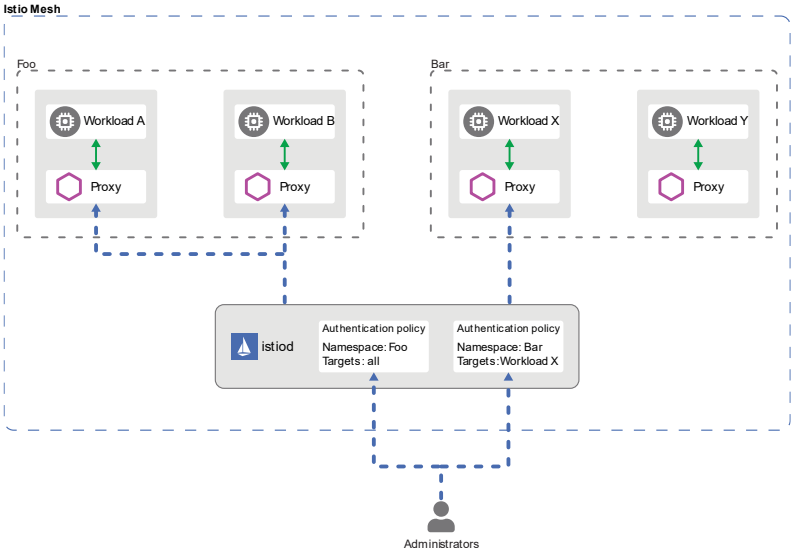


Ilustración 37: Arquitectura de autenticación.

Por defecto ya sabemos que la configuración de tráfico entre los diferentes microservicios es con mTLS. Pero, ¿cómo se configura esta política?

5.3.2. Configuración de la política mTLS

Un primer modo de configuración es aplicando una política restrictiva de mTLS. En la configuración por defecto, los microservicios pueden seguir recibiendo comunicaciones en texto plano, sin cifrar. Si aplicamos una política restrictiva, únicamente podrán recibir tráfico cifrado, mediante TLS.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "default"
  namespace: "default"
spec:
  mtls:
    mode: STRICT
EOF
peerauthentication.security.istio.io/default created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 38: Aplicación de mTLS estricto.

La aplicación de esta política es la presentada en la ilustración anterior, donde se puede observar que aplicamos la política a nivel de namespace, concretamente el de default, que es dónde tenemos ubicada la aplicación BookInfo.

Revisando el DashBoard y mirando el detalle de uno de los microservicios, podemos observar cómo efectivamente está configurando el tráfico mTLS.

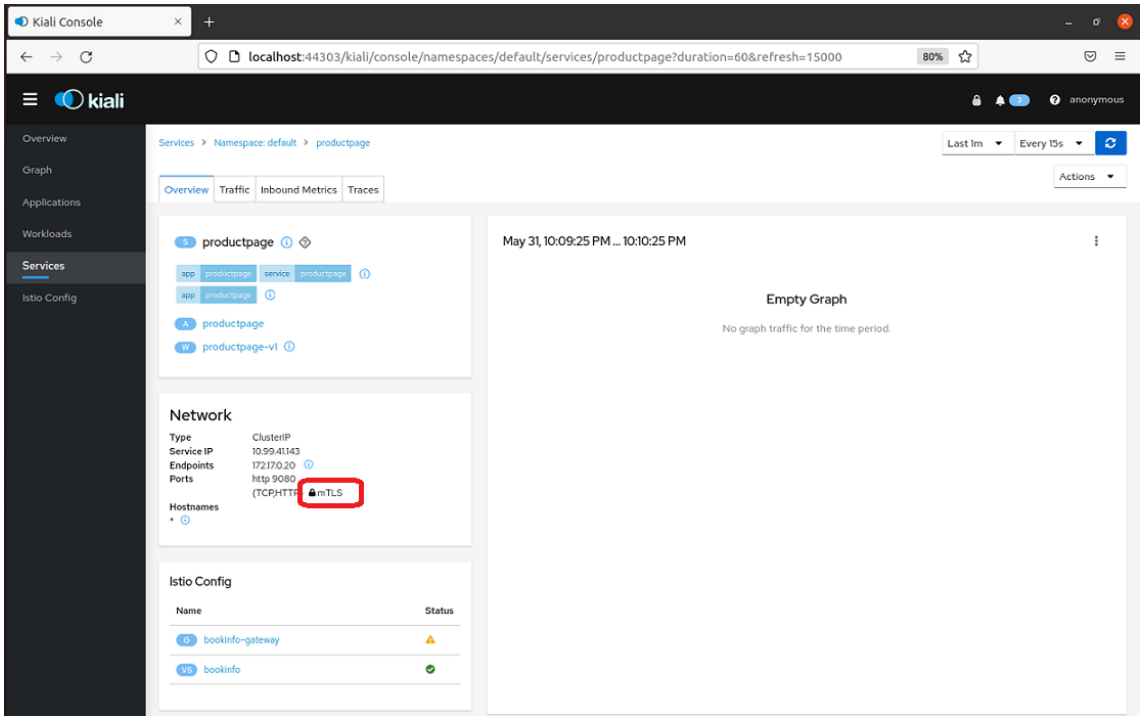


Ilustración 39: Implantación de configuración mTLS

Del mismo modo que se lo podemos aplicar a nivel de namespace, podemos hacer lo mismo para el resto de nodos de la arquitectura, y buscando obtener políticas de diferentes aplicaciones, por workload, por aplicaciones, por acceso de usuario…

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ cat <<EOF | kubectl apply -n default -f -
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "productpage-v1"
  namespace: "default"
spec:
  selector:
    matchLabels:
      app: httpbin
  mtls:
    mode: STRICT
EOF
peerauthentication.security.istio.io/productpage-v1 created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 40: Aplicación de mTLS en un Workload específico

5.4. Implantación de tráfico http basado en TLS

Una vez hemos visto cómo se realiza la configuración de tráfico cifrado entre los servicios que componen la arquitectura, vamos a realizar una configuración de tráfico http, basado precisamente en ese tráfico cifrado TLS previamente configurado.

Iremos haciendo una configuración de diferentes políticas que iremos aplicando paulatinamente y visualizando en el Dashboard de Istio. Comenzaremos por una regla muy restrictiva que deniega cualquier acceso al aplicativo.

5.4.1. Allow-nothing policy

Con la aplicación de esta política denegamos cualquier acceso al servicio, por este motivo la hemos denominado Allow-nothing. Al no tener ninguna especificación, se restringe cualquier acceso al namespace.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
> apiVersion: security.istio.io/v1beta1
> kind: AuthorizationPolicy
> metadata:
>   name: allow-nothing
>   namespace: default
> spec:
>   {}
> EOF
authorizationpolicy.security.istio.io/allow-nothing created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 41: Aplicación de la regla de denegación de tráfico.

Esta regla, efectivamente se aplica en la plataforma, de forma que podemos ver su detalle en el dashboard.

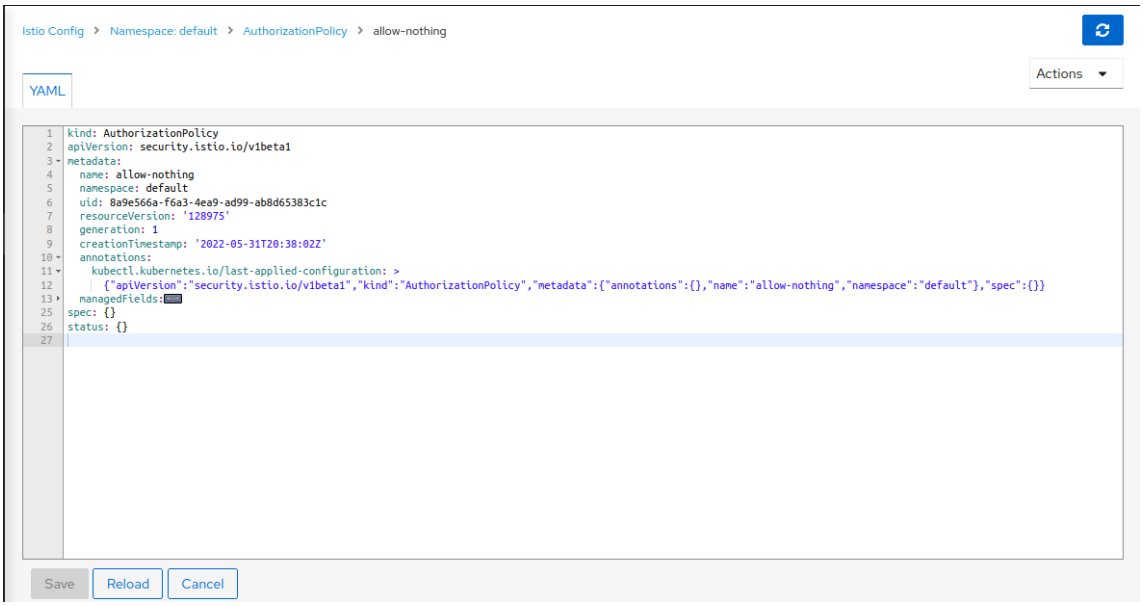


Ilustración 42: Dashboard con la política aplicada.

5.4.2. Product-page viewer policy

Con esta política permitimos realizar una operación GET sobre el namespace, de forma que cualquier origen podrá realizar esta acción sobre el aplicativo.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
> apiVersion: security.istio.io/v1beta1
> kind: AuthorizationPolicy
> metadata:
>   name: "productpage-viewer"
>   namespace: default
> spec:
>   selector:
>     matchLabels:
>       app: productpage
>   action: ALLOW
>   rules:
>   - to:
>     - operation:
>         methods: ["GET"]
> EOF
authorizationpolicy.security.istio.io/productpage-viewer created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 43: Aplicación de la regla Product-Page viewer

Que al igual que la anterior, también se visualiza en el dashboard. En el comando de aplicación de la política, se puede observar cómo la regla se aplica efectivamente al microservicio productpage (app: productpage)

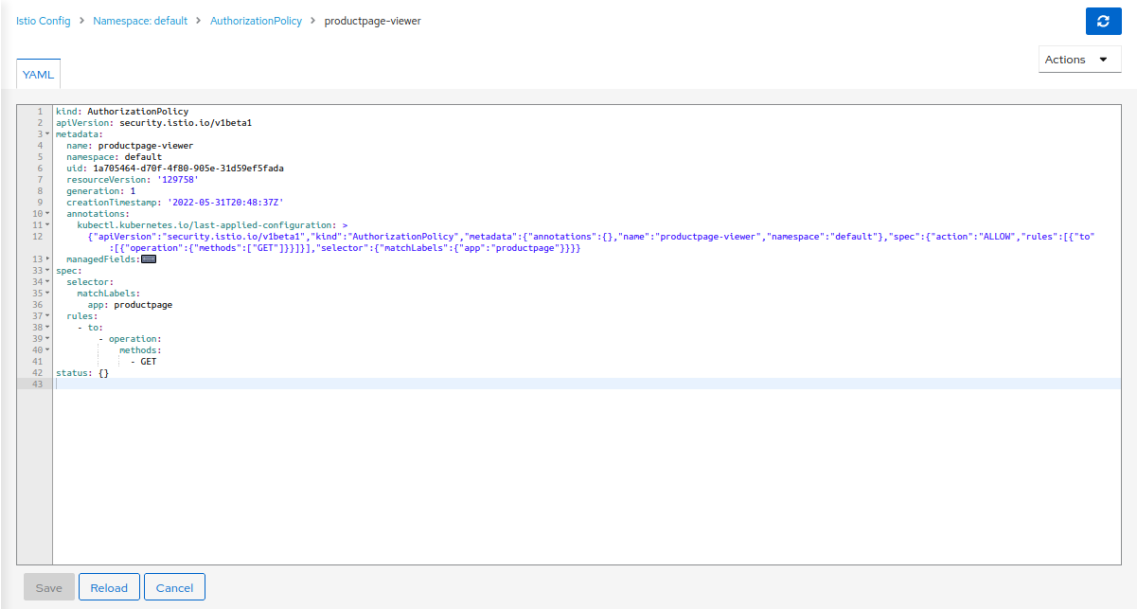


Ilustración 44: Dashboard con la aplicación de la política

5.4.3. Details viewer policy

Haremos lo mismo con la aplicación de la política de acceso al microservicio Details, haciendo que el único acceso permitido sea a través del microservicio productpage. Comenzamos a configurar una política de acceso gestionada desde el plano de control de Istio, basado en políticas de acceso a través de canales seguros.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
> apiVersion: security.istio.io/v1beta1
> kind: AuthorizationPolicy
> metadata:
>   name: "details-viewer"
>   namespace: default
> spec:
>   selector:
>     matchLabels:
>       app: details
>   action: ALLOW
>   rules:
>   - from:
>     - source:
>         principals: ["cluster.local/ns/default/sa/bookinfo-productpage"]
>     to:
>     - operation:
>         methods: ["GET"]
> EOF
authorizationpolicy.security.istio.io/details-viewer created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 45: Aplicación de la política de acceso a Details

De la misma forma, podemos examinar su aplicación a través del Dashboard, tal y como se muestra en la siguiente ilustración.

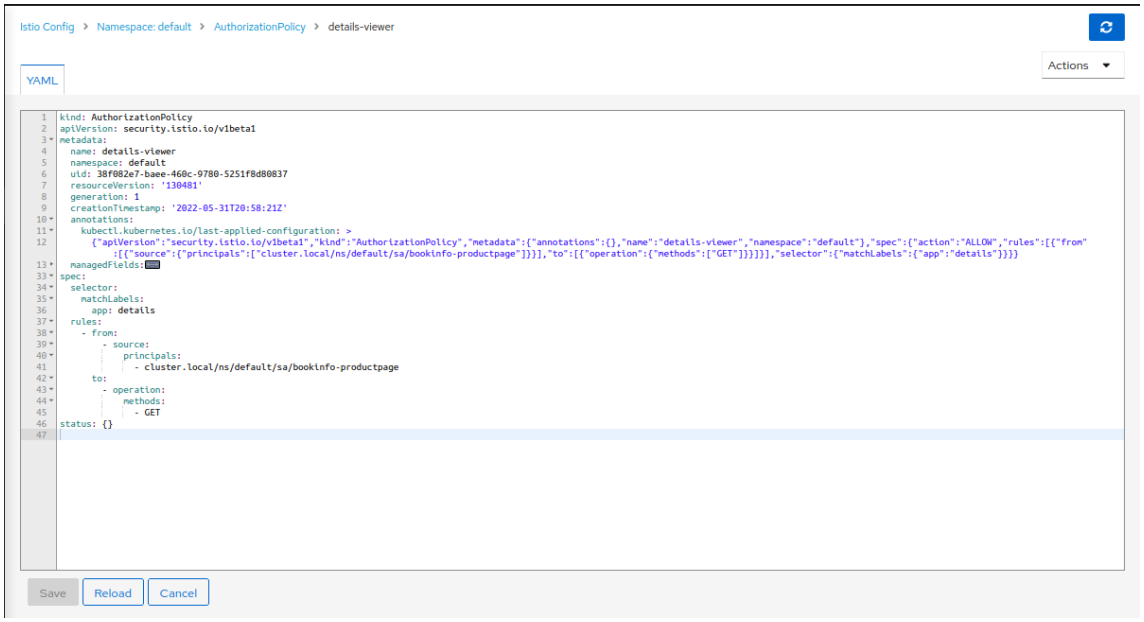


Ilustración 46: Política aplicada y visualizada en el Dashboard.

5.4.4. Reviews viewer policy

Al igual que hemos implementado el acceso al microservicio de detail, tenemos que dar acceso a productpage al microservicio reviews. La siguiente ilustración muestra esta configuración.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
> apiVersion: security.istio.io/v1beta1
> kind: AuthorizationPolicy
> metadata:
>   name: "reviews-viewer"
>   namespace: default
> spec:
>   selector:
>     matchLabels:
>       app: reviews
>   action: ALLOW
>   rules:
>   - from:
>     - source:
>         principals: ["cluster.local/ns/default/sa/bookinfo-productpage"]
>     to:
>     - operation:
>         methods: ["GET"]
> EOF
authorizationpolicy.security.istio.io/reviews-viewer created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 47: Aplicación de la política reviews-viewer

Nuevamente al aplicar esta regla, únicamente se da permiso de acceso a productpage, a través del método GET. Tal y como se puede ver en la siguiente ilustración, esta configuración también se puede visualizar en el Dashboard de Kiali.

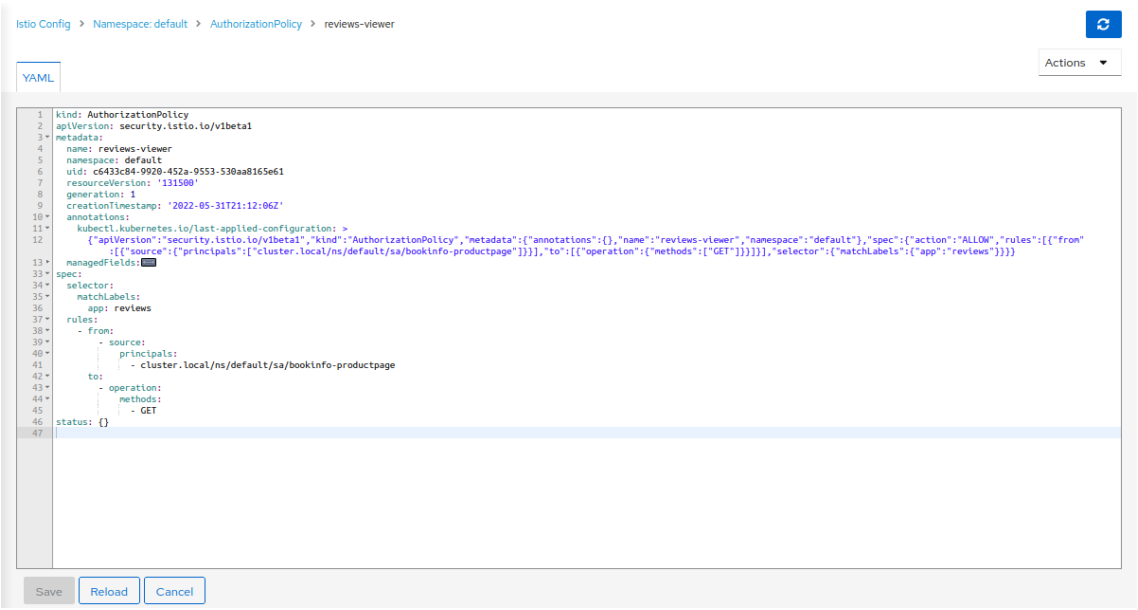


Ilustración 48: Implementación de la política visualizada en el Dashboard

5.4.5. Ratings-viewer policy

Tal y como hemos ido revisando a lo largo de todo el documento, efectivamente ahora nos queda el último acceso a implementar para el último microservicio. Tal y como hemos visto en las diferentes ilustraciones donde se visualizaba cómo trabaja la aplicación Bookinfo, el acceso a ratings únicamente estará permitido al servicio reviews.

De esta forma, tenemos una aplicación totalmente gestionada en base a configuraciones de desarrollo en Istio, y totalmente separadas de la implementación funcional de la aplicación.

En la siguiente ilustración se muestra la configuración correspondiente a la última política.

```
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$ kubectl apply -f - <<EOF
> apiVersion: security.istio.io/v1beta1
> kind: AuthorizationPolicy
> metadata:
>   name: "ratings-viewer"
>   namespace: default
> spec:
>   selector:
>     matchLabels:
>       app: ratings
>   action: ALLOW
>   rules:
>   - from:
>     - source:
>         principals: ["cluster.local/ns/default/sa/bookinfo-reviews"]
>     to:
>     - operation:
>         methods: ["GET"]
> EOF
authorizationpolicy.security.istio.io/ratings-viewer created
jgarciavelasco@jgarciavelasco-VirtualBox:~/istio-1.13.4$
```

Ilustración 49: Aplicación de la última política

Al igual que en las políticas anteriores, se muestra su implementación a través del dashboard de Kiali.

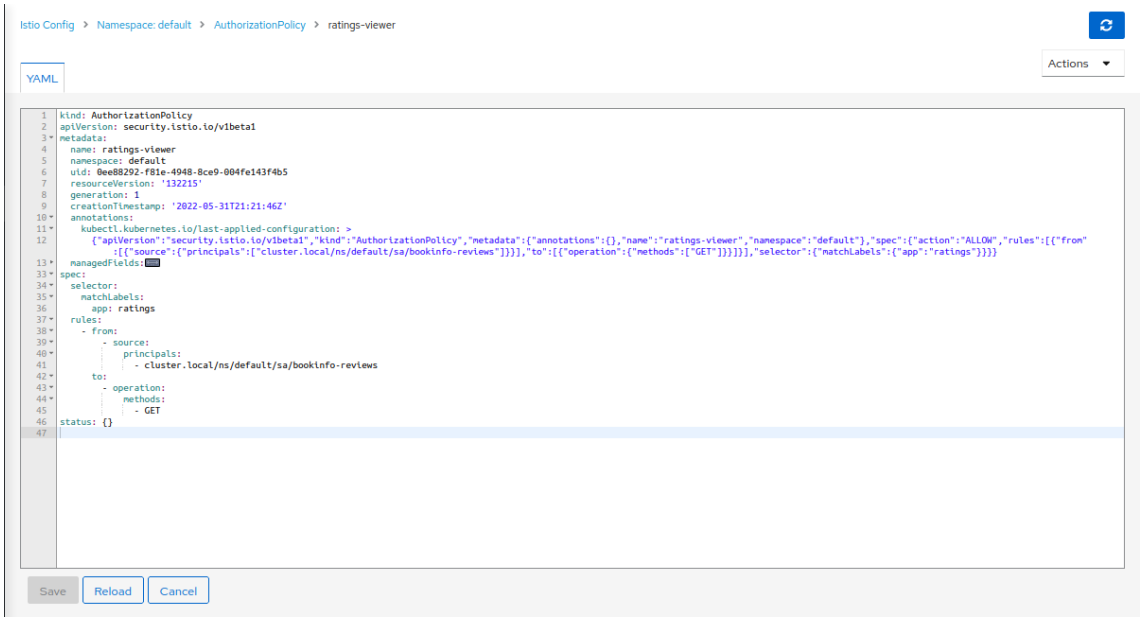


Ilustración 50: Dashboard de Kiali con la última política implementada.

5.5. Visión global de la aplicación de políticas

Por último y a modo de resumen de lo que sucede en este tipo de arquitecturas, se muestra una visión global de las políticas aplicadas en la arquitectura, que permiten como corolario final ver de un vistazo todas las políticas aplicadas en base al workload o microservicio que le corresponde.

De esta forma, se demuestra el objetivo del TFM, observando como Istio puede gestionar en base a políticas de desarrollo a través de su plano de control y su

aplicación en el plano de datos, el control de accesos a los diferentes microservicios y la comunicación a través de flujos de comunicación seguros.

El control de acceso lo hemos realizado a través de políticas de control de acceso. La aplicación de flujos de comunicación seguros lo hemos conseguido con aplicación de flujos basados en mTLS y por tanto con canales de comunicación cifrados.

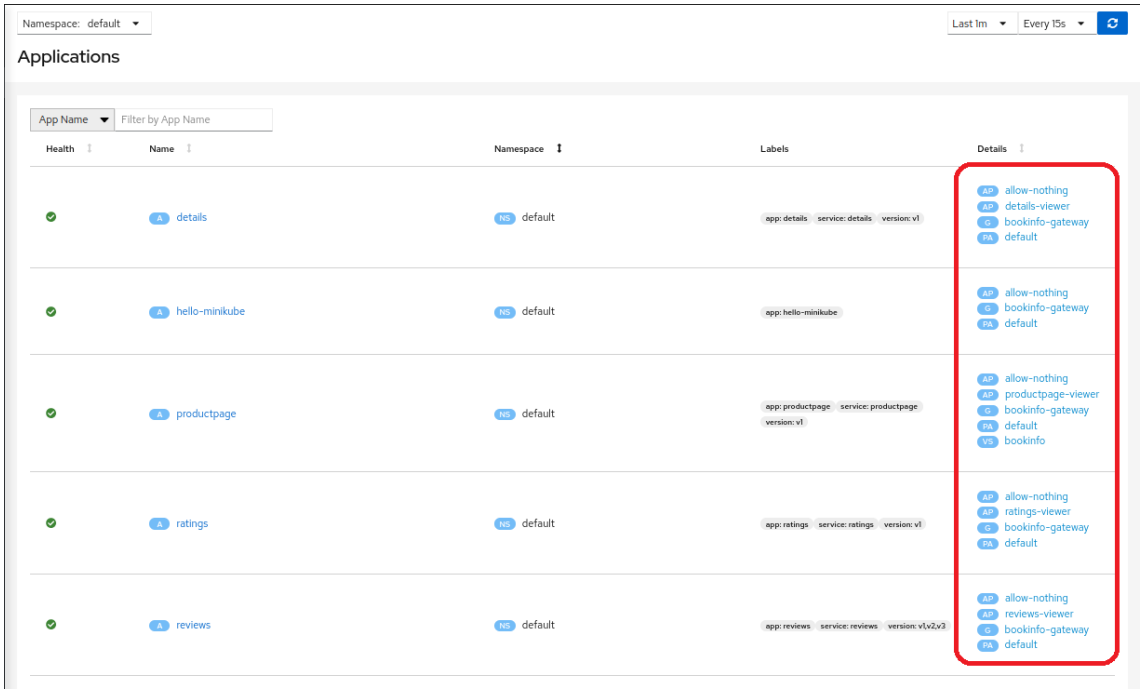


Ilustración 51: Implementación de las políticas por microservicio.

6. Instalación de Istio en GCP

Como prueba de concepto y tal y como se podrá ver en este apartado, se ha instalado Istio en una instancia de GCP, pudiendo instalar una aplicación por defecto para poder revisar un ejemplo de una arquitectura service mesh funcionando en la plataforma de Google Cloud (GCP).

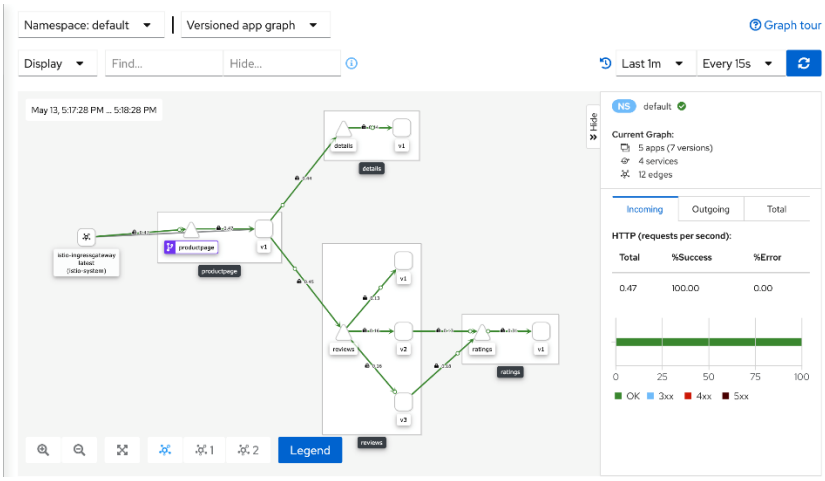


Ilustración 52: Visión en Kiali de la aplicación por defecto instalada.

6.1. Google Cloud Platform API Gateway

Para la realización de las primeras pruebas de concepto, he instalado una plataforma en Google Kubernetes Engine, la cual se puede gestionar a través del API Gateway mediante un proyecto.

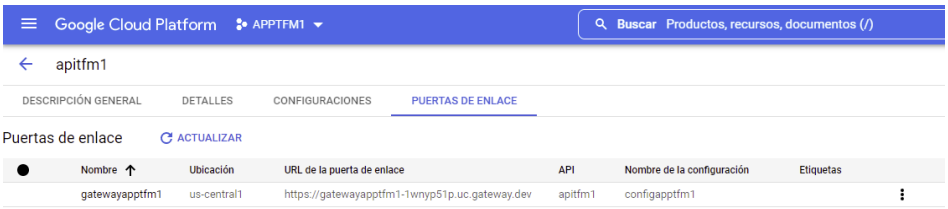


Ilustración 53: Puerta de enlace del proyecto APPTFM1

En este caso, el proyecto es APPTFM1, tal y como se muestra en la siguiente pantalla. Desde aquí se pueden gestionar diferentes capacidades, como por ejemplo la propia de Kubernetes Engine, donde se pueden ver las aplicaciones desplegadas de la solución por defecto de Istio.

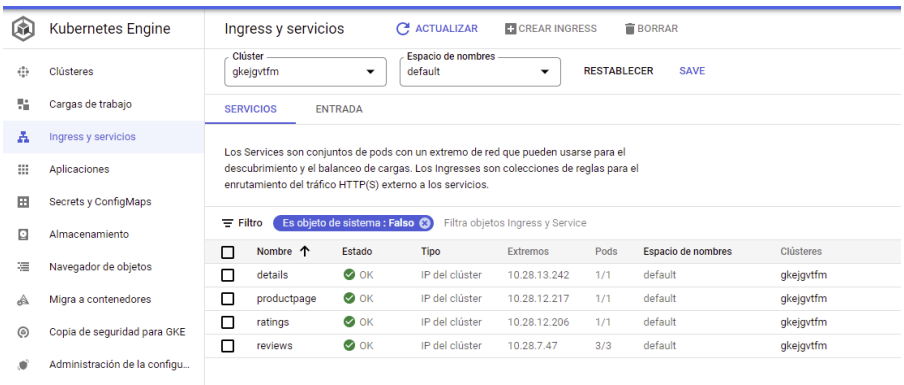


Ilustración 54: Cluster de kubernetes gkejgvtfm

7. Conclusiones finales

Si bien a lo largo de la memoria se ha desarrollado un racional de las necesidades que se pretendían cubrir con este TFM, se presentan a continuación las conclusiones finales del proyecto a modo resumen.

- Las arquitecturas orientadas a microservicios ofrecen una posibilidad de diseño en base a la ciberseguridad que en otro tipo de arquitecturas son más difíciles de diseñar, e incluso llegar a pensar.
- Soluciones como Istio ofrecen capacidades que abarcan un abanico de medidas técnicas que permiten ofrecer aplicaciones más seguras.
- Las soluciones en cloud tienen conceptos y tareas comunes, que ofrecen la posibilidad de monitorizar soluciones basadas en microservicios. Estas mismas soluciones, apoyadas en herramientas como Istio ofrecen capas de seguridad que otro tipo de arquitecturas no se podían diseñar.
- Podemos hablar de que este tipo de arquitecturas ofrecen soluciones seguras desde el diseño, y podrían ser denominadas como seguridad como desarrollo. El diseño e implementación de las mismas "obligan" a pensar en la seguridad del aplicativo.
- Istio se puede combinar con aplicaciones de monitorización como Kiali, que ofrecen una visibilidad y monitorización de sucesos en online.
- La modificación y adaptación a cualquier capacidad de seguridad es prácticamente inmediata, se realiza mediante ficheros de configuración y se puede realizar directamente sobre los entornos sin necesidad de realizar paradas en los mismos.
- La aplicación de este tipo de soluciones únicamente supone una capa de seguridad más en la arquitectura que tengamos que implementar. Desde mi punto de vista, la implantación de medidas en tiempo de diseño sobre el desarrollo, no supone que no se deban implementar medidas de seguridad con elementos de electrónica de red, preventivos o de monitorización.
- Istio se puede implementar con cualquier solución cloud en la que se puedan instalar arquitecturas basadas en Kubernetes. Es decir, a la ya capacidad de Istio de añadir medidas de seguridad, se le puede añadir también las medidas propias de los propios cloud.

7.1. Problemas encontrados en la implementación

El TFM lo abordé con una visión de análisis de las diferentes soluciones que cada uno de los cloud de referencia ofrecen para aplicaciones basadas en API. En un primer momento no enfoqué el proyecto orientado a Istio hasta que analizando estas soluciones Cloud encontré esta plataforma y sus posibilidades.

En el apartado 1.5 Estado del arte, se puede ver una parte muy resumida de esta fase del proyecto. Fueron horas de investigación que no llegaron a buen término para el objetivo final, si bien han servido para tener un contexto, no sólo de posibles soluciones en el ámbito Cloud, sino de las posibilidades que luego he sido capaz de revisar con Istio para su implantación en un Cloud.

También me planteé poder hacer un desarrollo basado en contenedores y poderlo desplegar en Kubernetes. De hecho, llegué a implementar alguna solución de prueba en Google Cloud Platform. Nuevamente este tiempo de investigación no iba en la buena dirección puesto que necesitaba centrarme en el objetivo principal de análisis e implementación de capacidades de seguridad sobre cualquier aplicativo.

Me ha costado cerrar un objetivo claro hasta llegar a entender y comprender las capacidades de Istio y cómo poderlas implementar. Básicamente me he basado en la documentación propia de Istio, pero también han servido vídeos de compañías como Google e IBM que apuestan por esta tecnología. En definitiva, ha costado centrar el

análisis y la implementación de una solución que me ayudara a alcanzar los objetivos del proyecto.

7.2. Evaluación de objetivos alcanzados

En el apartado 1.2 Objetivos del trabajo. Se definen los objetivos del trabajo, si bien, tal y como he descrito en el apartado justamente anterior, he tenido diversos problemas, pero también fases de investigación que me han ayudado a completar y centrar no sólo el análisis, sino el objetivo del proyecto.

Tomando los objetivos definitivos del proyecto y en base a las diferentes fases que he tenido que realizar para finalizar el TFM, podríamos hablar de los siguientes puntos para definir la evaluación de objetivos:

- Análisis de clouds.
- Análisis de riesgos.
- Objetivos de implantación.

7.2.1. Análisis de clouds

Aunque no está en los objetivos finales del proyecto, todo el análisis de documentación propia de los diferentes clouds y sus soluciones, así como poder trabajar con versiones trial, me ha dado una visión global de las capacidades que éstas ofrecen y cómo la aplicación práctica de soluciones basadas en microservicios y soluciones basadas en contenedores y kubernetes, tienen puntos en común.

De hecho la aplicación práctica en los clouds no difiere mucho de la aplicación que se puede realizar en una máquina virtual con un host propio de un PC en local. Esto último, añadido a la no necesidad de tenerme que apoyar en versiones trial en el cloud, me animó a montar una máquina en mi PC local.

7.2.2. Análisis de riesgos

El planteamiento de implantación de medidas técnicas que aportaba Istio, debía tener un análisis de vulnerabilidades y un sentido real del porqué se plantean este tipo de soluciones.

Por este motivo creí conveniente definir un contexto de análisis de riesgos sobre el cual describir las soluciones y medidas técnicas que aporta Istio y en definitiva cómo puede mitigar estos riesgos.

7.2.3. Objetivos de implantación

Además de los riesgos que toda solución de ciberseguridad debe mitigar, he añadido un apartado que resume los diferentes conceptos clave de las arquitecturas analizadas y que ofrece un contexto de la solución.

Basándonos en este contexto, se han analizado algunas de las capacidades que ofrece Istio. No eran objeto del TFM aquellas capacidades que no estuvieran orientadas a la ciberseguridad. Nos hemos centrado en las capacidades que dotan a la arquitectura de comunicaciones seguras y un control de acceso y sobre éstas se ha hecho una implementación sobre una aplicación que la propia plataforma de Istio ofrece con objetivos de aprendizaje.

Por último, en cuanto a objetivos de implantación, no sólo se ha demostrado que Istio se puede configurar con estos objetivos de implantación de medidas de seguridad, sino que además se puede monitorizar y revisar estos cambios en tiempo real, lo cual hace de Istio una herramienta muy versátil para la realización de diseños orientados a la ciberseguridad.

7.3. Trabajo futuro

En cuanto a posibles trabajos que se puedan derivar de éste, en mi opinión hay dos líneas que se pueden desarrollar de forma independiente:

- Trabajo de análisis de capacidades de Istio.
- Trabajos que pudieran ser de interés en aplicación de arquitecturas con Kubernetes y apoyadas en Istio.

7.3.1. Trabajo de análisis de capacidades

Son muchas, el listado es muy largo y los conceptos y capacidades que pueden implementarse con Istio pueden ser de diferente índole. Istio los divide en los siguientes elementos conceptuales:

- Seguridad, que es en el que nos hemos centrado, pero hay funcionalidades que no hemos llegado a implementar como la aplicación de certificados autogenerados.
- Observalidad. Ya hemos visto que Istio se puede monitorizar, pero hay una capacidad que puede entenderse también como propia de ciberseguridad, ya que Istio, en el módulo de Data Plane, puede enviar información al Control Plane para que ésta sea monitorizada. Trabajos a futuro pueden analizar cómo ésta información puede ser ingestada en un SIEM y analizada desde allí.
- Gestión de tráfico. Este tipo de capacidades, son vistas y analizadas por Istio como un apartado diferenciado de la seguridad. En la implementación de este TFM se ha demostrado cómo haciendo uso de capacidades de seguridad también se llega a realizar un control de tráfico controlando los accesos a los microservicios. Sin embargo, un análisis más detallado, debería abordar cómo hacer que se aborde la gestión de tráfico entre diferentes versiones de software desplegado en la arquitectura Istio.

7.3.2. Otros trabajos de interés

Durante la fase de investigación, teniendo en cuenta algunas de las asignaturas cursadas en el Máster y dada la versatilidad de las arquitecturas orientadas a microservicios, se pueden analizar soluciones de integración con el mundo de arquitecturas distribuidas.

Aunque ya desde un principio lo descarté me resultó de gran interés el artículo publicado en la 2018 IEEE International Conference on Internet of Things[13], en la que se postulaba un entorno de creación de claves públicas basadas en tecnología blockchain.

Justamente, la integración de este tipo de arquitecturas con las capacidades de las arquitecturas de microservicios, creo que pueden ser un buen punto de inicio para análisis y trabajos futuros.

[13] *Distributed Public Key Infrastructure and PSK Exchange Based on Blockchain Technology* [en línea] [fecha de consulta: 31 de mayo de 2022]. Disponible en:

https://www.researchgate.net/publication/333585677_Distributed_Public_Key_Infrastructure_and_PSK_Exchange_Based_on_Blockchain_Technology

8. Glosario

Se identifican en este apartado algunas de las palabras y términos que se han mencionado a lo largo de la presente memoria. En la redacción de la misma se ha intentado explicar cada uno de los términos así como el razonamiento de su inclusión en el proyecto, así como su concepto.

Este apartado pretende ser una guía de consulta de estos términos:

- **Service Mesh**: Una malla de servicios o service mesh, como el proyecto open source Istio, se utiliza para controlar el intercambio de datos entre las distintas partes de una aplicación.
- **mTLS**: El TLS mutuo (mTLS) es un tipo de autenticación en el que las dos partes de una conexión se autentican mutuamente mediante el uso del protocolo TLS.
- **TLS**: Seguridad de la capa de transporte (en inglés: Transport Layer Security o TLS) y su antecesor Secure Sockets Layer (SSL; en español capa de puertos seguros) son protocolos criptográficos, que proporcionan comunicaciones seguras por una red, comúnmente Internet
- **https**: protocolo de transferencia de hiper texto seguro, y es la versión encriptada de HTTP. Se utiliza para una comunicación segura a través de Internet o de una red. El protocolo de comunicación se cifra mediante Transport Layer Security (TLS).
- **http**: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos.
- **API**: El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.
- **Servicios web**: Un servicio web (en inglés, web service o web services) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet.
- **Contenedores**: Los contenedores, en esencia, son procesos que corren dentro del sistema operativo los cuáles cumplen con algunas características que los hacen "especiales".
- **Kubernetes**: plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa.
- **Microservicios**: Los microservicios son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí.
- **Riesgo inherente o intrínseco**: El riesgo inherente es aquel que puede existir de manera intrínseca en toda actividad.
- **Riesgo residual**: El riesgo residual es el riesgo que permanece después de se han hecho todos los esfuerzos para identificar y eliminar el riesgo (es decir, sus controles de mitigación).
- **Istio**: Tal y como se define la propia Istio "*Istio addresses the challenges developers and operators face with a distributed or microservices architecture. Whether you're building from scratch or migrating existing applications to cloud native, Istio can help.*"
- **Ubuntu**: distribución Linux basada en Debian GNU/Linux, que incluye principalmente software libre y de código abierto.
- **Google Cloud Platform**: suite de infraestructuras y servicios que Google utiliza a nivel interno y, ahora, disponible para cualquier empresa, de tal forma que sea aplicable a multitud de procesos empresariales.

- **Amazon Web Services**: colección de servicios de computación en la nube pública (también llamados servicios web) que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com.
- **Azure**: Azure es una nube pública de pago por uso que te permite compilar, implementar y administrar rápidamente aplicaciones en una red global de datacenters (centros de datos) de Microsoft.
- **SIEM**: La información sobre seguridad y gestión de eventos o SIEM (Security Information and Event Management) es un sistema de seguridad que persigue proporcionar a las empresas una respuesta rápida y precisa para detectar y responder ante cualquier amenaza sobre sus sistemas informáticos.
- **IEEE**: El IEEE es la asociación profesional más grande del mundo, encargada de promover la innovación y excelencia tecnológica en beneficio de la humanidad.
- **Movimiento lateral**: El movimiento lateral es parte de una de las etapas que utiliza un hacker ético y cracker en un ataque dirigido, que típicamente son; la recolección de información y escaneo, acceso y escalamiento de privilegios, exfiltración, sostenimiento, asalto y ofuscación.
- **Man in the middle**: En criptografía, un ataque de intermediario1 (en inglés, man-in-the-middle attack, MitM o Janus) es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad
- **Extracción de información**: Se conoce como exfiltración de datos, o fuga de información, a la transferencia de información sensible entre la red de una organización víctima y una ubicación externa controlada por atacantes externos a la organización.
- **Suplantación de identidad**: La suplantación se basa en la habilidad de un hacker para hacerse pasar por otra persona o entidad. Algunos atacantes camuflan sus comunicaciones (como los correos electrónicos o las llamadas de teléfono) para que parezca que proceden de una persona u organización de confianza.
- **Dashboard**: Un dashboard es una herramienta de gestión de la información que monitoriza, analiza y muestra de manera visual los indicadores clave de desempeño (KPI), métricas y datos fundamentales para hacer un seguimiento del estado de una empresa, un departamento, una campaña o un proceso específico.

9. Bibliografía y fuentes consultadas

Se presentan en este apartado las diferentes fuentes bibliográficas consultadas, agrupadas según su tipología.

- Artículos
- Libros
- Vídeos
- Páginas web

9.1. Artículos

- Microservicios. URL https://www.redhat.com/es/topics/microservices
- Arquitectura de microservicios. URL https://www.redhat.com/en/resources/service-mesh-and-api-management-e-book?source=resourcelisting&sc_cid=7013a0000026TGbAAM&gclid=Cj0KCQjwnNyUBhCZARIsAI9AYlFmErmfBW8OaSeW9P2paR5dK4FHmLy04pruBF4qkBmvEpvEYRkw0FoaAhniEALw_wcB&gclsrc=aw.ds
- Istio Paradigma Digital. URL https://www.paradigmadigital.com/dev/jugando-con-istio-the-next-big-thing-en-microservicios-1-2/
- Arquitecturas distribuidas. URL https://www.researchgate.net/publication/333585677_Distributed_Public_Key_Infrastructure_and_PSK_Exchange_Based_on_Blockchain_Technology

9.2. Libros

- Introducing Istio Service Mesh for Microservices – O´Reilly
- Istio Explained_ Getting Started with Service Mesh – O´Reilly
- Kubernetes Patterns – O´Reilly

9.3. Vídeos

- Istio IBM. URL https://mediacenter.ibm.com/id/1_rr9zl8oo
- Linkeding Learning. URL https://es.linkedin.com/learning/istio-y-kubernetes-esencial/introduccion-a-istio-y-las-service-mesh
- What is a container. URL https://youtu.be/EnJ7qX9fkcU
- Kubernetes vs. Docker. URL https://youtu.be/2vMEQ5zs1ko
- Containers and VM. URL https://youtu.be/L1ie8negCjc
- Kubernetes Security Best Practices you need to know | THE Guide for securing your K8s cluster! URL https://youtu.be/oBf5lrmquYI
- SSL, TLS, HTTP, HTTPS Explained URL https://youtu.be/hExRDVZHhig
- Microservicios URL https://youtu.be/9R2hFwIPGnQ
- Kubernetes in 5 minutes. URL https://youtu.be/PH-2FfFD2PU

9.4. Páginas web

- What is an API?. URL https://github.com/OAI/Documentation/blob/main/introduction.md
- Especificación X.590v3 URL https://datatracker.ietf.org/doc/rfc5280/
- Especificación TLS 1.3.URL https://www.rfc-editor.org/rfc/rfc8446.txt
- Malla de servicios – Service Mesh. URL https://www.redhat.com/es/topics/microservices/what-is-a-service-mesh
- Istio. URL https://istio.io/latest/about/service-mesh/
- Istio IBM. URL https://www.ibm.com/cloud/istio?utm_content=SRCWW&p1=Search&p4=437000679887393

08&p5=e&gclid=Cj0KCQjwnNyUBhCZARIsAI9AYlFoIPo4B2etmsRPUKFFOTa_h6yTLCK8CJ7XSaB0vsaAXVOsJl4fytwaAhvcEALw_wcB&gclsrc=aw.ds
- Istio IBM. URL https://www.ibm.com/es-es/cloud/learn/istio
- Istio Google Cloud. URL https://cloud.google.com/learn/what-is-istio?hl=es
- About API Gateway. URL https://cloud.google.com/api-gateway/docs/about-api-gateway
- Acerca de API Management URL https://docs.microsoft.com/es-es/azure/api-management/api-management-key-concepts
- ¿Qué es Amazon API Gateway?. URL https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/welcome.html
- What is Istio URL https://istio.io/latest/about/service-mesh/
- Getting started URL https://istio.io/latest/docs/setup/getting-started/
- Identity and certificate management URL https://istio.io/latest/docs/concepts/security/#pki
- Istio Docs URL https://istio.io/latest/docs/
- Software Kiali URL https://github.com/istio/istio/tree/release-1.13/samples/addons