Implementación de un Service Mesh con k8s y Linkerd

Óscar Lucas Leo
I.E.S Gonzalo Nazareno

Índice

1. Introducción

La implementación de un Service Mesh con Kubernetes y Linkerd es una tarea fundamental en el despliegue de aplicaciones en entornos de contenedores. En este trabajo, exploraremos en detalle los conceptos, beneficios y desafíos asociados con la creación de un Service Mesh utilizando la plataforma de orquestación de contenedores Kubernetes y la herramienta de Service Mesh, Linkerd. Un Service Mesh es esencial para gestionar la comunicación entre microservicios en una arquitectura de aplicaciones distribuidas, proporcionando capacidades como el balanceo de carga, la seguridad, la monitorización y la gestión del tráfico de manera eficiente y escalable. A lo largo de este trabajo, examinaremos los pasos necesarios para llevar a cabo esta implementación, destacando las ventajas que ofrece Linkerd y cómo se integra de manera efectiva con Kubernetes. También se discutirán casos de uso comunes, prácticas recomendadas y posibles desafíos que pueden surgir en el proceso de implementación. En última instancia, este trabajo proporcionará una visión completa sobre cómo utilizar Kubernetes y Linkerd para crear un Service Mesh sólido y fiable en entornos de contenedores, permitiendo a las organizaciones gestionar sus aplicaciones de manera más eficiente y segura.

2. Objetivos que se quieren conseguir y se han conseguido

Objetivos que se quieren conseguir:

- ➔ Comprender los conceptos clave de Linkerd
- ➔ Implementar Linkerd en una aplicación web
- ➔ Mejorar la resiliencia de la aplicación
- ➔ Monitorizar la aplicación con Linkerd
- ➔ Asegurar la comunicación en la aplicación
- ➔ Generar informes y métricas
- ➔ Validar la mejora de la aplicación con Linkerd

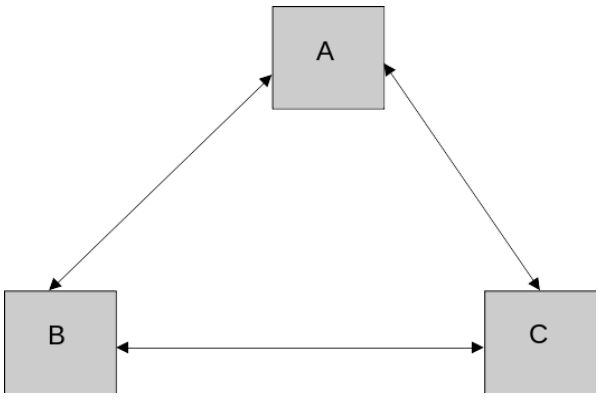
Objetivos que se han conseguido:

- ➔ Implementar Linkerd con éxito en la aplicación.
- ➔ Configurar las políticas de resiliencia, como retry y timeout, para mejorar la tolerancia a fallos en la aplicación.
- ➔ Configurar la monitorización de Linkerd y generar informes detallados sobre el rendimiento de los microservicios en la aplicación.
- ➔ Asegurar la comunicación entre los servicios de la aplicación mediante la encriptación y la autenticación proporcionadas por Linkerd.
- ➔ Generado informes y métricas que permiten evaluar el rendimiento y la eficiencia de la aplicación.
- ➔ Demostrar la mejora en la resiliencia y la monitorización de la aplicación después de implementar Linkerd.

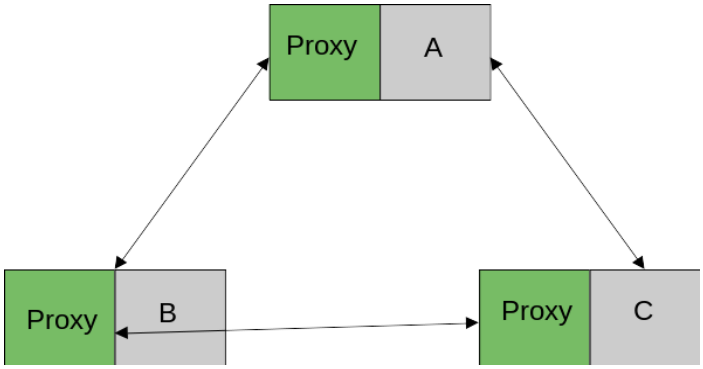
En resumen, he demostrado cómo Linkerd se utiliza para mejorar la resiliencia, la monitorización y la seguridad en una aplicación de microservicios.

3. Escenario necesario para la realización del proyecto

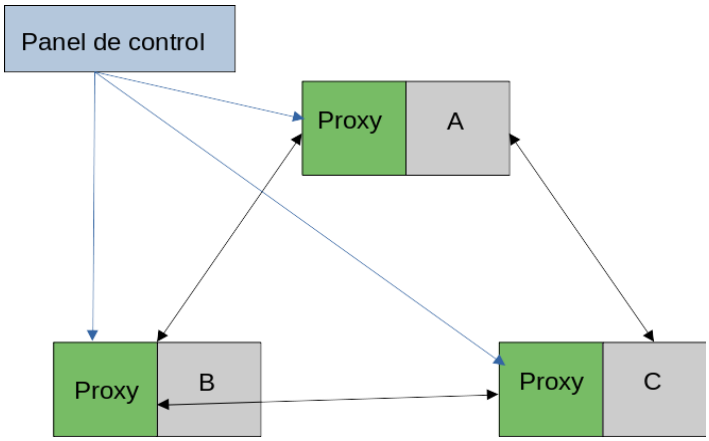
Comenzamos con un conjunto de microservicios agrupados en un clúster. Cada uno de estos microservicios se encuentra desplegado en su propio entorno (llamado "pod") y se comunican entre sí utilizando protocolos como HTTP o RPC.



Cuando se realiza la instalación de Linkerd, se procede a la inyección de un proxy en cada uno de los pods mencionados. De este modo, Linkerd asume el control completo de todas las comunicaciones que ingresan y salen de dichos pods, lo que implica que todas las comunicaciones pasan a través de estos proxies. Es de suma relevancia destacar que, a diferencia de otros proyectos de malla de servicios, el proxy de Linkerd ha sido meticulosamente diseñado específicamente para este propósito. Además, vale la pena señalar que está implementado en el lenguaje Rust, lo que nos proporciona una serie de garantías en términos de seguridad y rendimiento. Estas ventajas incluyen una reducción en el consumo de CPU y memoria, así como una menor latencia en las operaciones.



Por otra parte, el panel de control constituye un componente adicional que establece comunicación con los proxies, transmitiéndoles configuraciones y una identidad cifrada, mientras simultáneamente recopila datos telemétricos del conjunto compuesto por el plano de control y el plano de datos. Esta interacción conjunta entre el plano de control y el plano de datos es la que da forma a la malla de servicios en su totalidad.



4. Fundamentos teóricos y conceptos

4.1 Service Mesh

Un service mesh (malla de servicios) es una infraestructura de red que se utiliza para gestionar y controlar las comunicaciones entre los servicios en una arquitectura de aplicaciones distribuidas basada en microservicios. Su objetivo principal es proporcionar una capa de abstracción y control sobre las interacciones entre los servicios, lo que permite una gestión más eficiente, segura y observabilidad de las comunicaciones en un entorno distribuido.

Las características y funciones comunes de un service mesh incluyen:

1. Descubrimiento de servicios: Facilita la ubicación y el acceso a los servicios disponibles en la red, lo que permite a los servicios comunicarse sin necesidad de conocer las ubicaciones exactas de los demás.
2. Balanceo de carga: Distribuye equitativamente el tráfico entre múltiples instancias de un servicio, lo que mejora la escalabilidad y la redundancia.
3. Monitorización y trazabilidad: Recopila métricas detalladas y traza las solicitudes a través de la red, lo que permite una mayor visibilidad y diagnóstico de problemas.
4. Seguridad: Proporciona características de seguridad, como la autenticación, la autorización y el cifrado de datos, para proteger las comunicaciones entre servicios.
5. Gestión del tráfico: Permite implementar reglas de enrutamiento para controlar cómo se dirige el tráfico a los servicios, lo que puede ser útil para pruebas, despliegues gradualmente, etc.
6. Alta disponibilidad y tolerancia a fallos: Ayuda a garantizar que las aplicaciones sigan siendo operativas incluso en situaciones de fallos parciales.
7. Retroalimentación y ajuste dinámico: Puede ajustar automáticamente la configuración de red en función de las condiciones en tiempo real, como la carga del servicio y la disponibilidad de instancias.

Algunos ejemplos populares de service mesh en el mundo de la informática en la nube y la arquitectura de microservicios incluyen Istio, Linkerd y Envoy. Estas herramientas se utilizan comúnmente en entornos de contenedores y orquestación de contenedores, como Kubernetes, para proporcionar una capa de servicios de red que mejora la comunicación y la administración de aplicaciones en microservicios.

4.2 Microservicios

La arquitectura de microservicios es un enfoque de diseño de software en el que una aplicación monolítica se descompone en componentes más pequeños, autónomos e interconectados. Cada uno de estos componentes, conocidos como microservicios, es responsable de una funcionalidad específica de la aplicación. Los microservicios se ejecutan de manera independiente, lo que significa que pueden ser desarrollados, implementados y escalados de forma independiente.

Descomposición de las aplicaciones:

En la arquitectura de microservicios, una aplicación se divide en múltiples microservicios. Cada microservicio se enfoca en una tarea o función específica de la aplicación. Esto permite que los equipos de desarrollo trabajen de manera más eficiente en componentes más pequeños y especializados. Cada microservicio puede ser desarrollado utilizando diferentes tecnologías o lenguajes de programación, lo que facilita la elección de la tecnología más adecuada para cada tarea.

Beneficios:

- Escalabilidad: Los microservicios permiten escalar componentes individuales en función de la demanda, lo que mejora la eficiencia y el rendimiento de la aplicación.
- Mantenimiento simplificado: Los microservicios son más fáciles de mantener y actualizar, ya que los cambios en un microservicio no afectan necesariamente a otros.
- Desarrollo ágil: Los equipos pueden trabajar de manera independiente en microservicios, lo que acelera el desarrollo y la implementación de nuevas características.

- Mejora la resiliencia: Los fallos en un microservicio no afectan a toda la aplicación, lo que aumenta la tolerancia a fallos y la resiliencia.
- Adaptabilidad tecnológica: Cada microservicio puede utilizar la tecnología más adecuada para su tarea, lo que facilita la adopción de nuevas tecnologías.

Desafíos:

- Complejidad en la gestión: La arquitectura de microservicios introduce una mayor complejidad en la gestión de múltiples componentes y comunicaciones entre ellos.
- Requisitos de infraestructura: Es necesario contar con una infraestructura robusta para implementar y gestionar los microservicios, lo que puede requerir inversiones en herramientas y recursos.
- Coordinación de servicios: La coordinación entre microservicios puede ser complicada, y es necesario establecer mecanismos efectivos para garantizar la coherencia de la aplicación.
- Seguridad y monitoreo: La seguridad y el monitoreo se vuelven más complejos en un entorno de microservicios, lo que requiere una atención especial.

En resumen, la arquitectura de microservicios es un enfoque de diseño que descompone las aplicaciones en componentes más pequeños y autónomos. Ofrece ventajas como escalabilidad, mantenimiento simplificado y desarrollo ágil, pero también plantea desafíos relacionados con la gestión, la coordinación y la seguridad. Las organizaciones deben evaluar cuidadosamente si esta arquitectura es adecuada para sus necesidades antes de adoptarla.

4.3 Kubernetes (k8s)

Kubernetes, a menudo abreviado como K8s, es un sistema de código abierto diseñado para automatizar, escalar y administrar aplicaciones en contenedores. Fue desarrollado originalmente por Google y luego donado a la Cloud Native Computing Foundation (CNCF). Kubernetes se ha convertido en una de las herramientas más populares y ampliamente utilizadas para orquestar contenedores en entornos de contenedorización, como Docker.

Algunos de los conceptos clave de Kubernetes incluyen:

1. Contenedores: Kubernetes se utiliza comúnmente para administrar contenedores, que son unidades ligeras y portables que encapsulan aplicaciones y sus dependencias, lo que facilita su implementación y administración.
2. Orquestación: Kubernetes automatiza tareas como el despliegue, escalado, actualización y recuperación de aplicaciones en contenedores. Esto permite una administración eficiente y sin problemas de aplicaciones distribuidas.
3. Escalabilidad: Kubernetes permite escalar aplicaciones de manera flexible según la demanda, ya sea horizontalmente (aumentando el número de instancias) o verticalmente (aumentando los recursos de una instancia).
4. Alta disponibilidad: Kubernetes se esfuerza por garantizar que las aplicaciones sean altamente disponibles mediante la distribución de contenedores en múltiples nodos (máquinas) y la recuperación automática de fallos.
5. Configuración declarativa: Los usuarios describen el estado deseado de sus aplicaciones y clústeres en archivos de configuración, y Kubernetes se encarga de llevar el sistema al estado deseado, lo que se conoce como "infraestructura como código".
6. Gestión de recursos: Kubernetes gestiona los recursos del clúster, como CPU, memoria y almacenamiento, para garantizar un uso eficiente de los mismos y prevenir conflictos entre aplicaciones.
7. Networking: Kubernetes proporciona una capa de red virtual que permite la comunicación entre contenedores y servicios de aplicaciones, así como la exposición de servicios a través de direcciones IP externas.

En resumen, Kubernetes es una plataforma poderosa para administrar aplicaciones en contenedores en entornos de infraestructura distribuida y se ha convertido en una herramienta fundamental en el mundo de la contenerización y la orquestación de aplicaciones en la nube.

4.4 Proxy en un Service Mesh

En un Service Mesh, se utilizan proxies tanto en el lado del cliente como en el lado del servidor para facilitar la gestión de tráfico, la seguridad y la monitorización en las comunicaciones entre los servicios. Estos proxies son componentes de software que actúan como intermediarios entre los servicios que se comunican. Cada servicio en el Service Mesh tiene un proxy asociado que maneja el tráfico de entrada y salida hacia y desde ese servicio.

- Proxy de lado cliente: El proxy de lado cliente se encuentra en el servicio que inicia una solicitud. Cuando un servicio A desea comunicarse con un servicio B, el proxy de lado cliente de A es el responsable de enrutar la solicitud hacia el servicio B a través del Service Mesh. Además, este proxy puede aplicar políticas de seguridad, como autenticación y autorización, y realizar funciones de monitorización y registro.
- Proxy de lado servidor: El proxy de lado servidor se encuentra en el servicio que recibe una solicitud. Cuando un servicio B recibe una solicitud de servicio A, el proxy de lado servidor de B es el encargado de procesar la solicitud, aplicar políticas de seguridad y luego enrutar la respuesta a través del Service Mesh hacia el servicio A. También se encarga de la monitorización y el registro de eventos relacionados con la solicitud y la respuesta.

Inyección de Proxy:

La inyección de proxy es un proceso mediante el cual los proxies de lado cliente y servidor se insertan automáticamente en las comunicaciones entre los servicios de un Service Mesh, sin que los desarrolladores tengan que realizar cambios significativos en el código de sus aplicaciones. Esto se logra generalmente mediante el uso de inyección automática de proxy a través de middleware o agentes específicos del Service Mesh.

La inyección de proxy ofrece varios beneficios:

- Transparencia: Los desarrolladores pueden seguir desarrollando sus servicios sin preocuparse por detalles de red y seguridad, ya que la inyección de proxy se encarga de estos aspectos de manera automática.
- Gestión de tráfico: Los proxies permiten la aplicación de reglas de gestión de tráfico, como enrutamiento, balanceo de carga y circuit breakers, de forma centralizada y dinámica.
- Seguridad: Los proxies pueden implementar políticas de seguridad, como la autenticación y la autorización, a nivel de red, garantizando que las comunicaciones sean seguras.
- Monitorización y registro: Los proxies generan métricas y registros que proporcionan información valiosa sobre el rendimiento de la aplicación y permiten la detección de problemas y la resolución de incidentes.

En resumen, en un Service Mesh, los proxies de lado cliente y servidor son componentes clave que se utilizan para gestionar el tráfico, mejorar la seguridad y habilitar la monitorización en las comunicaciones entre servicios. Estos proxies se inyectan automáticamente en las comunicaciones, lo que simplifica la administración y el control de las aplicaciones distribuidas.

4.5 Linkerd

Linkerd es una plataforma de servicio de malla (service mesh) de código abierto diseñada para facilitar la comunicación, supervisión, seguridad y gestión de servicios en aplicaciones distribuidas basadas en microservicios. Una malla de servicios es una infraestructura que se sitúa entre los servicios que componen una aplicación, lo que permite una gestión más granular y una mayor visibilidad de las interacciones entre ellos.

Algunas de las características y funciones clave de Linkerd incluyen:

1. Observabilidad: Linkerd proporciona métricas detalladas y seguimiento (tracing) de solicitudes, lo que permite a los equipos de operaciones y desarrollo comprender mejor el rendimiento y el comportamiento de sus aplicaciones distribuidas.
2. Seguridad: Linkerd ofrece características de seguridad, como la autenticación, autorización y el cifrado de datos, que ayudan a proteger las comunicaciones entre los servicios en la malla.
3. Alta disponibilidad: Linkerd está diseñado para ser altamente disponible y tolerante a fallos, lo que garantiza que las aplicaciones sigan funcionando incluso en situaciones de fallos parciales.
4. Enrutamiento y equilibrio de carga: Linkerd permite el enrutamiento y el equilibrio de carga de tráfico entre los servicios, lo que ayuda a distribuir las solicitudes de manera equitativa y a dirigirlas a las instancias de servicio adecuadas.
5. Integración: Linkerd se integra con diversas tecnologías y sistemas, lo que facilita su implementación en aplicaciones existentes y su uso en entornos de Kubernetes y contenedores.

Linkerd es conocido por ser fácil de configurar y utilizar, y se esfuerza por ofrecer una experiencia sin problemas para los desarrolladores y administradores de sistemas. Es uno de los proyectos de código abierto de la Cloud Native Computing Foundation (CNCF) y se utiliza en combinación con tecnologías como Kubernetes para administrar aplicaciones basadas en microservicios de manera eficiente y segura.

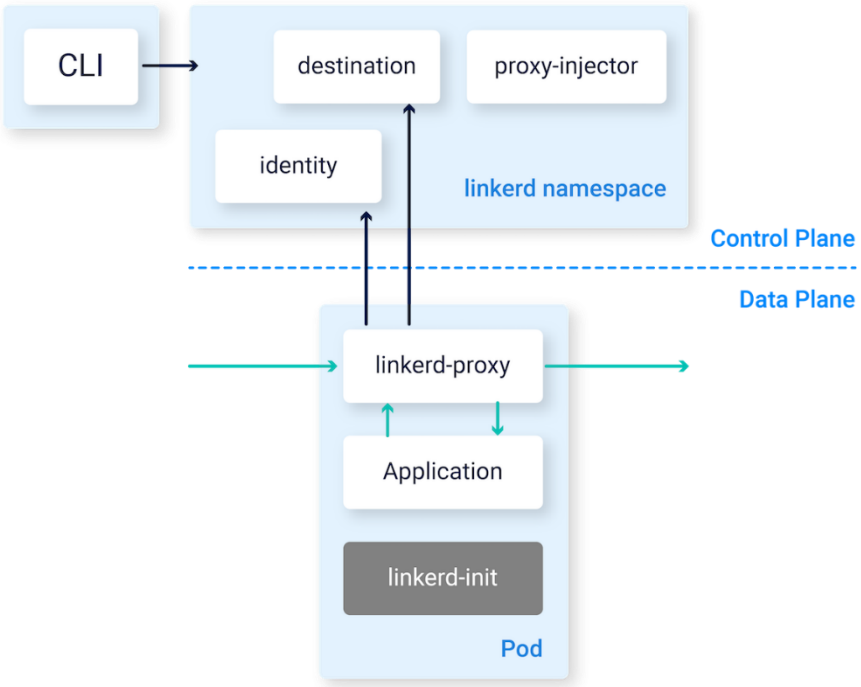
4.6 Arquitectura de Linkerd

A un nivel elevado, Linkerd se estructura en un plano de control y un plano de datos.

El plano de control se compone de un conjunto de servicios que confieren control sobre la totalidad de Linkerd.

El plano de datos está constituido por microproxies transparentes que se ejecutan de manera adyacente a cada instancia de servicio, actuando como contenedores secundarios en los pods. Estos servidores proxy gestionan de manera automática todo el tráfico TCP que fluye hacia y desde el servicio, estableciendo comunicación con el plano de control para su configuración.

Además, Linkerd dispone de una interfaz de línea de comandos (CLI) que puede utilizarse para interactuar con los planos de control y datos.



4.6.1 CLI

La interfaz de línea de comandos (CLI) de Linkerd se ejecuta típicamente fuera del clúster, por ejemplo, en su máquina local, y se emplea para la interacción con Linkerd.

4.6.2 Plano de Control

El plano de control de Linkerd consiste en un conjunto de servicios que operan en un espacio de nombres específico de Kubernetes (por defecto, "linkerd"). Este plano de control comprende diversos componentes, detallados a continuación.

El Servicio de Destino

Los servidores proxy del plano de datos emplean el servicio de destino para determinar diversos aspectos de su comportamiento. Este servicio se utiliza para obtener información de descubrimiento de servicios, es decir, para determinar dónde enviar una solicitud específica y verificar la identidad TLS esperada en el extremo receptor. Además, se utiliza para obtener información sobre políticas que define qué tipos de solicitudes están permitidas. También se recupera información del perfil de servicio, la cual se utiliza para informar métricas, gestionar reintentos y establecer tiempos de espera específicos para cada ruta, entre otras funciones.

El Servicio de Identidad

El servicio de identidad de Linkerd cumple la función de una autoridad de certificación TLS que recibe solicitudes de firma de certificados (CSR, por sus siglas en inglés) provenientes de los servidores proxy. A su vez, emite certificados firmados en el momento de la inicialización del proxy. Estos certificados se emplean para las conexiones de proxy a proxy, implementando así el mecanismo de Transport Layer Security mutuo (mTLS, por sus siglas en inglés).

El Inyector Proxy

El inyector de proxy de Linkerd es un controlador de admisión de Kubernetes que recibe una solicitud de webhook cada vez que se crea un pod. Este inyector examina los recursos en busca de una anotación específica de Linkerd (linkerd.io/inject: enabled). Cuando dicha anotación está presente, el inyector modifica la especificación del pod y añade los contenedores proxy-init y linkerd-proxy al pod, junto con la configuración de inicio correspondiente.

4.6.3 Plano de Datos

El plano de datos de Linkerd está compuesto por microproxies extremadamente livianos que se despliegan como contenedores adicionales (sidecar) dentro de los módulos de aplicaciones. Estos proxies interceptan de manera transparente las conexiones TCP entrantes y salientes de cada pod, aprovechando las reglas de iptables establecidas por linkerd-init (o, como alternativa, por el complemento CNI de Linkerd).

Apoderado

Linkerd2-proxy es un microproxy ultraligero y transparente desarrollado en Rust. Este proxy está específicamente diseñado para su implementación en casos de uso de mallas de servicios, no siendo concebido como un proxy de uso general.

Las características distintivas del proxy incluyen:

- Funcionalidad transparente y sin necesidad de configuración para protocolos HTTP, HTTP/2 y TCP arbitrarios.
- Exportación automática de métricas de Prometheus para el tráfico HTTP y TCP.
- Soporte transparente y sin configuración para conexiones WebSocket.
- Equilibrio de carga automático a nivel de la capa 7, consciente de la latencia.
- Equilibrio de carga automático a nivel de la capa 4 para el tráfico no relacionado con HTTP.
- Configuración automática de TLS.
- Una API de diagnóstico bajo demanda.
- Y diversas funcionalidades adicionales.

El proxy facilita el descubrimiento de servicios a través de DNS y la API gRPC de destino.

Para obtener información más detallada sobre estos microproxies, se puede consultar el siguiente enlace:

Por qué Linkerd no usa Envoy Bajo el capó del proxy Rust de última generación de Linkerd, Linkerd2-proxy

Contenedor de Inicio de Linkerd

El contenedor linkerd-init se incorpora a cada pod incluido en la malla de servicios como un contenedor de inicio de Kubernetes. Este contenedor se ejecuta antes que los demás contenedores del pod. Utiliza iptables para dirigir el tráfico TCP entrante y saliente del pod hacia el proxy. El modo de ejecución del contenedor de inicio de Linkerd puede variar, determinando así qué variante de iptables se utiliza.

4.7 Balanceo de Carga y Enrutamiento en un Service Mesh

Un Service Mesh, como Linkerd, desempeña un papel esencial en la distribución eficiente del tráfico entre los servicios de una aplicación distribuida. El balanceo de carga y el enrutamiento inteligente son dos funcionalidades clave que permiten lograr esto de manera efectiva.

- Balanceo de Carga: El balanceo de carga se refiere a la distribución equitativa del tráfico entre múltiples instancias de un mismo servicio para garantizar que no se sobrecarguen y que el tráfico se distribuya de manera uniforme. Linkerd, como un componente del Service Mesh, es capaz de llevar a cabo el balanceo de carga de manera automática y dinámica. Esto significa que, cuando un servicio cliente realiza una solicitud, Linkerd puede decidir a qué instancia del servicio destino debe dirigir esa solicitud, basándose en diversos factores como la capacidad de procesamiento, la latencia o el estado de salud de las instancias del servicio.
- Enrutamiento Inteligente: El enrutamiento inteligente implica tomar decisiones sobre cómo dirigir el tráfico de manera efectiva y dinámica en función de diversas consideraciones, como el contexto de la solicitud, las políticas de seguridad y los objetivos de rendimiento. Linkerd permite configurar reglas de enrutamiento flexibles y personalizadas. Por ejemplo, puede dirigir solicitudes a versiones específicas de un servicio, aplicar políticas de seguridad (como autenticación y autorización), o incluso redirigir el tráfico en función de condiciones como el porcentaje de error o la latencia. Esto es especialmente valioso para implementar estrategias de despliegue gradual y pruebas A/B.

Beneficios:

- Distribución de carga equitativa: El balanceo de carga garantiza que las instancias de un servicio reciban una cantidad equitativa de tráfico, lo que mejora el rendimiento y evita sobrecargas.
- Tolerancia a fallos: Distribuir el tráfico de manera equitativa permite que la aplicación sea más resistente a fallos, ya que si una instancia de servicio falla, las solicitudes pueden ser redirigidas automáticamente a instancias saludables.
- Flexibilidad: El enrutamiento inteligente permite personalizar las reglas de enrutamiento para satisfacer las necesidades específicas de la aplicación, lo que facilita implementar estrategias de despliegue y prueba más avanzadas.
- Mejora del rendimiento: El enrutamiento inteligente también se puede utilizar para dirigir el tráfico a las instancias que ofrezcan el mejor rendimiento en función de las métricas de latencia y otros factores.

En resumen, un Service Mesh como Linkerd facilita el balanceo de carga y el enrutamiento inteligente del tráfico entre los servicios de una aplicación distribuida. Esto mejora la distribución de carga, aumenta la tolerancia a fallos y brinda mayor flexibilidad y rendimiento a la aplicación, lo que es esencial en entornos de microservicios y aplicaciones distribuidas.

4.8 Monitorización y Métricas

La monitorización y las métricas son aspectos críticos en cualquier entorno de aplicaciones distribuidas, y un Service Mesh, como Linkerd, juega un papel fundamental en proporcionar capacidades de observabilidad para rastrear el rendimiento de los servicios. A continuación, te explicaré la importancia de la monitorización y las métricas en un Service Mesh y cómo Linkerd contribuye a esta área:

Importancia de la Monitorización y las Métricas en un Service Mesh:

La monitorización y las métricas son esenciales para comprender y optimizar el funcionamiento de una aplicación distribuida. Algunas de las razones clave para su importancia incluyen:

1. Visibilidad: La monitorización y las métricas proporcionan visibilidad en tiempo real sobre cómo los servicios se están comportando, lo que permite a los equipos de operaciones y desarrollo identificar problemas y tomar decisiones informadas.
2. Diagnóstico de problemas: Las métricas permiten la detección temprana de problemas y facilitan la identificación de las causas subyacentes de los errores o las degradaciones del rendimiento.
3. Optimización del rendimiento: Las métricas proporcionan datos sobre el rendimiento de los servicios, lo que permite a los equipos ajustar y optimizar sus aplicaciones para satisfacer mejor las demandas de los usuarios.

Cómo Linkerd Proporciona Capacidades de Observabilidad:

Linkerd, como un Service Mesh, se centra en mejorar la observabilidad de las aplicaciones distribuidas. Aquí hay algunas formas en las que Linkerd proporciona capacidades de monitorización y métricas:

1. Recopilación de Datos: Linkerd recopila una variedad de datos de tráfico y rendimiento, incluyendo métricas de latencia, tasas de error y rendimiento de servicios. Estos datos se utilizan para ofrecer información sobre el rendimiento de los servicios y la salud de la aplicación.
2. Tablero de Control y Visualizaciones: Linkerd proporciona un tablero de control y visualizaciones que permiten a los equipos de operaciones y desarrollo ver los datos de monitorización de manera gráfica y comprensible. Esto facilita la identificación de tendencias, patrones y problemas.
3. Tracing (seguimiento): Linkerd permite el seguimiento de solicitudes a través de múltiples servicios, lo que facilita la identificación de cuellos de botella y problemas en la comunicación entre servicios.

4. Alertas: Linkerd permite configurar alertas basadas en umbrales y reglas personalizadas. Esto ayuda a los equipos a ser notificados cuando se detectan problemas de rendimiento o errores en la aplicación.
5. Compatibilidad con Herramientas de Observabilidad Externas: Linkerd se integra con herramientas de observabilidad de terceros, como Prometheus, Grafana y Jaeger, lo que permite a los equipos aprovechar herramientas ya familiares para una mayor visibilidad.

En resumen, Linkerd proporciona capacidades de observabilidad críticas para rastrear el rendimiento de los servicios en un entorno de aplicaciones distribuidas. La recopilación de datos, la visualización, el seguimiento y las alertas ayudan a los equipos a monitorear y solucionar problemas, garantizando un rendimiento óptimo y una mejor experiencia para los usuarios. La observabilidad que ofrece Linkerd es esencial para operar aplicaciones basadas en microservicios y para garantizar la calidad y confiabilidad del sistema.

4.9 Seguridad y Autenticación

La seguridad es un aspecto crítico en las aplicaciones distribuidas, y un Service Mesh como Linkerd desempeña un papel importante en la protección de las comunicaciones entre los servicios. A continuación, te explicaré cómo Linkerd contribuye a la seguridad de las comunicaciones entre servicios, incluyendo la autenticación y la encriptación de datos sensibles:

Seguridad en las Comunicaciones entre Servicios con Linkerd:

1. Encriptación de Datos: Linkerd proporciona una capa de encriptación de comunicaciones entre servicios. Utiliza el protocolo TLS (Transport Layer Security) para cifrar los datos en tránsito, garantizando que las comunicaciones sean seguras y protegiendo contra posibles ataques de escucha o interceptación de datos.

2. Autenticación Mutua: Linkerd admite la autenticación mutua de servicios a través de certificados TLS. Esto significa que tanto los servicios cliente como los servicios servidor se autentican entre sí mediante certificados digitales. Esto proporciona un alto nivel de confianza en la identidad de los servicios y ayuda a prevenir la suplantación de identidad.
3. Control de Acceso y Autorización: Linkerd también permite la implementación de políticas de control de acceso y autorización. Esto asegura que solo los servicios autorizados puedan comunicarse entre sí y que se cumplan las políticas de seguridad establecidas. Se pueden definir reglas de autorización para determinar quién puede acceder a qué recursos.
4. Detección de Ataques: Linkerd es capaz de detectar y bloquear ataques como el ataque por repetición (replay attacks) y proporciona medidas para prevenir ataques de denegación de servicio (DoS) y otros intentos maliciosos.
5. Gestión de Identidades y Credenciales: Linkerd puede integrarse con sistemas de gestión de identidades y credenciales, como OAuth o OpenID Connect, para una autenticación más avanzada y segura. Esto es particularmente útil en entornos donde se requiere una gestión de identidades más robusta.
6. Auditoría y Registro: Linkerd registra las actividades de comunicación entre servicios, lo que facilita la auditoría y la revisión de seguridad. Esto es fundamental para el cumplimiento normativo y para la identificación de posibles amenazas.

En resumen, Linkerd contribuye a la seguridad de las comunicaciones entre servicios en una aplicación distribuida al proporcionar encriptación de datos, autenticación mutua, control de acceso, detección de ataques y opciones de integración con sistemas de gestión de identidades. Esto asegura que las comunicaciones sean seguras y que se cumplan las políticas de seguridad, lo que es fundamental en entornos de microservicios y aplicaciones distribuidas donde la protección de datos y la confiabilidad son prioridades clave.

4.10 Resiliencia y Tolerancia a Fallos

La resiliencia y la tolerancia a fallos son aspectos críticos en la construcción de aplicaciones distribuidas, y un Service Mesh como Linkerd desempeña un papel fundamental en el aumento de la resiliencia de una aplicación. A continuación, te explicaré cómo Linkerd contribuye a la resiliencia, incluyendo el manejo de retries y timeouts:

Resiliencia y Tolerancia a Fallos en Linkerd:

1. Circuit Breakers: Linkerd implementa circuit breakers, que son mecanismos diseñados para prevenir que una sobrecarga o un fallo en un servicio afecte negativamente a otros servicios. Cuando un servicio experimenta un alto índice de errores o latencia, el circuit breaker puede interrumpir las solicitudes a ese servicio, evitando así que se propague el fallo.
2. Retries Automáticos: Linkerd puede configurarse para realizar automáticamente retries de solicitudes cuando se produce un fallo. Esto significa que si un servicio recibe un error temporal, como un tiempo de espera, Linkerd puede intentar nuevamente la solicitud en lugar de generar un error al cliente. Esto mejora la tolerancia a fallos y puede ayudar a suavizar situaciones transitorias de alta carga o problemas de red.
3. Timeouts Configurables: Linkerd permite configurar timeouts para las solicitudes entre servicios. Esto es esencial para evitar que una solicitud quede en espera indefinidamente, lo que podría afectar el rendimiento general de la aplicación. Los timeouts garantizan que las solicitudes se manejen de manera adecuada, incluso si un servicio destino está experimentando problemas.
4. Medición y Monitorización: Linkerd recopila métricas y datos de tráfico, lo que permite a los equipos de operaciones y desarrollo supervisar el rendimiento de la aplicación y detectar problemas relacionados con la resiliencia. Estos datos son valiosos para la detección temprana de problemas y la toma de decisiones informadas.

5. Backpressure: Linkerd permite la implementación de mecanismos de backpressure (retroalimentación) para evitar que un servicio cliente sature a un servicio servidor con demasiadas solicitudes. Esto garantiza que los servicios puedan administrar la carga de trabajo de manera efectiva y sin verse abrumados por solicitudes excesivas.
6. Distribución de Carga y Balanceo de Carga: Linkerd también contribuye a la resiliencia al permitir el balanceo de carga y la distribución de tráfico equitativa, lo que evita la sobrecarga de instancias de servicio individuales y contribuye a la tolerancia a fallos.

En resumen, Linkerd contribuye a aumentar la resiliencia de una aplicación distribuida al proporcionar mecanismos como circuit breakers, retries automáticos, timeouts configurables, monitorización y balanceo de carga. Estos componentes ayudan a la aplicación a mantener su funcionamiento en condiciones adversas o de fallo, garantizando una experiencia más robusta y confiable para los usuarios.

5. Implementación de la malla de servicios

La implementación de una malla de servicios, también conocida como Service Mesh, es un proceso crítico para gestionar la comunicación entre los microservicios en una arquitectura de aplicaciones distribuidas. En este apartado, se detallarán los pasos clave para llevar a cabo la implementación de una malla de servicios utilizando Kubernetes y Linkerd.

5.1 Instalación de Kubernetes

Instalar Docker:

- Instala Docker, que es el contenedor que utiliza Kubernetes para ejecutar aplicaciones en contenedores.

```
sudo apt-get update
sudo apt-get install docker.io
```

Agregar el Repositorio de Kubernetes y Claves GPG:

- Agrega la clave GPG de Kubernetes y el repositorio de Kubernetes.

```
sudo apt-get update && sudo apt-get install -y apt transport-https
```

```
curl -s https://packages.cloud.google.com/apt/doc/apt key.gpg | sudo apt-key add - echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
sudo apt-get update
```

Instalar kubeadm, kubelet y kubectl:

- Instala las herramientas kubeadm, kubelet y kubectl.

```
sudo apt-get install -y kubeadm kubelet kubectl
```

Configurar kubectl para Usar el Clúster:

- Configura kubectl para que apunte al clúster creado.

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

5.2 Configuración del clúster (minikube)

Para la implementación de mi clúster de Kubernetes, he tomado la decisión de utilizar Minikube debido a su versatilidad y simplicidad. Minikube ofrece la posibilidad de crear y gestionar clústeres de Kubernetes en entornos locales, proporcionando una solución ideal para desarrolladores que buscan una configuración fácil y eficiente.

Requisitos Previos

Antes de comenzar, asegúrate de tener instaladas las siguientes herramientas:

1. Minikube: Instala Minikube según las instrucciones de la página oficial.
2. kubectl: La herramienta de línea de comandos kubectl se utiliza para interactuar con clústeres de Kubernetes. Puedes obtenerlo aquí.

Creación de un Clúster Minikube

Paso 1: Iniciamos minikube con el siguiente comando:

```
minikube start
```

Esto creará un clúster de Kubernetes local y configurará automáticamente kubectl para apuntar a él.

Paso 2: Verificaremos el estado del clúster con el siguiente comando:

```
minikube status
```

Asegúrate de que el clúster esté en estado Running.

Paso 3: Detener y eliminar el Clúster.(opcional)

Para detener minikube ejecutaremos el siguiente comando:

```
minikube stop
```

Para eliminar el clúster ejecutaremos el siguiente comando:

```
minikube delete
```

Esto eliminará completamente el clúster de Minikube de tu máquina.

5.3 Instalación de Linkerd

Descargar e instalar Linkerd:

Si es la primera vez que ejecutas Linkerd, necesitarás descargar la interfaz de línea de comandos de Linkerd (Linkerd CLI) en tu máquina local. La CLI te permitirá interactuar con la implementación de Linkerd.

Para llevar a cabo la instalación de la CLI de forma manual, ejecuta el siguiente comando:

```
curl --proto '=https' --tlsv1.2 -sSfL
https://run.linkerd.io/install | sh
```

Asegúrate de seguir detenidamente las instrucciones proporcionadas para agregar la CLI a tu ruta. Puedes hacer esto ejecutando el siguiente comando:

```
export PATH=$HOME/.linkerd2/bin:$PATH
```

Esta acción garantizará que la CLI de Linkerd esté disponible en tu ruta y listo para su uso.

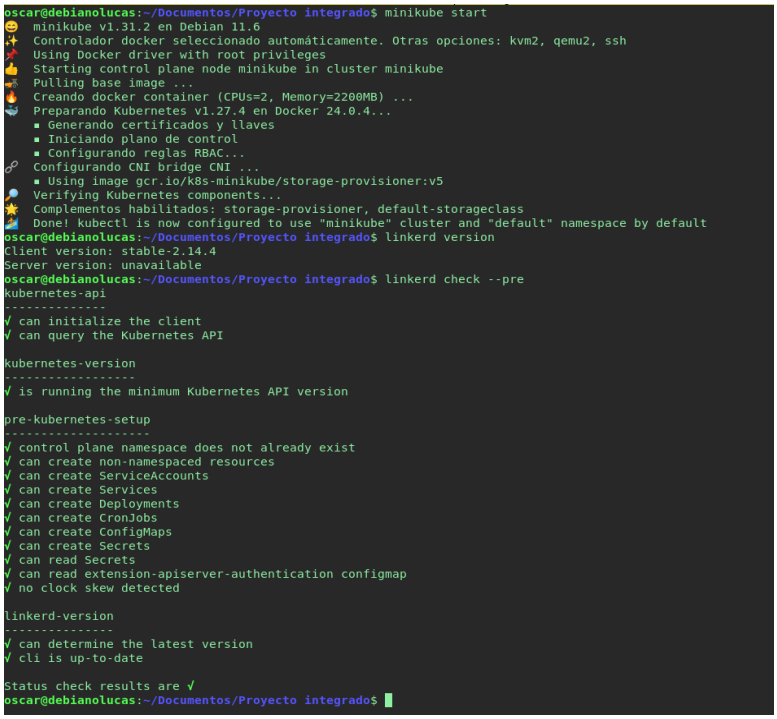
Verificar la versión de Linkerd:

```
linkerd version
```

Este comando muestra la versión de Linkerd que acabas de instalar. Asegúrate de que la instalación fue exitosa y de que la versión sea la esperada.

Verificar la configuración previa a la instalación:

```
linkerd check -pre
```



Este paso realiza algunas verificaciones previas a la instalación para asegurarse de que tu entorno esté configurado correctamente para la instalación de Linkerd.

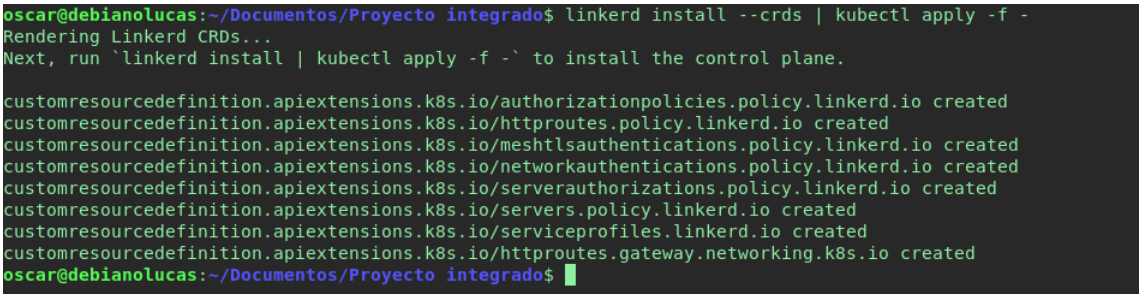
Instalar los Custom Resource Definitions (CRDs):

Nota:

En el contexto de Kubernetes, las Definiciones de Recursos Personalizadas (CRDs, por sus siglas en inglés) son una característica que posibilita a los usuarios la creación y utilización de recursos específicos personalizados en su entorno de Kubernetes.

Estas CRDs ofrecen la posibilidad de ampliar las capacidades de Kubernetes más allá de los recursos predeterminados que la plataforma proporciona, como Deployments, Services, Pods, entre otros.

```
linkerd install --crds | kubectl apply -f -
```



Este comando instala los Custom Resource Definitions necesarios para Linkerd. Los CRDs son extensiones de Kubernetes que permiten definir recursos personalizados.

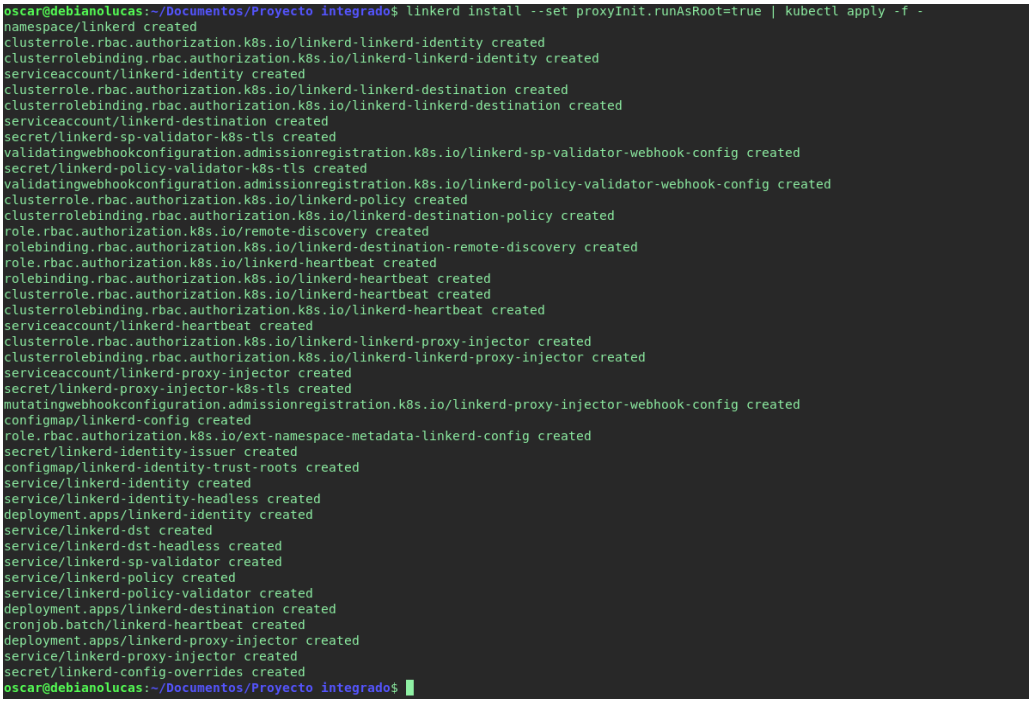
Instalar Linkerd:

Nota:

El Plano de Control, también conocido como Control Plane, se compone de una serie de servicios que operan dentro de un espacio de nombres particular.

Estos servicios desempeñan diversas funciones, como recopilar datos telemétricos, suministrar una interfaz de programación de aplicaciones (API) orientada al usuario, ofrecer datos de control a los proxies del plano de datos, entre otras tareas. En conjunto, estos servicios guían el comportamiento del plano de datos.

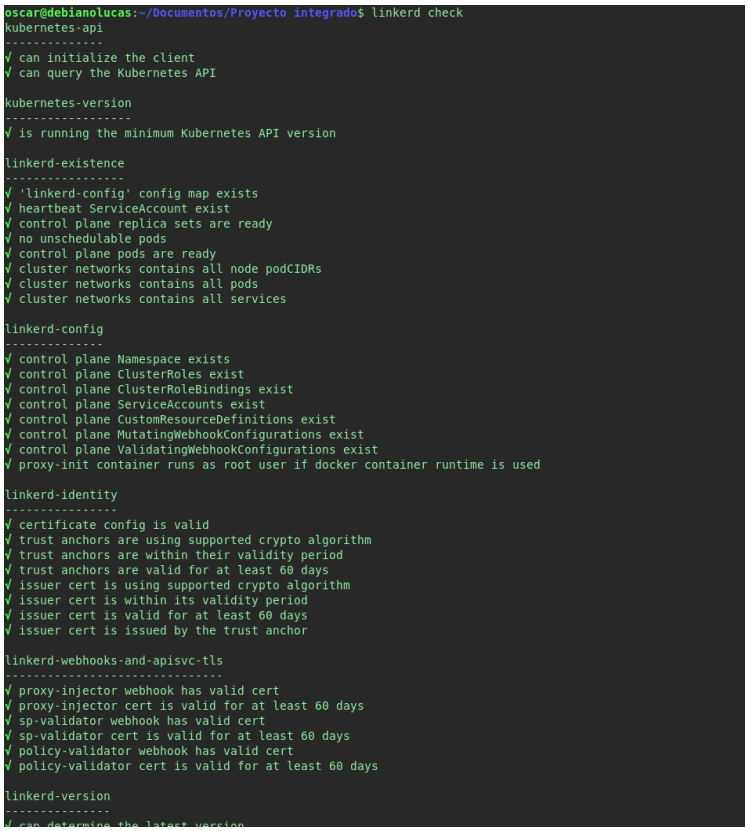
```
linkerd install --set proxyInit.runAsRoot=true | kubectl
apply -f –
```



Este comando instala Linkerd en tu clúster de Kubernetes.

Verificar la instalación de Linkerd:

```
linkerd check
```



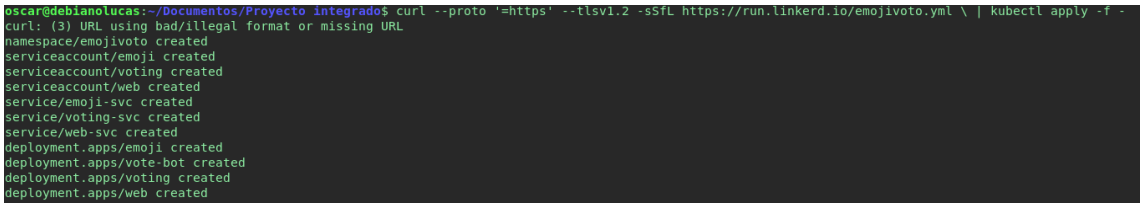
Después de la instalación, es recomendable verificar que Linkerd esté funcionando correctamente.

5.4 Demo

5.4.1 Implementación de Aplicación Web con Linkerd

Ejecuta el siguiente comando para descargar y aplicar el manifiesto de Emojivoto en tu clúster de Kubernetes:

```
curl --proto '=https' --tlsv1.2 -sSfL
https://run.linkerd.io/emojivoto.yml | kubectl apply -f -
```



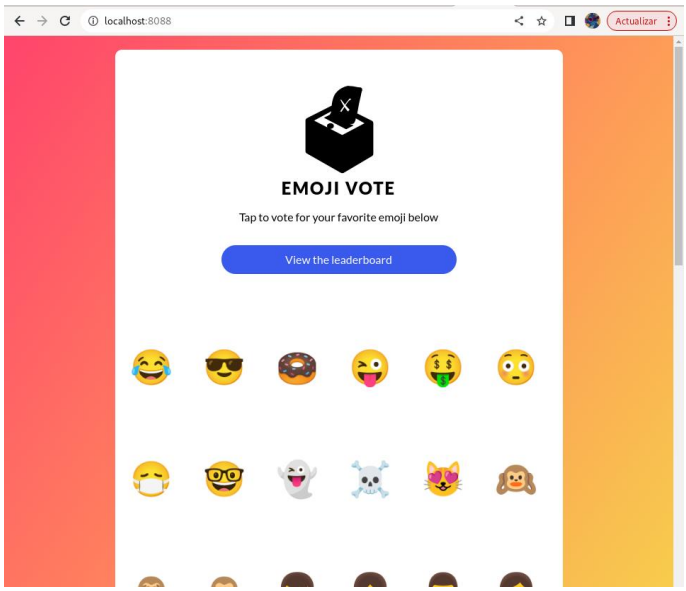
Reenviar Tráfico al Servicio Web de Emojivoto

Si deseas ver Emojivoto en su estado natural antes de la inyección de Linkerd, ejecuta el siguiente comando para reenviar el tráfico localmente al servicio web de Emojivoto:

```
kubectl -n emojivoto port-forward svc/web-svc 8088:80
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl -n emojivoto port-forward svc/web-svc 8088:80
Forwarding from 127.0.0.1:8088 -> 8080
Forwarding from [::1]:8088 -> 8080
Handling connection for 8088
Handling connection for 8088
```

Visita http://localhost:8088 en tu navegador para interactuar con Emojivoto.



Inyectar el Proxy de Linkerd en la Aplicación:

Ejecuta el siguiente comando para inyectar el proxy de Linkerd en los recursos de Emojivoto:

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

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl get -n emojivoto deploy -o yaml \
  | linkerd inject - \
  | kubectl apply -f -

deployment "emoji" injected
deployment "vote-bot" injected
deployment "voting" injected
deployment "web" injected

deployment.apps/emoji configured
deployment.apps/vote-bot configured
deployment.apps/voting configured
deployment.apps/web configured
```

Este comando recupera la configuración de implementación de Emojivoto, ejecuta la inyección de Linkerd y luego aplica la configuración modificada en el clúster.

Verificar la Inyección de Linkerd

Puedes verificar que los proxies de Linkerd se han inyectado en los pods de Emojivoto ejecutando:

```
kubectl -n emojivoto get pods
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl -n emojivoto get pods
NAME                        READY   STATUS    RESTARTS   AGE
emoji-9f6758b4d-j8knh       2/2     Running   0          23h
vote-bot-db7d9c4d9-55db6    2/2     Running   0          23h
voting-5d66f899b7-68pdj     2/2     Running   0          23h
web-8559b97f7c-vfgt4        2/2     Running   0          23h
oscar@debianolucas:~/Documentos/Proyecto integrado$
```

Deberías ver los pods de Emojivoto con los proxies de Linkerd.

Verificar el Estado de Linkerd en el Plano de Datos

Para asegurarte de que todo esté funcionando correctamente en el plano de datos, puedes ejecutar el siguiente comando:

```
linkerd -n emojivoto check --proxy
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ linkerd -n emojivoto check --proxy
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ control plane pods are ready
√ cluster networks contains all node podCIDRs
√ cluster networks contains all pods
√ cluster networks contains all services

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ proxy-init container runs as root user if docker container runtime is used

linkerd-identity
----------------
√ certificate config is valid
√ trust anchors are using supported crypto algorithm
√ trust anchors are within their validity period
√ trust anchors are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust anchor

linkerd-webhooks-and-apisvc-tls
-------------------------------
√ proxy-injector webhook has valid cert
√ proxy-injector cert is valid for at least 60 days
√ sp-validator webhook has valid cert
√ sp-validator cert is valid for at least 60 days
√ policy-validator webhook has valid cert
√ policy-validator cert is valid for at least 60 days

linkerd-identity-data-plane
---------------------------
```

Este comando verifica el estado de Linkerd en el espacio de nombres de Emojivoto.

A continuación, instalamos una extensión mediante el panel de control (Viz) de Linkerd. Además, observaremos que instala Prometheus para el monitoreo y Grafana para una visualización más efectiva de estas métricas.

5.4.2 Configuración de Linkerd-Viz (Dashboard)

```
linkerd viz install | kubectl apply -f –
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ linkerd viz install | kubectl apply -f -
namespace/linkerd-viz created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-metrics-api created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-metrics-api created
serviceaccount/metrics-api created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-prometheus created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-prometheus created
serviceaccount/prometheus created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-admin created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-auth-delegator created
serviceaccount/tap created
rolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-auth-reader created
secret/tap-k8s-tls created
apiservice.apiregistration.k8s.io/v1alpha1.tap.linkerd.io created
role.rbac.authorization.k8s.io/web created
rolebinding.rbac.authorization.k8s.io/web created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-check created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-check created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-admin created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-api created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-api created
serviceaccount/web created
service/metrics-api created
deployment.apps/metrics-api created
server.policy.linkerd.io/metrics-api created
authorizationpolicy.policy.linkerd.io/metrics-api created
meshtlsauthentication.policy.linkerd.io/metrics-api-web created
networkauthentication.policy.linkerd.io/kubelet created
configmap/prometheus-config created
service/prometheus created
deployment.apps/prometheus created
server.policy.linkerd.io/prometheus-admin created
authorizationpolicy.policy.linkerd.io/prometheus-admin created
service/tap created
deployment.apps/tap created
server.policy.linkerd.io/tap-api created
authorizationpolicy.policy.linkerd.io/tap created
clusterrole.rbac.authorization.k8s.io/linkerd-tap-injector created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-tap-injector created
serviceaccount/tap-injector created
secret/tap-injector-k8s-tls created
mutatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-tap-injector-webhook-config created
service/tap-injector created
deployment.apps/tap-injector created
server.policy.linkerd.io/tap-injector-webhook created
authorizationpolicy.policy.linkerd.io/tap-injector created
networkauthentication.policy.linkerd.io/kube-api-server created
service/web created
deployment.apps/web created
serviceprofile.linkerd.io/metrics-api.linkerd-viz.svc.cluster.local created
serviceprofile.linkerd.io/prometheus.linkerd-viz.svc.cluster.local created
oscar@debianolucas:~/Documentos/Proyecto integrado$
```

Este comando instala Linkerd Viz, que es una extensión de Linkerd que proporciona visualizaciones y monitoreo para tus servicios en Kubernetes.

Verificar la instalación de Linkerd Viz

```
linkerd check
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ linkerd check
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ control plane pods are ready
√ cluster networks contains all node podCIDRs
√ cluster networks contains all pods
√ cluster networks contains all services

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ proxy-init container runs as root user if docker container runtime is used

linkerd-identity
----------------
√ certificate config is valid
√ trust anchors are using supported crypto algorithm
√ trust anchors are within their validity period
√ trust anchors are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust anchor

linkerd-webhooks-and-apisvc-tls
-------------------------------
√ proxy-injector webhook has valid cert
√ proxy-injector cert is valid for at least 60 days
√ sp-validator webhook has valid cert
√ sp-validator cert is valid for at least 60 days
√ policy-validator webhook has valid cert
√ policy-validator cert is valid for at least 60 days
```

```
linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

control-plane-version
---------------------
√ can retrieve the control plane version
√ control plane is up-to-date
√ control plane and cli versions match

linkerd-control-plane-proxy
---------------------------
√ control plane proxies are healthy
√ control plane proxies are up-to-date
√ control plane proxies and cli versions match

linkerd-viz
-----------
√ linkerd-viz Namespace exists
√ can initialize the client
√ linkerd-viz ClusterRoles exist
√ linkerd-viz ClusterRoleBindings exist
√ tap API server has valid cert
√ tap API server cert is valid for at least 60 days
√ tap API service is running
√ linkerd-viz pods are injected
√ viz extension pods are running
√ viz extension proxies are healthy
√ viz extension proxies are up-to-date
√ viz extension proxies and cli versions match
√ prometheus is installed and configured correctly
√ viz extension self-check

Status check results are √
oscar@debianolucas:~/Documentos/Proyecto integrado$
```

Después de instalar Linkerd Viz, es recomendable verificar nuevamente que todo esté configurado correctamente.

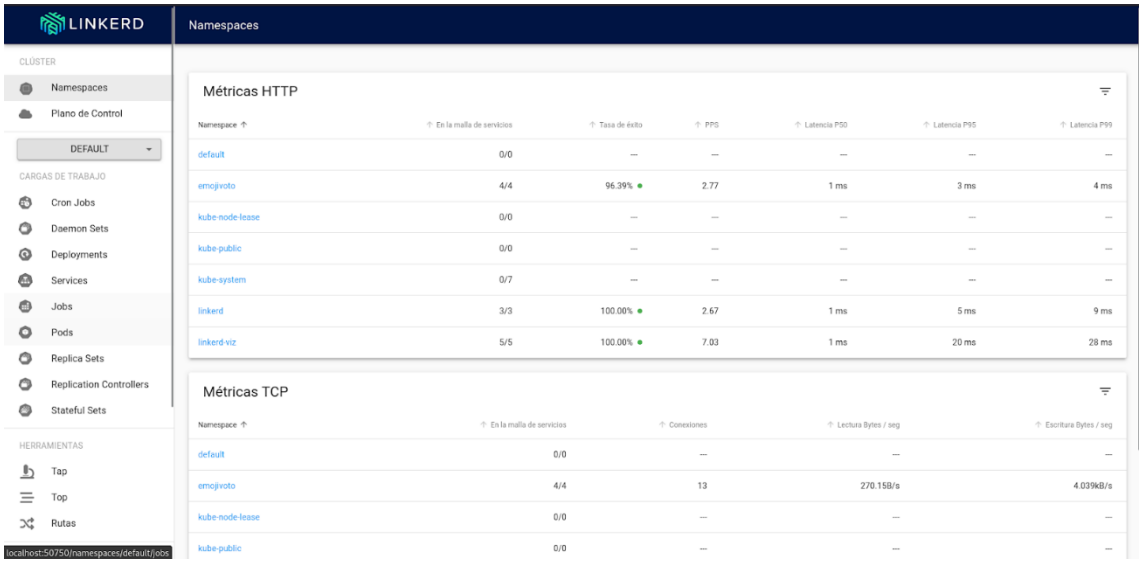
Abrir el panel de control de Linkerd Viz

```
linkerd viz dashboard &
```

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ Linkerd dashboard available at:
http://localhost:50750
Grafana dashboard available at:
http://localhost:50750/grafana
Opening Linkerd dashboard in the default browser
```

Este comando abre el panel de control de Linkerd Viz en tu navegador.

Dashboard Viz de Linkerd



5.4.3 Instalación de Dashboards de Prometheus y Grafana

Si la instalacion de Prometheus y Grafana no la incluye nuestra versión de linkerd podemos instalarlos con Helm

Instalación de Prometheus con Helm (en el namespace de linkerd)

1. Añadimos la repo de prometheus a Helm

```
helm repo add prometheus-community https://prometheus
community.github.io/helm-charts
```

2. Instalamos prometheus con helm

```
helm install prometheus prometheus-community/prometheus --
namespace linkerd
```

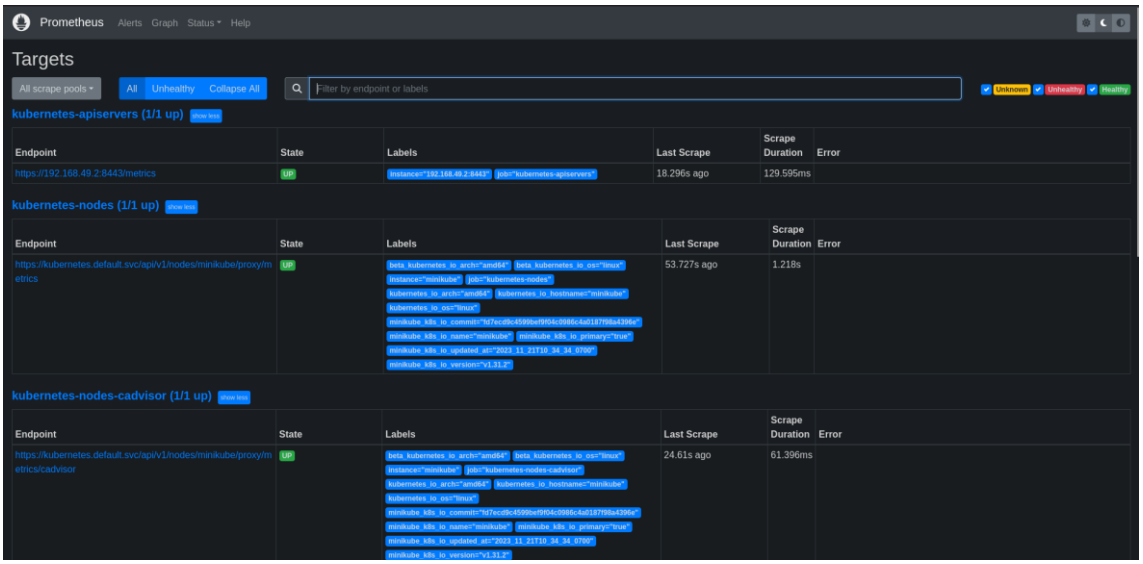
3. Exponemos el puerto de prometheus

```
kubectl expose service prometheus-server --type=NodePort --
target-port=9090 --name=prometheus-server-np --namespace
linkerd
```

4. Arrancamos el servicio de prometheus y lanzamos el dashboard de prometheus

```
minikube service prometheus-server-np -n linkerd &
```

Prometheus dashboard



Instalación de Grafana con Helm (en el namespace de linkerd)

1. Agregamos la repo de grafana

```
helm repo add grafana https://grafana.github.io/helm-charts
```

2. Instalamos grafana con Helm

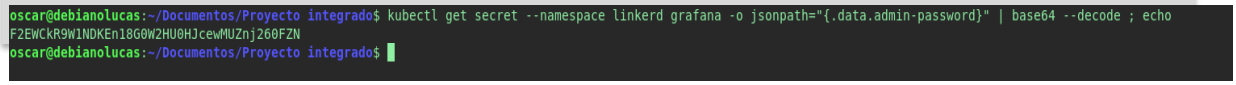
```
helm install grafana grafana/grafana --namespace linkerd
```

3. Exponemos el puerto de grafana

```
kubectl expose service grafana --type=NodePort --target-
port=3000 --name=grafana-np --namespace linkerd
```

4. Obtenemos la clave de admin para grafana

```
kubectl get secret --namespace linkerd grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
```



5. Iniciamos el servicio de grafana y lanzamos el dashboard

```
minikube service grafana-np -n linkerd &
```

Grafana dashboard

Para utilizar Grafana con Prometheus, es necesario configurar un origen de datos de tipo Prometheus:

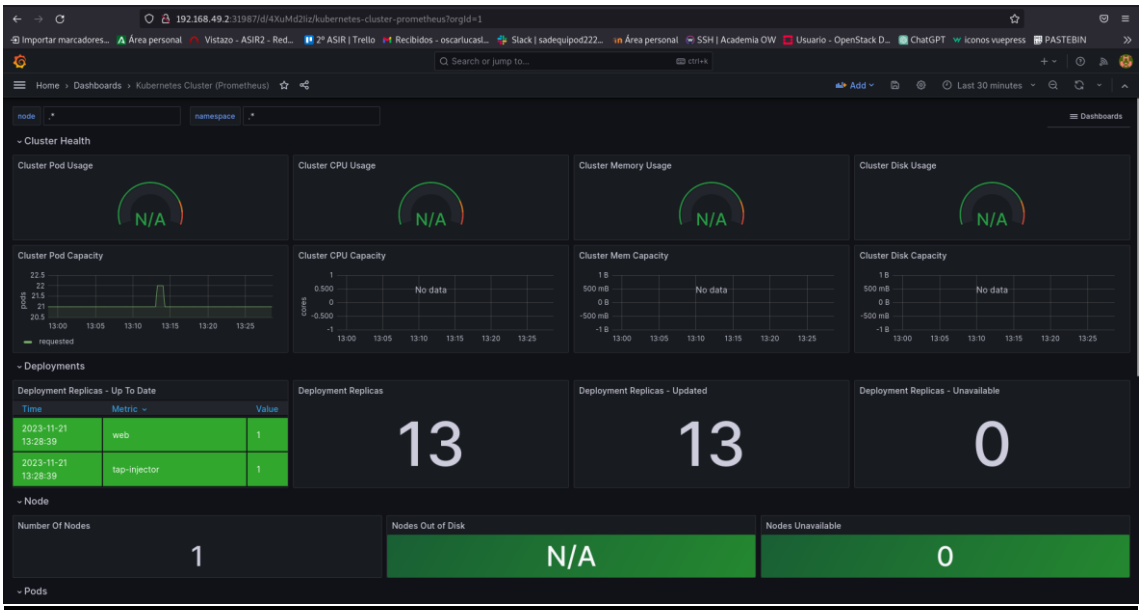
Para ello, accedemos al menú Configuration > Plugin y agregamos una nueva instancia de Prometheus.

En la sección HTTP URL, proporcionamos la dirección y el puerto donde está ubicado nuestro servidor Prometheus.

Posteriormente, para visualizar las métricas, es necesario importar un panel. Para hacerlo, seguimos estos pasos:

1. Navegamos a Dashboards.
2. Seleccionamos Import.
3. Optamos por Import via grafana.com y copiamos el código del panel específico que deseamos utilizar con Kubernetes que es “6417”.

Pulsamos en load ,luego importamos guardamos la configuracion y ya lo tendremos.



El usuario de grafana inicial es: admin

Y la clave la obtenemos con: `kubectl get secret --namespace linkerd grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo`

Pods, services y deploys del namespace linkerd

```
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl get pods -n linkerd
NAME                                      READY   STATUS    RESTARTS   AGE
grafana-74d7cdbbcb-vkcs6                  1/1     Running   0          26h
linkerd-destination-64cbb6d894-xtzvc      4/4     Running   0          26h
linkerd-identity-77779ddd4b-crp5j         2/2     Running   0          26h
linkerd-proxy-injector-7855cb9b5c-dz5sd   2/2     Running   0          26h
prometheus-alertmanager-0                 1/1     Running   0          26h
prometheus-kube-state-metrics-85596bfdb6-gqflt   1/1     Running   0          26h
prometheus-prometheus-node-exporter-wx5mt        1/1     Running   0          26h
prometheus-prometheus-pushgateway-79745d4495-2qxs4   1/1     Running   0          26h
prometheus-server-7fdb9c87d-2z8l5         2/2     Running   0          26h
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl get svc -n linkerd
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
grafana                             ClusterIP   10.104.195.95    <none>        80/TCP         26h
grafana-np                          NodePort    10.105.119.66    <none>        80:31987/TCP   26h
linkerd-dst                         ClusterIP   10.103.99.221    <none>        8086/TCP       26h
linkerd-dst-headless                ClusterIP   None             <none>        8086/TCP       26h
linkerd-identity                    ClusterIP   10.98.63.253     <none>        8080/TCP       26h
linkerd-identity-headless           ClusterIP   None             <none>        8080/TCP       26h
linkerd-policy                      ClusterIP   None             <none>        8090/TCP       26h
linkerd-policy-validator            ClusterIP   10.110.56.143    <none>        443/TCP        26h
linkerd-proxy-injector              ClusterIP   10.102.117.242   <none>        443/TCP        26h
linkerd-sp-validator                ClusterIP   10.101.155.207   <none>        443/TCP        26h
prometheus-alertmanager             ClusterIP   10.106.8.133     <none>        9093/TCP       26h
prometheus-alertmanager-headless    ClusterIP   None             <none>        9093/TCP       26h
prometheus-kube-state-metrics       ClusterIP   10.101.15.184    <none>        8080/TCP       26h
prometheus-prometheus-node-exporter ClusterIP   10.101.186.131   <none>        9100/TCP       26h
prometheus-prometheus-pushgateway   ClusterIP   10.102.210.198   <none>        9091/TCP       26h
prometheus-server                   ClusterIP   10.102.38.157    <none>        80/TCP         26h
prometheus-server-np                NodePort    10.100.254.221   <none>        80:32630/TCP   26h
oscar@debianolucas:~/Documentos/Proyecto integrado$ kubectl get deploy -n linkerd
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
grafana                             1/1     1            1           26h
linkerd-destination                 1/1     1            1           26h
linkerd-identity                    1/1     1            1           26h
linkerd-proxy-injector              1/1     1            1           26h
prometheus-kube-state-metrics       1/1     1            1           26h
prometheus-prometheus-pushgateway   1/1     1            1           26h
prometheus-server                   1/1     1            1           26h
oscar@debianolucas:~/Documentos/Proyecto integrado$
```

6. Conclusiones

1. Implementación Exitosa del Service Mesh: He logrado implementar con éxito un service mesh utilizando Linkerd en el clúster de Minikube. Esto implica que he integrado de manera efectiva los servicios de la aplicación, permitiendo la comunicación, la observabilidad y la gestión del tráfico de manera más eficiente.
2. Monitorización y Métricas Avanzadas: Al integrar Prometheus y Grafana para la monitorización y visualización de métricas, he mejorado la visibilidad de la aplicación. Esto es crucial para comprender el rendimiento de los servicios, identificar posibles problemas y optimizar el rendimiento general.
3. Facilitación de Comunicación entre Servicios: Con la implementación del service mesh, he mejorado la comunicación entre los servicios de la aplicación. Esto puede conducir a una arquitectura más robusta y escalable, ya que cada servicio puede interactuar de manera más eficiente y confiable.
4. Resiliencia Mejorada: La capacidad de Linkerd para manejar automáticamente la resiliencia y la recuperación de fallos puede ser un punto destacado. He fortalecido la fiabilidad de la aplicación al permitir que el service mesh gestione automáticamente situaciones como la recuperación de fallos.
5. Aprendizaje sobre Kubernetes y Service Mesh: A lo largo de este proyecto, es probable que hayas adquirido una comprensión más profunda de Kubernetes y los conceptos asociados a los service mesh. Este conocimiento es valioso y puede aplicarse en futuros proyectos o mejoras.

7. Bibliografía

Página de la instalación de Linkerd: https://linkerd.io/2.14/getting-started/
Página de configuración grafana en Linkerd:
https://linkerd.io/2.14/tasks/grafana/#install-grafana
Mi repositorio de GitHub de la demo: https://github.com/oscarlucas22/Demo-PI