Sesión 06

Seguridad y Observabilidad

Instructor:

ERICK ARÓSTEGUI

earostegui@galaxy.edu.pe

GALAXY
TRAINING

ÍNDICE

GALAXY
TRAINING

01

GALAXY
TRAINING

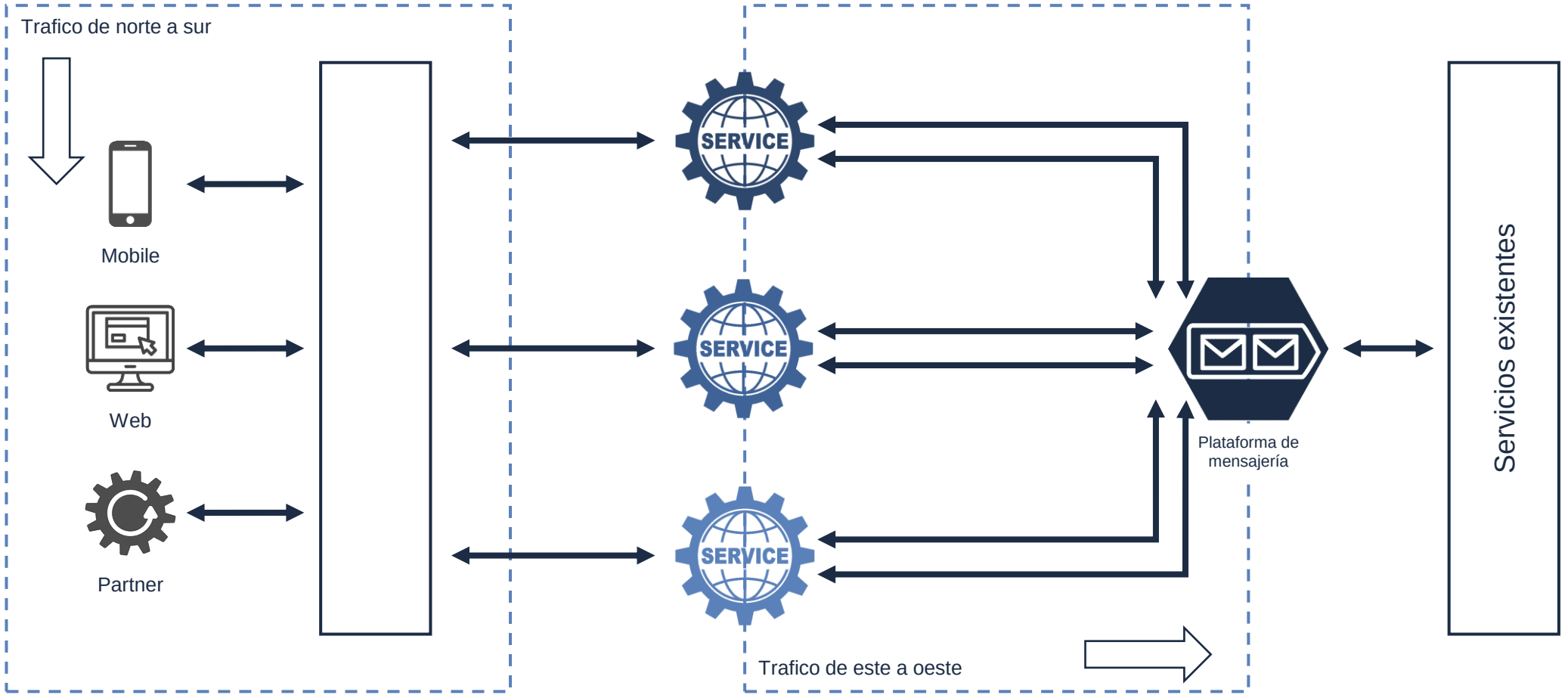
Seguridad en los microservicios (OpenID, OAuth2 y JWT).

Seguridad en los microservicios

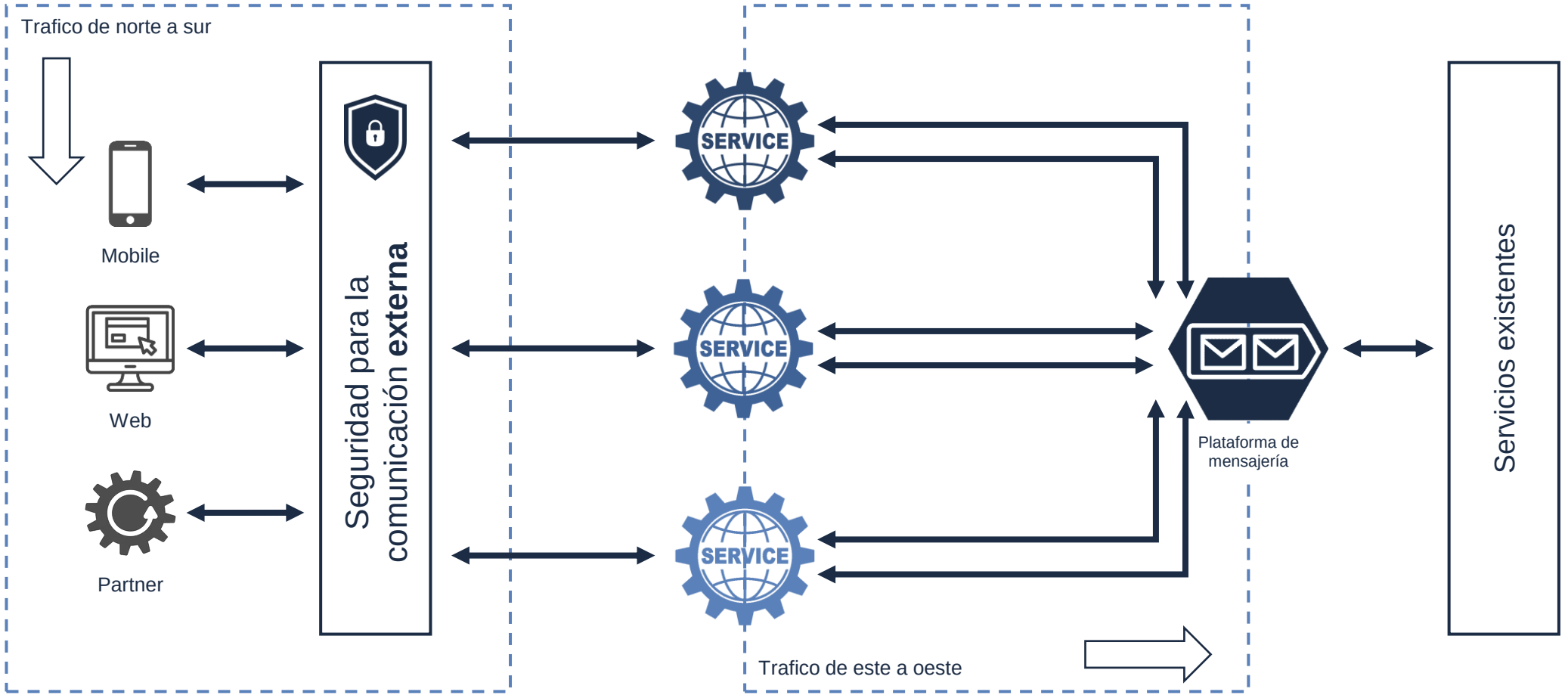
Una de las observaciones de la arquitectura de microservicios es que **“aumenta el riesgo de seguridad”**, debido a que se expande la superficie de riesgo. Cuando tenemos más microservicios que exponen la funcionalidad, tenemos que **proteger cada servicio** de los consumidores externos.

En una **aplicación monolítica**, podríamos tener la mayoría de estas **funcionalidades como programas internos** que no están expuestos a consumidores externos.

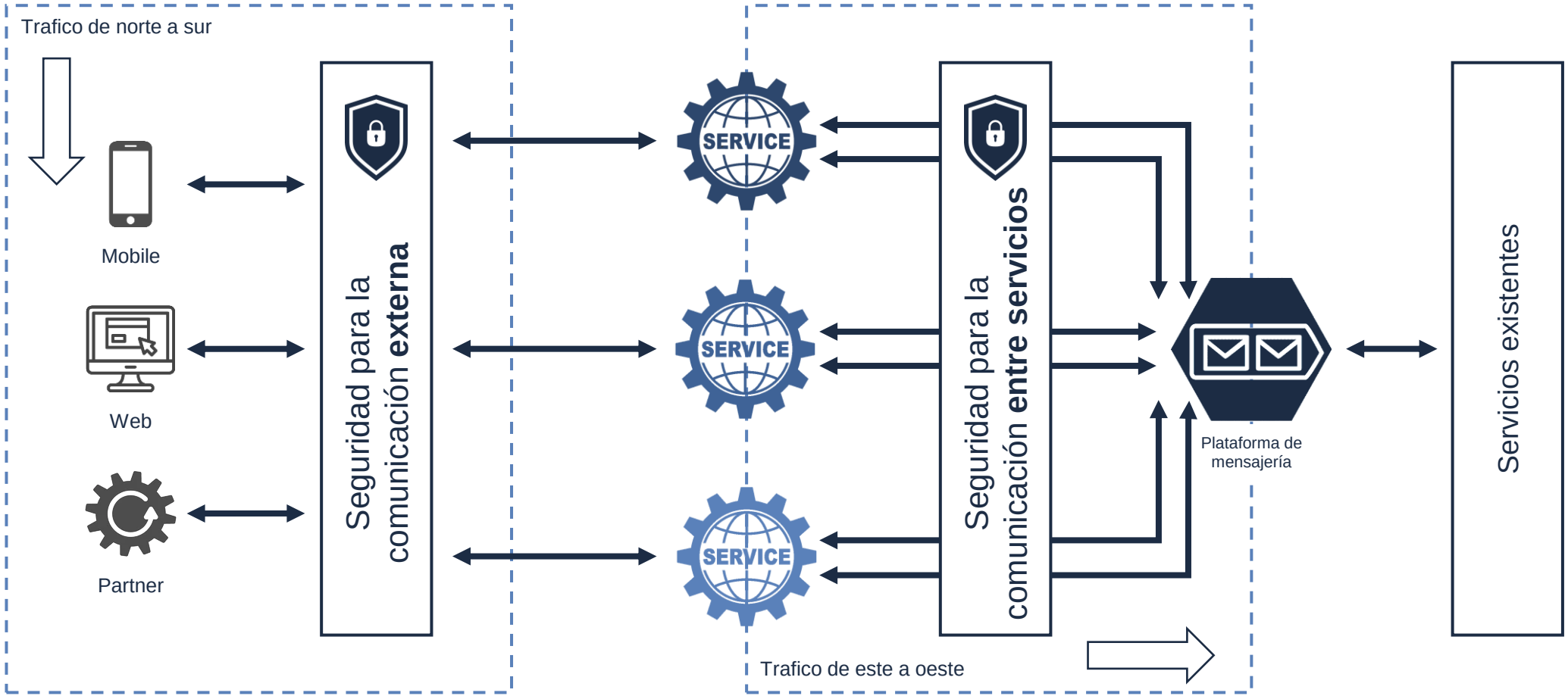
Seguridad en los microservicios



Seguridad en los microservicios



Seguridad en los microservicios



Protección del tráfico norte-sur (consumidores externos)

La mayoría de las empresas se centran más en proteger los microservicios de los **consumidores externos**, ya que ese es **el camino más vulnerable** que puede ser explotado por los malos (hackers).

Podemos implementar seguridad para el tráfico norte-sur utilizando diferentes enfoques.

1. Implementar seguridad en **cada nivel de microservicio**
2. Implementar la seguridad mediante un **sidecar**
3. Implementar la seguridad mediante un **Gateway** compartido

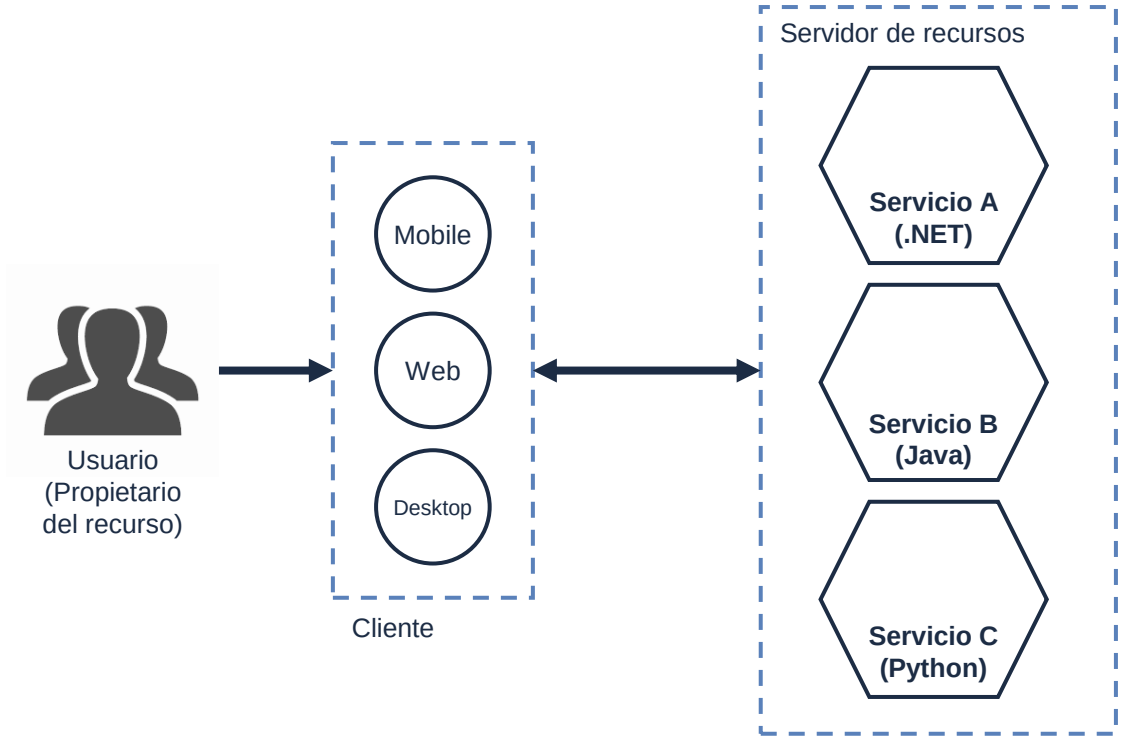
Implementar seguridad en cada nivel de microservicio

Dada la **naturaleza políglota del desarrollo de microservicios**, diferentes equipos pueden crear sus propios lenguajes de programación y marcos para implementar la seguridad.

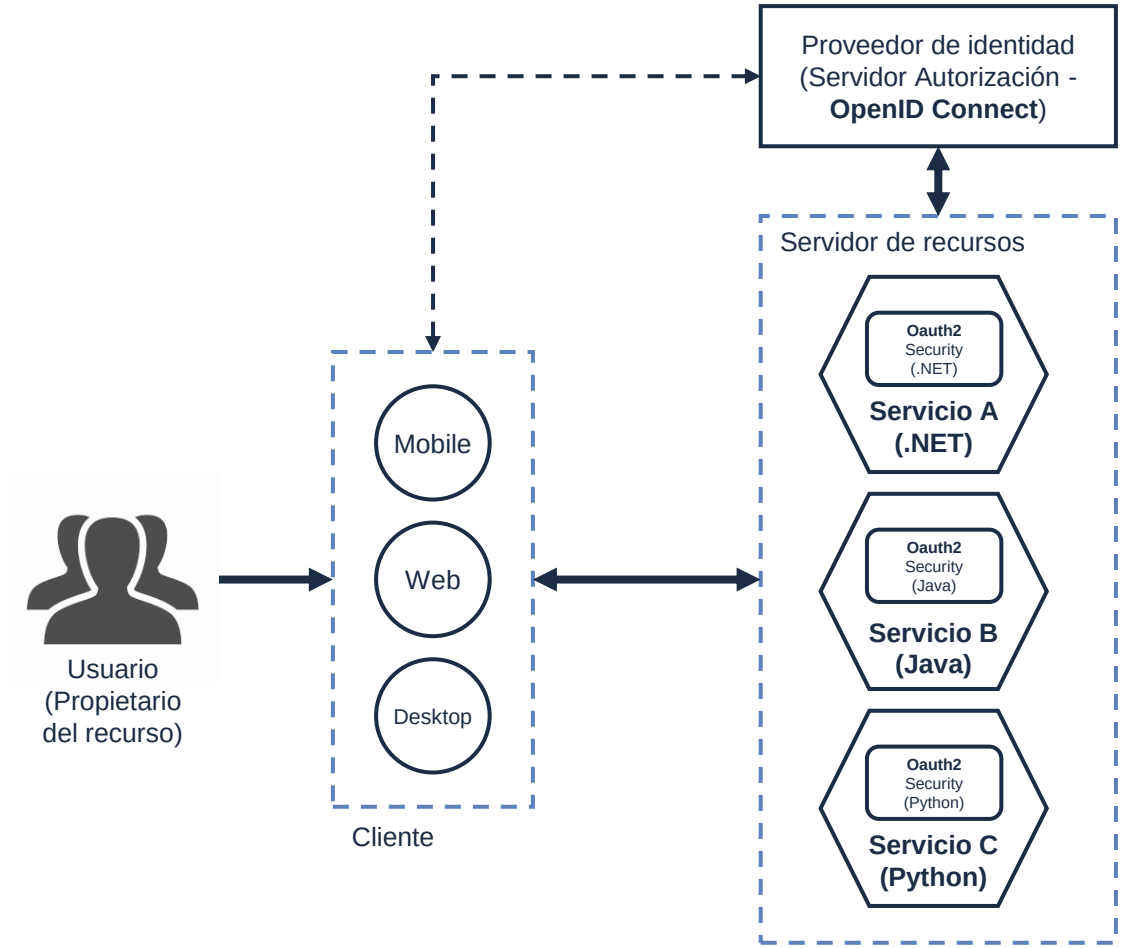
En tal escenario, podemos seguir este enfoque donde **los microservicios individuales implementan seguridad para cada servicio**.

Es importante adherirse a un estándar de seguridad común como **OpenID Connect**, **OAuth 2.0** al implementar la seguridad, ya que eso haría la vida de los clientes mucho más fácil.

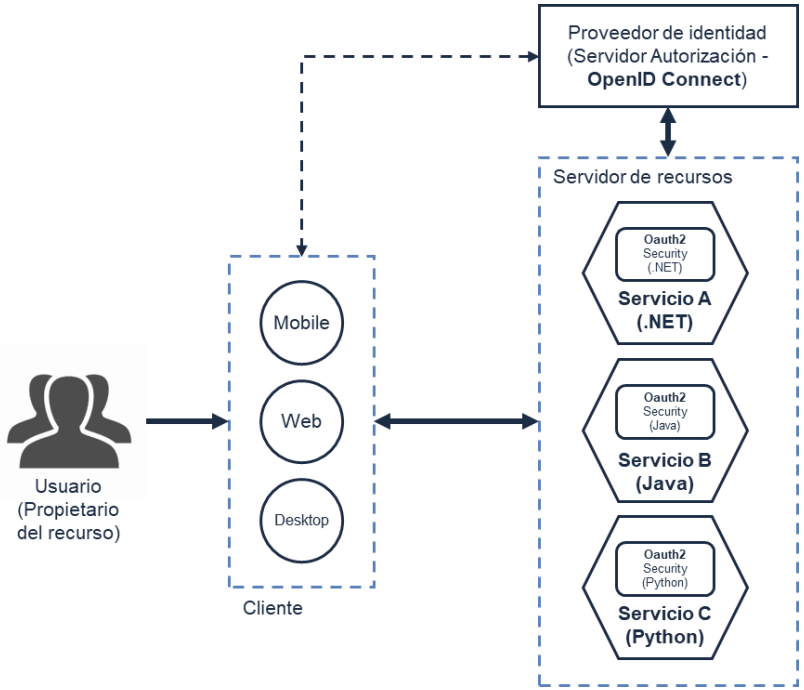
Implementar seguridad en cada nivel de microservicio



Implementar seguridad en cada nivel de microservicio



Implementar seguridad en cada nivel de microservicio



Los microservicios A, B y C se implementan utilizando diferentes lenguajes de programación y **cada microservicio ha implementado seguridad basada en el estándar OAuth2**.

Los clientes se comunicarán con estos servicios utilizando un enfoque común (**token JWT o token opaco**) y se utiliza un **proveedor de identidad (IDP)** común para validar las credenciales de seguridad presentadas por los clientes.

Si el tipo de token es un token JWT autónomo, los microservicios validarán el token por sí mismo sin ponerse en contacto con el IDP.

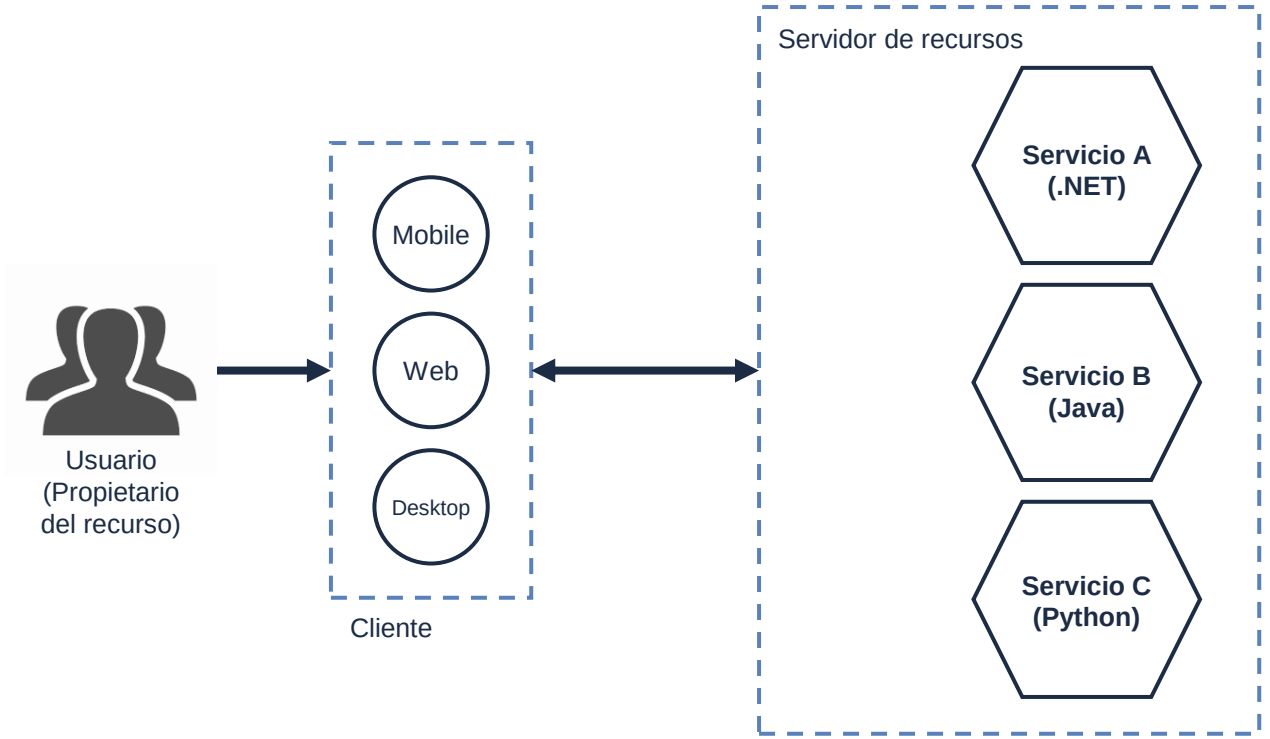
Este enfoque funciona bien y permite a los equipos decidir sobre la mejor tecnología para implementar la seguridad. **Pero la desventaja** de este enfoque es que cada **equipo dedica tiempo** a implementar la misma funcionalidad.

Implementar la seguridad mediante un sidecar

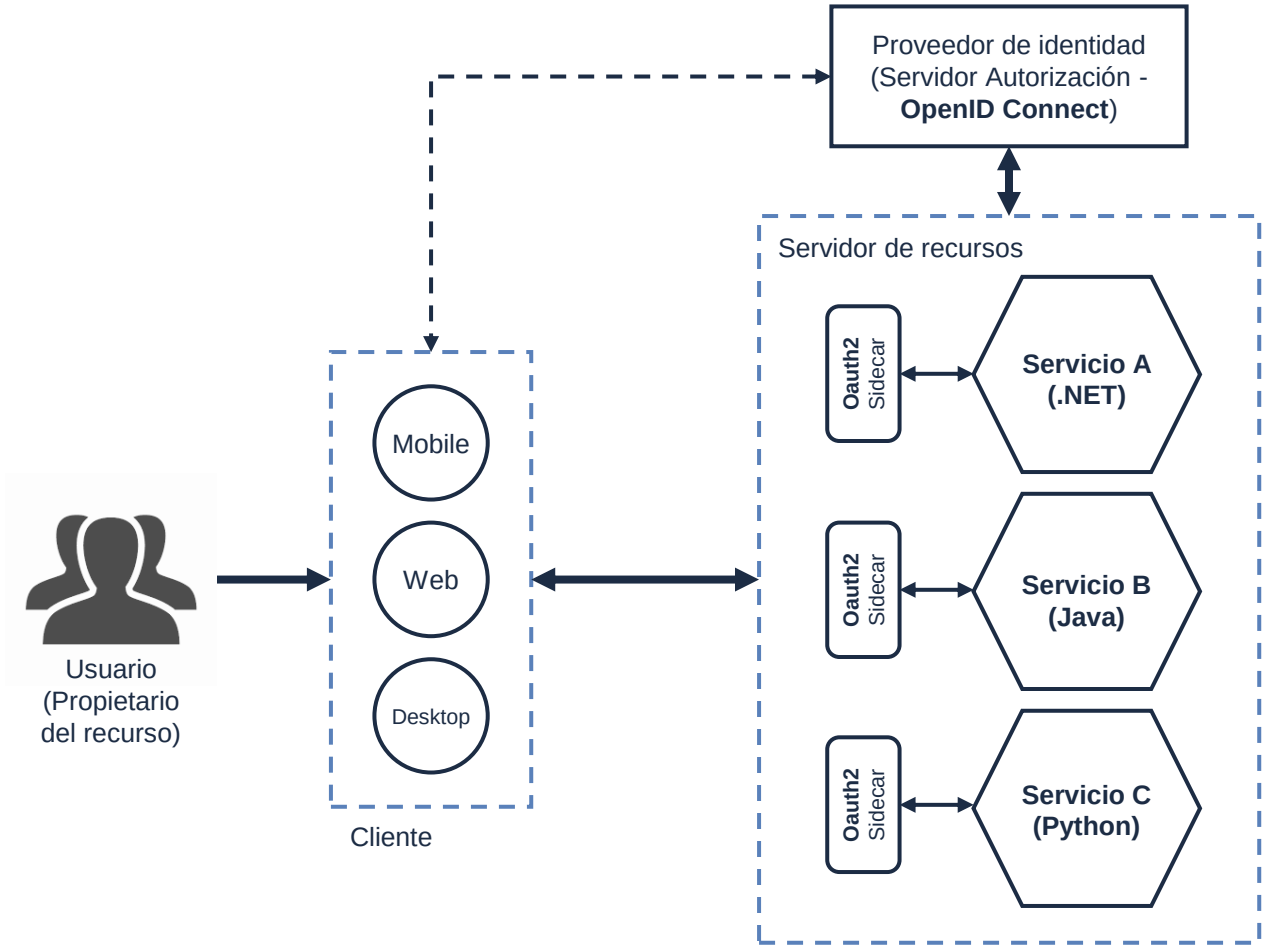
Este enfoque es una ligera mejora con respecto al enfoque anterior.

Aquí usamos un marco externo como **Istio o Open Policy Agent (OPA)** para implementar la seguridad de cada microservicio y este componente de seguridad (**agente**) se ejecuta junto con el microservicio como un sidecar.

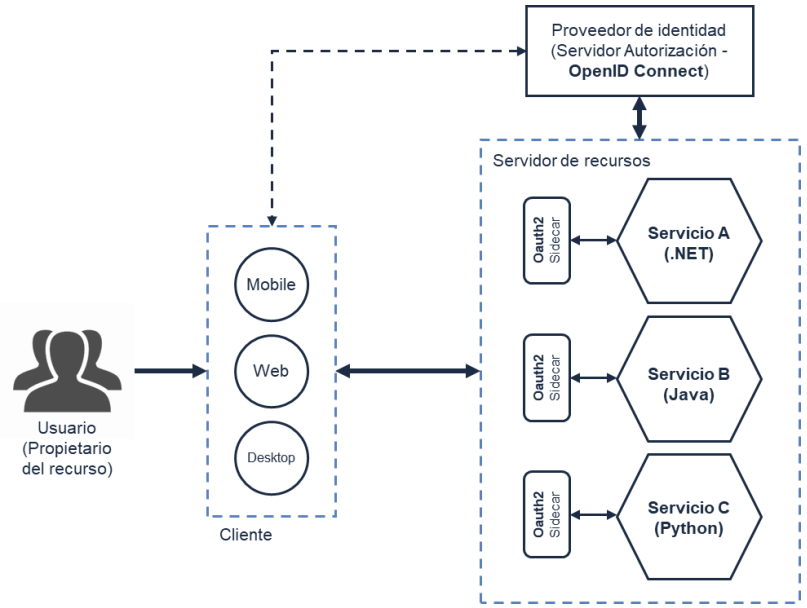
Implementar la seguridad mediante un sidecar



Implementar la seguridad mediante un sidecar



Implementar la seguridad mediante un sidecar

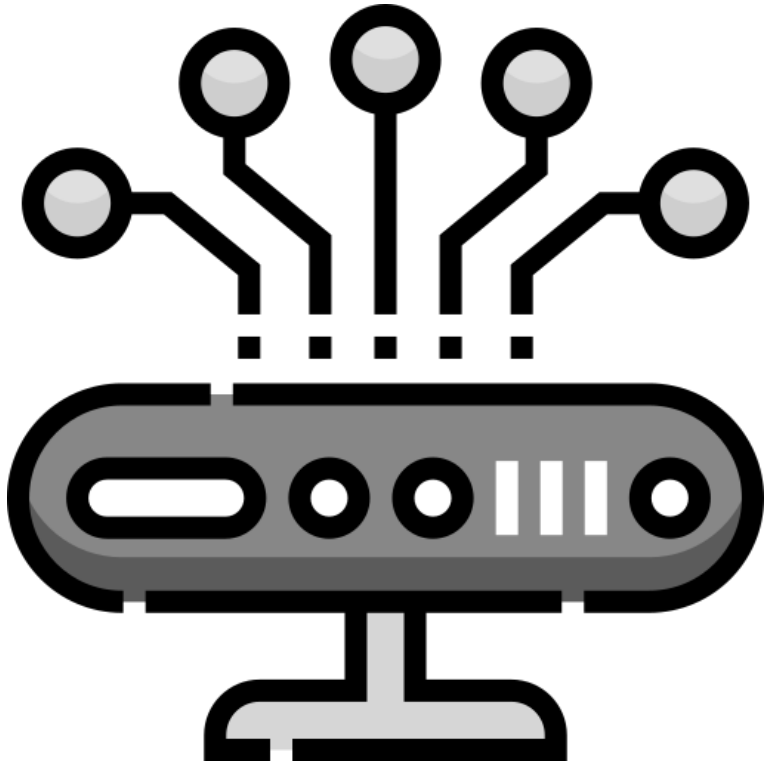


Con este enfoque, los microservicios se pueden implementar de manera políglota y la funcionalidad de **seguridad se maneja a través de un componente externo (sidecar)** que se puede configurar **independientemente del propio microservicio**.

Esto permite al usuario **cambiar las configuraciones de seguridad sin cambiar el código fuente** del microservicio.

También mantiene la **arquitectura general lo más independiente y amigable posible** con los microservicios. La validación de seguridad se puede realizar dentro del propio sidecar o utilizando un servicio separado (IDP).

Implementar la seguridad mediante un Gateway

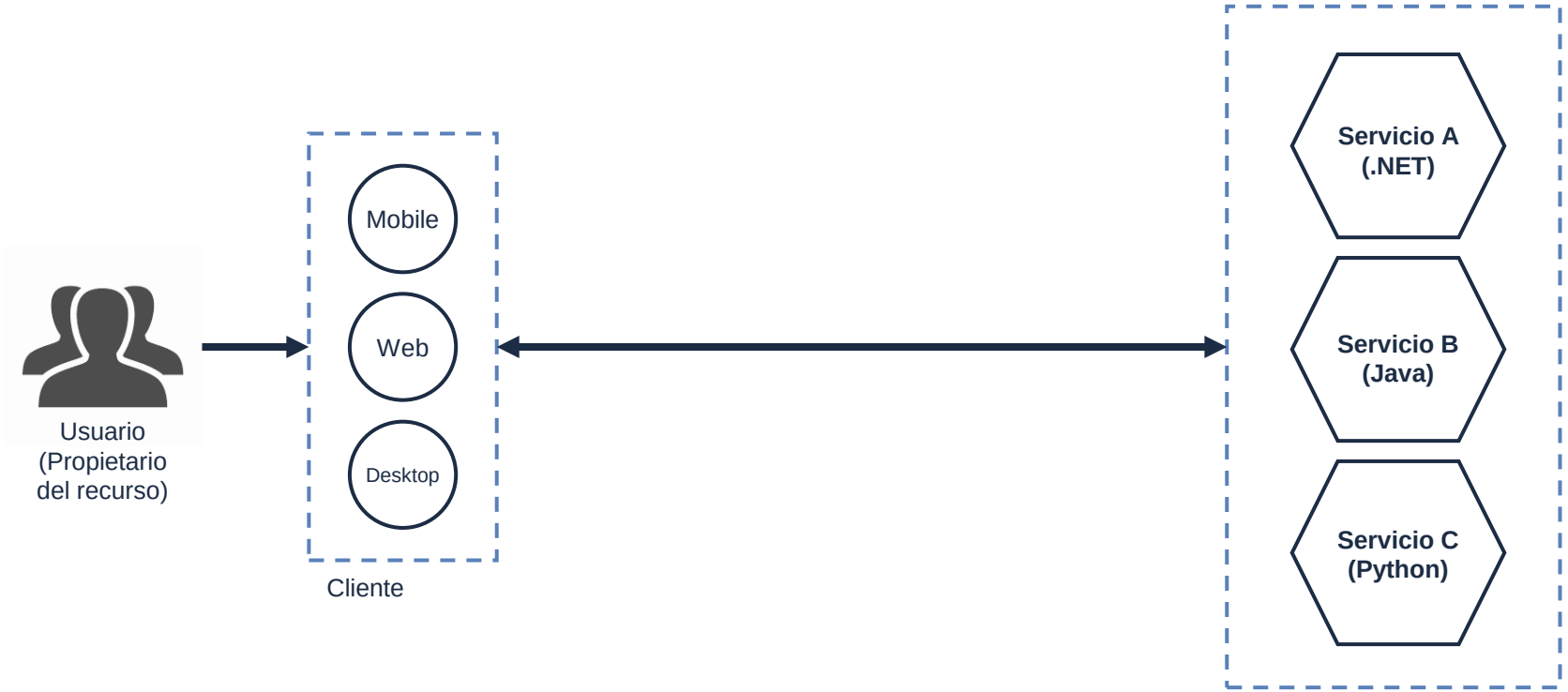


Otro enfoque para implementar la seguridad de los microservicios es **usar un componente compartido** para implementar la seguridad de los microservicios individuales.

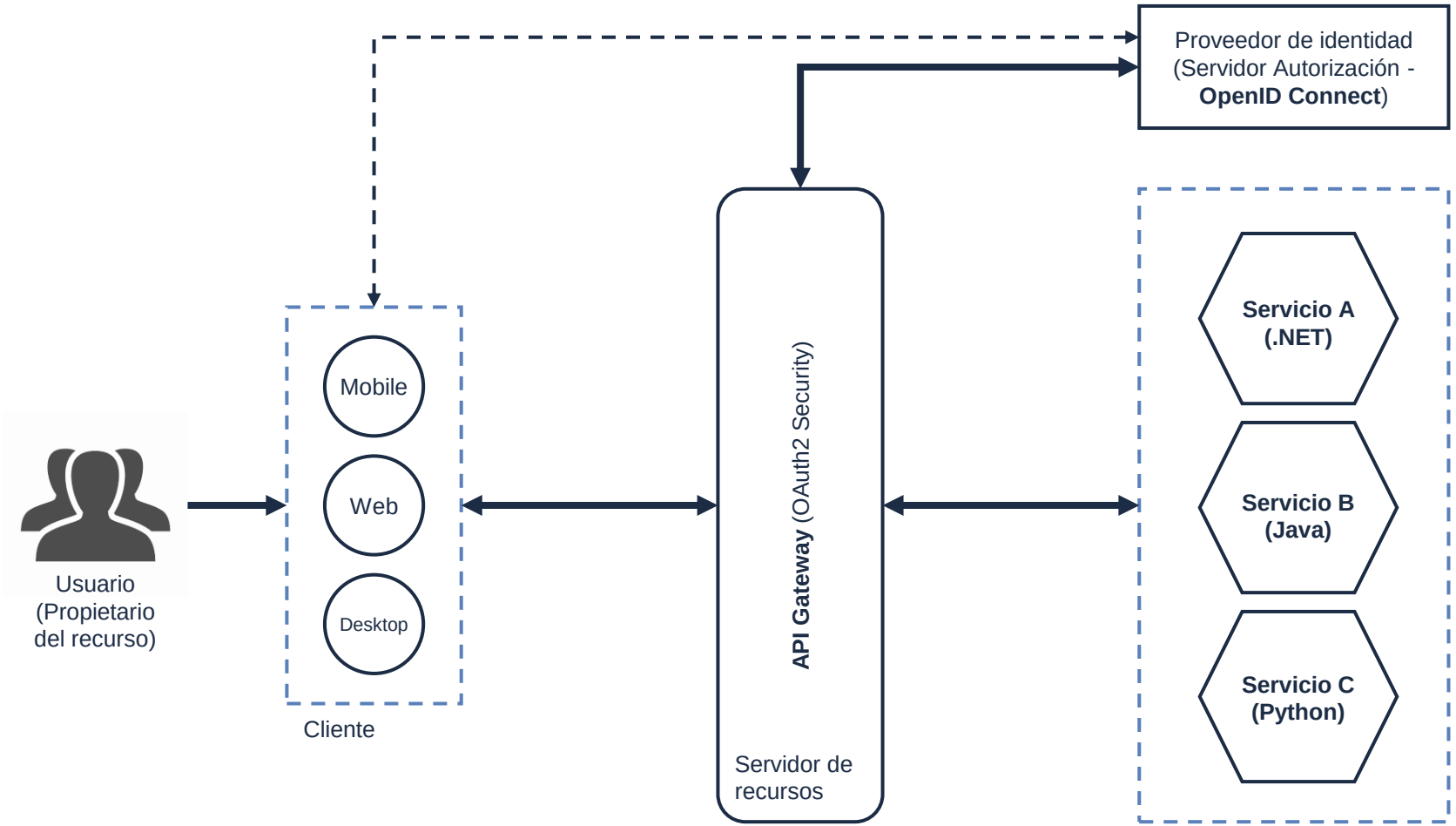
Este componente compartido puede ser un **API Gateway** o un **Security Gateway,** que se ubicará delante de la capa de microservicios.

Cada llamada al microservicio pasará por este componente y se validará para las credenciales antes de comunicarse con el microservicio.

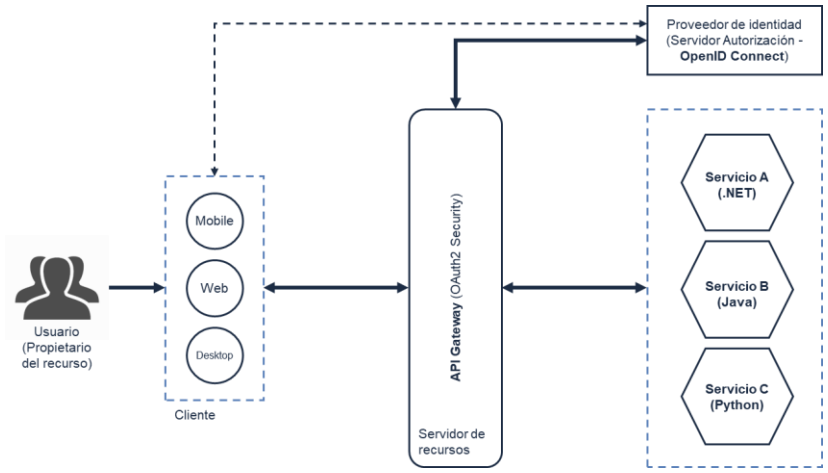
Implementar la seguridad mediante un Gateway



Implementar la seguridad mediante un Gateway



Implementar la seguridad mediante un Gateway



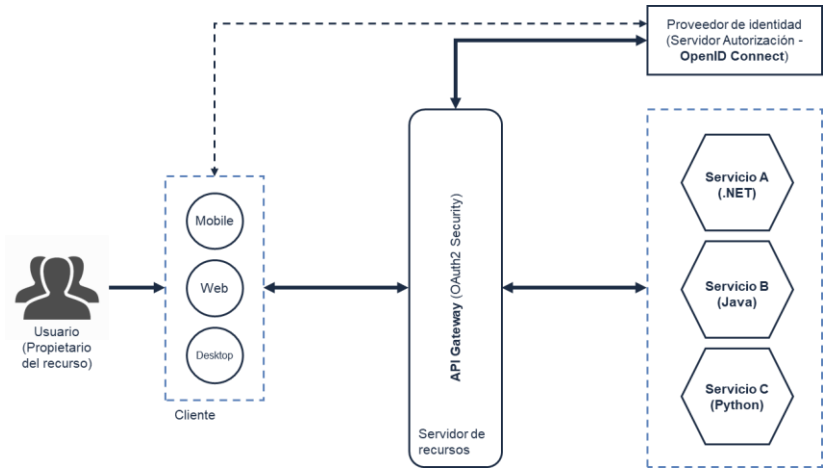
Se usará un **API Gateway** para implementar la seguridad de los microservicios. En esta API Gateway, habrá un servicio proxy que se creará para representar el microservicio e implementar la seguridad para el mismo.

No es necesario tocar los microservicios si necesita cambiar las configuraciones de seguridad.

Un inconveniente de este enfoque es que introduce un componente **monolítico** en la arquitectura general.

En este enfoque, el API Gateway se comunicará con el IDP para validar las credenciales del cliente en función del token o el enfoque de autenticación utilizado para implementar la seguridad.

Implementar la seguridad mediante un Gateway



Existen enfoques más seguros con esta arquitectura en los que tanto el API Gateway como los microservicios individuales están protegidos con seguridad basada en OAuth 2.0.

En tal escenario, el API Gateway puede validar la solicitud del cliente y generar los tokens necesarios de acuerdo con la autenticación de back-end o pasar el token original al back-end directamente si es válido para la autenticación de back-end.

Protección del tráfico entre servicios

El siguiente paso para proteger los microservicios es **proteger la comunicación** entre microservicios (comunicación entre servicios).

En comparación con el tráfico de consumidores externos, la seguridad de la comunicación entre servicios puede considerarse de manera diferente.

Dado que tanto el **consumidor como el proveedor residen en una red interna segura**, a veces está bien aplicar solo el nivel de seguridad requerido en lugar de aplicar seguridad dura para esto.

Protección del tráfico entre servicios

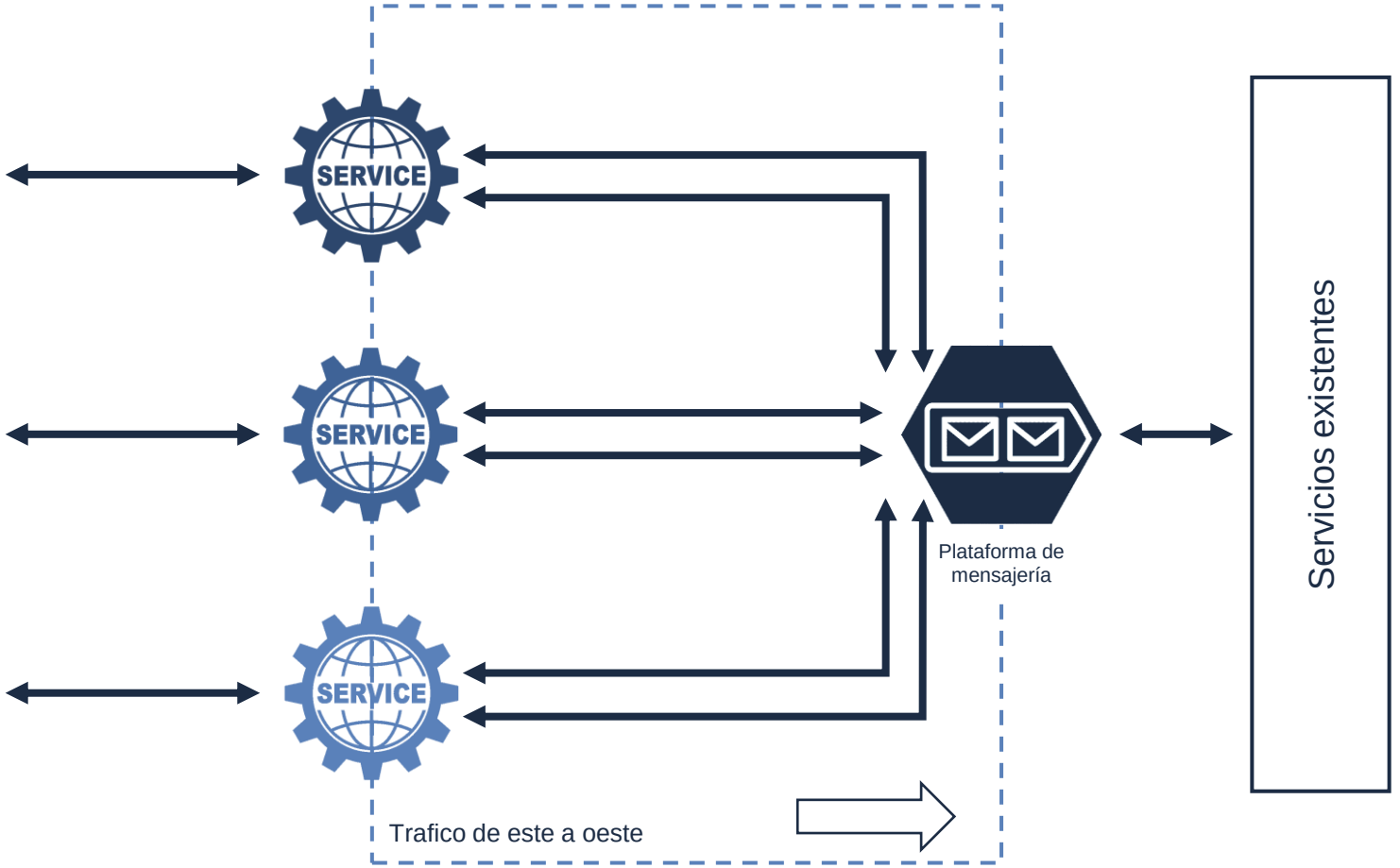
Depende de los requisitos de seguridad de la plataforma empresarial y los estándares de seguridad generales podemos implementar seguridad con los siguientes enfoques:

1. Seguridad de la capa de transporte

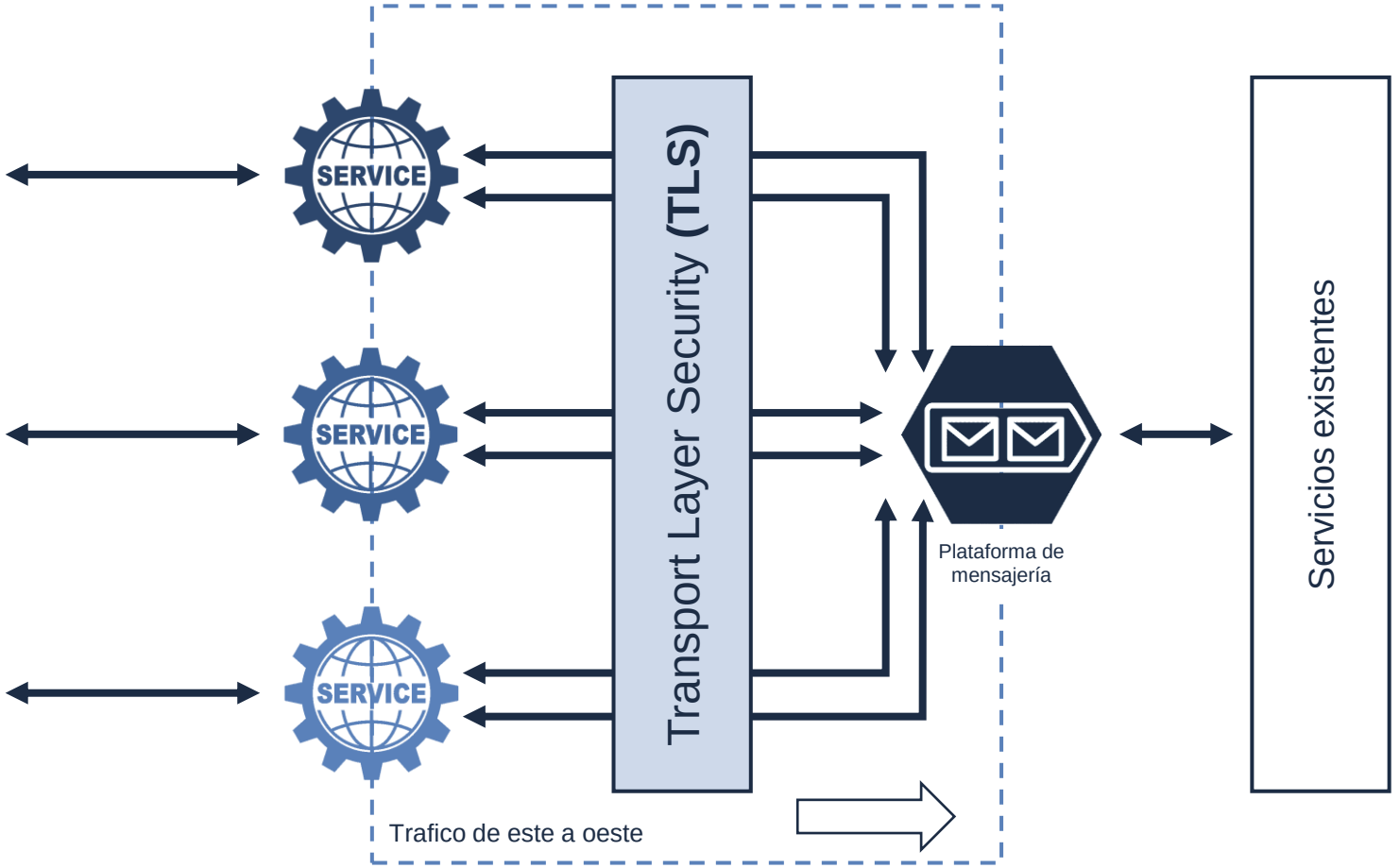
2. Seguridad de la capa de transporte con seguridad de la capa de mensajes mediante autenticación.

3. Seguridad de la capa de transporte con seguridad de la capa de mensajes mediante autenticación y autorización

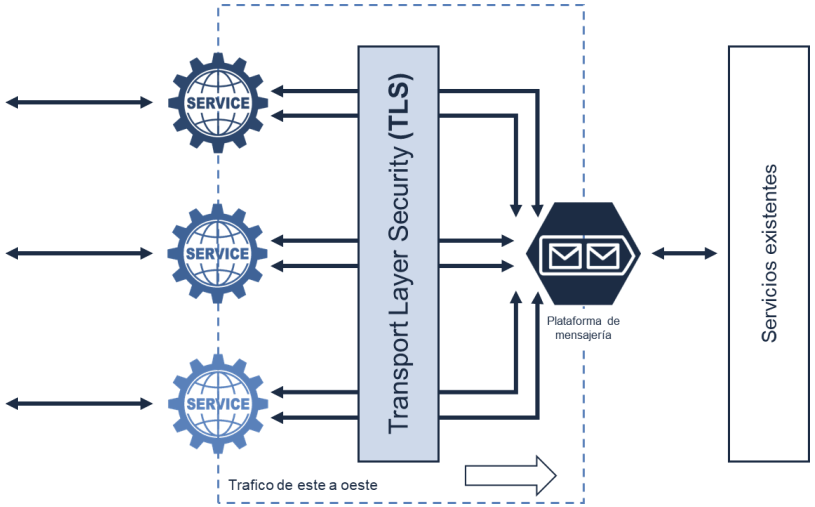
Seguridad de la capa de transporte



Seguridad de la capa de transporte



Seguridad de la capa de transporte

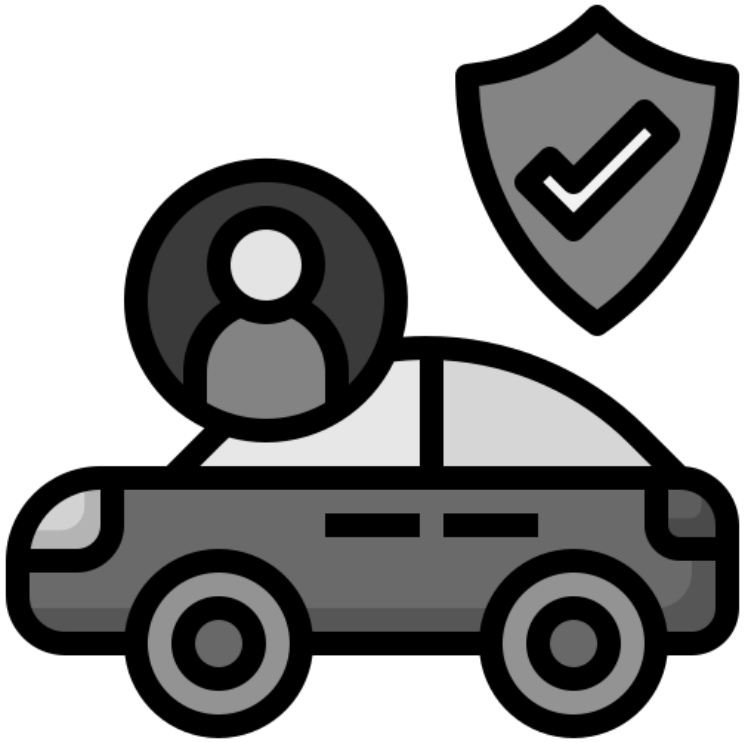


Los **microservicios se comunicarán** con la plataforma de mensajería a **través de TLS**, y la **comunicación se firmará y cifrará** en movimiento.

Ningún tercero podrá ver los datos con este enfoque, que es el nivel máximo de seguridad proporcionado con este enfoque.

Los servicios se comunicarán entre sí a través de la plataforma de mensajería (por ejemplo, Kafka o NATS).

Seguridad de la capa de transporte con autenticación

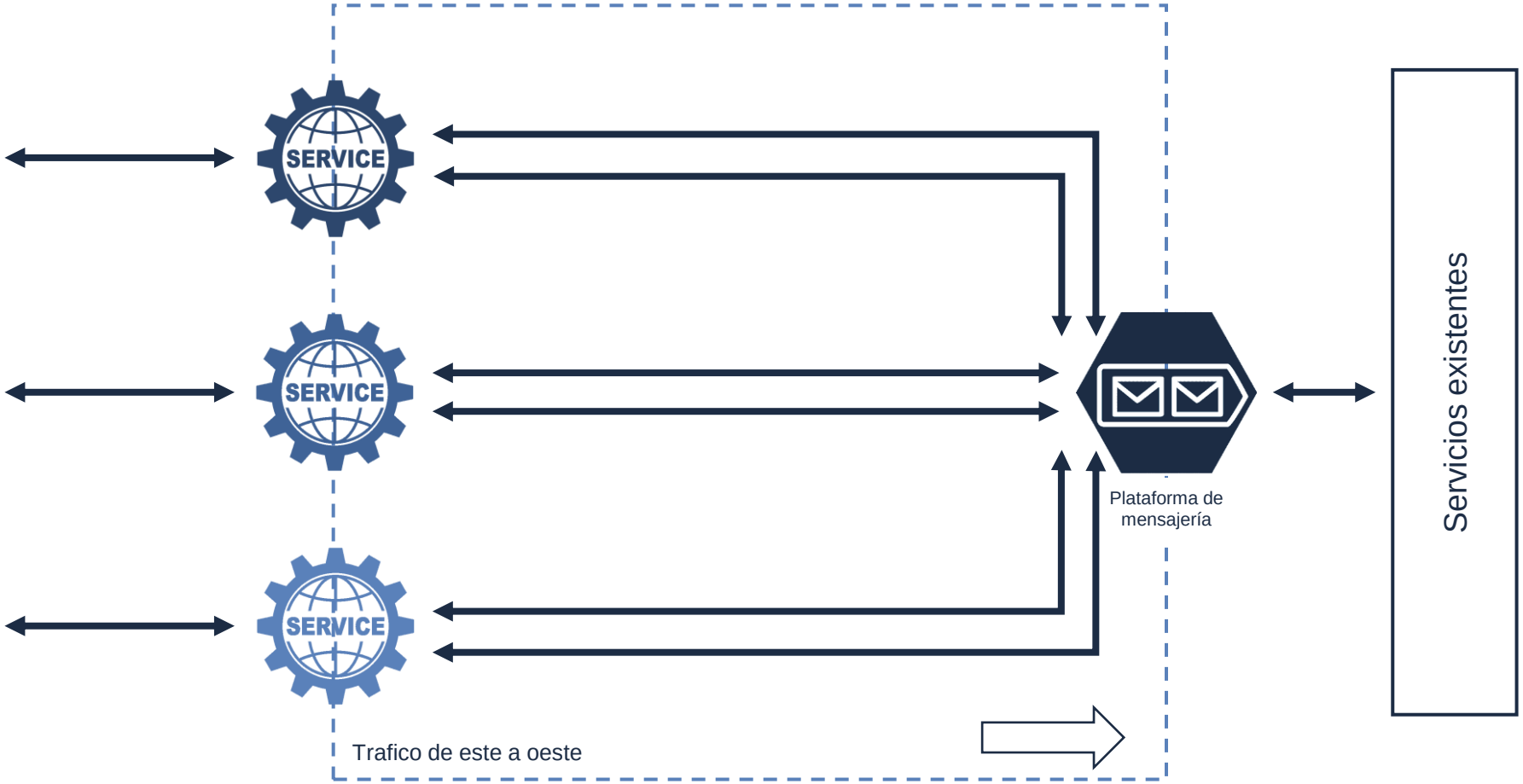


Si el enfoque anterior no es lo suficientemente seguro para la organización, puede **implementar la autenticación** para la comunicación entre los microservicios y la plataforma de mensajería.

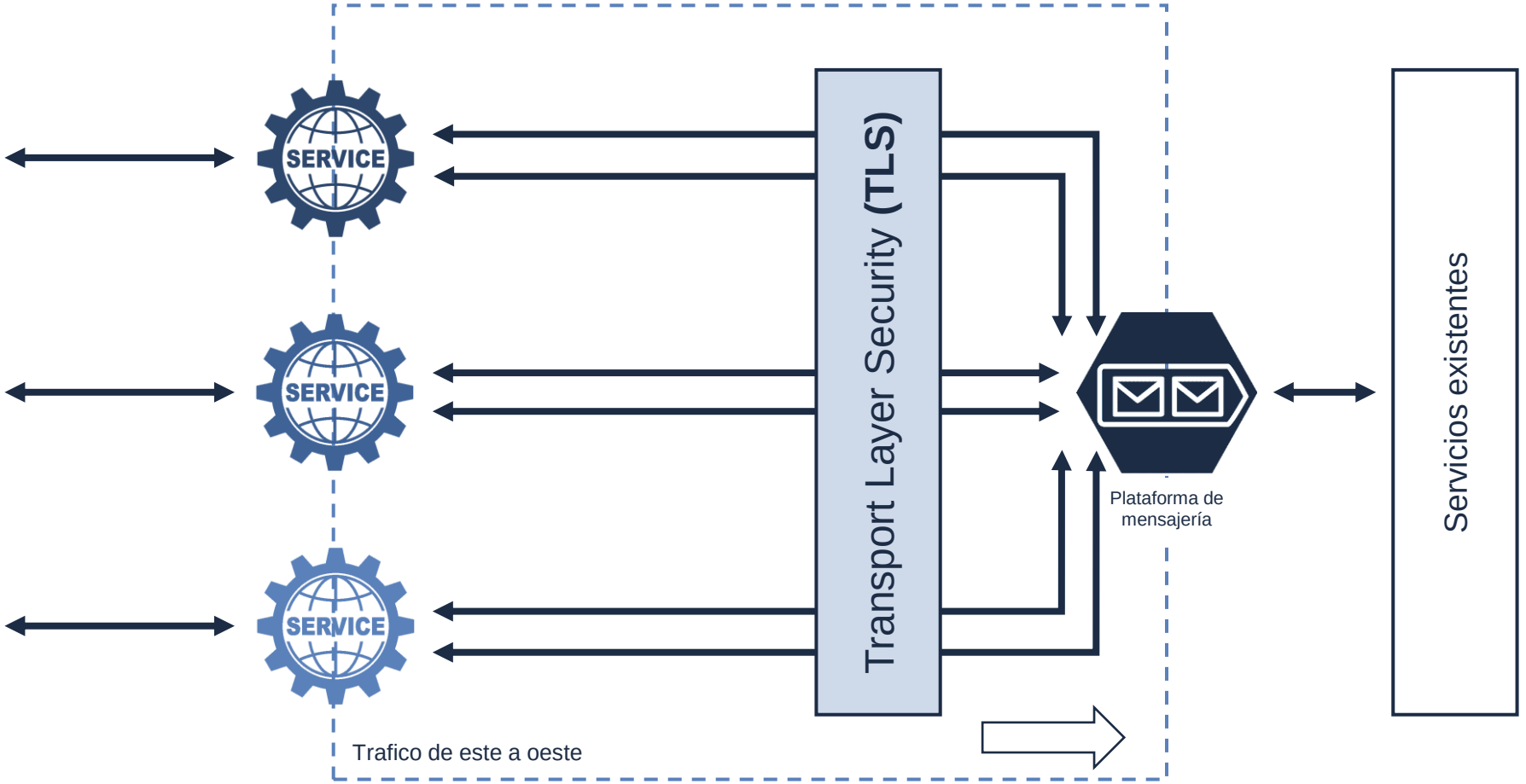
Esto asegurará que solo **los microservicios con credenciales válidas** puedan comunicarse con la plataforma de mensajería y, finalmente, con los otros microservicios.

GALAXY TRAINING

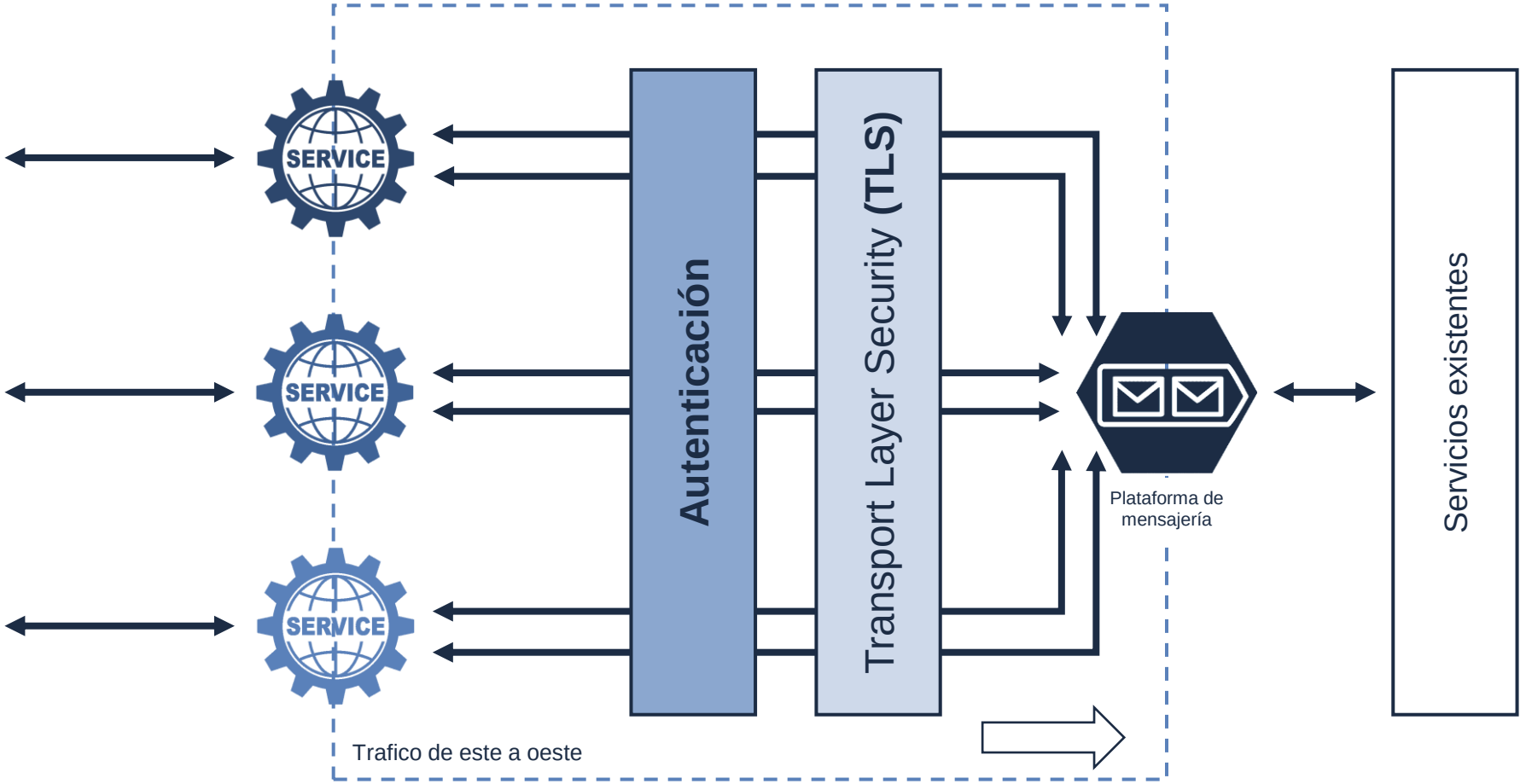
Seguridad de la capa de transporte con autenticación



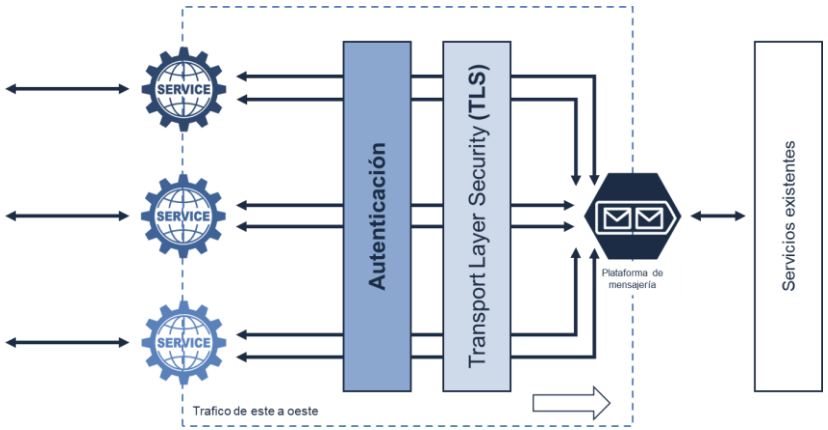
Seguridad de la capa de transporte con autenticación



Seguridad de la capa de transporte con autenticación



Seguridad de la capa de transporte con autenticación



Los microservicios **proporcionan credenciales para validar su comunicación** con la plataforma de mensajería para proporcionar seguridad avanzada.

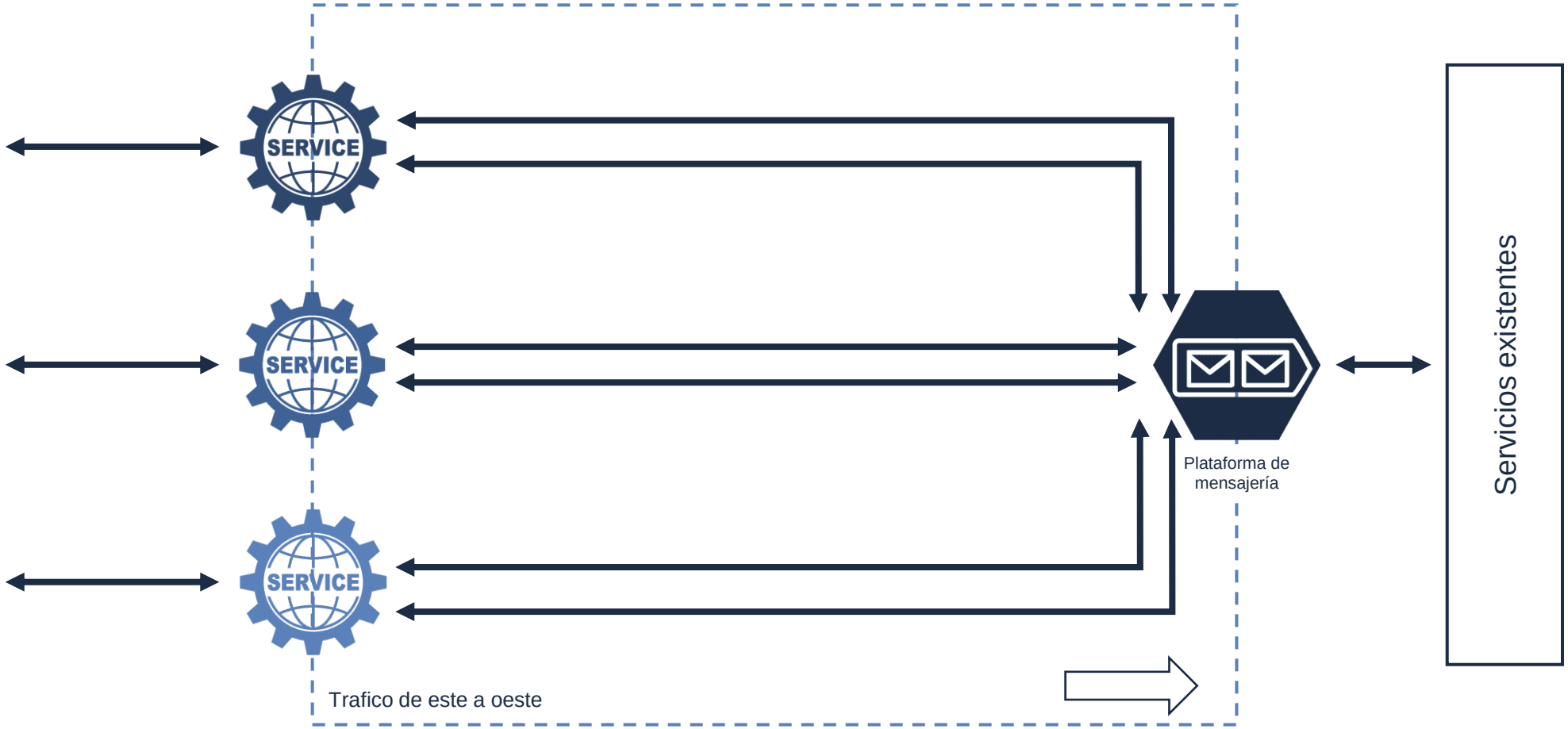
Según la plataforma de mensajería que elija, puede implementar la autenticación mediante un mecanismo como **OAuth 2.0** u otros mecanismos, como la **basic authentication** o la **autenticación basada en tokens**.

Una cosa importante a tener en cuenta aquí es que **no necesariamente necesita usar el mismo mecanismo de seguridad que utilizó para el tráfico externo para el tráfico entre servicios**, ya que son 2 enlaces de comunicación separados.

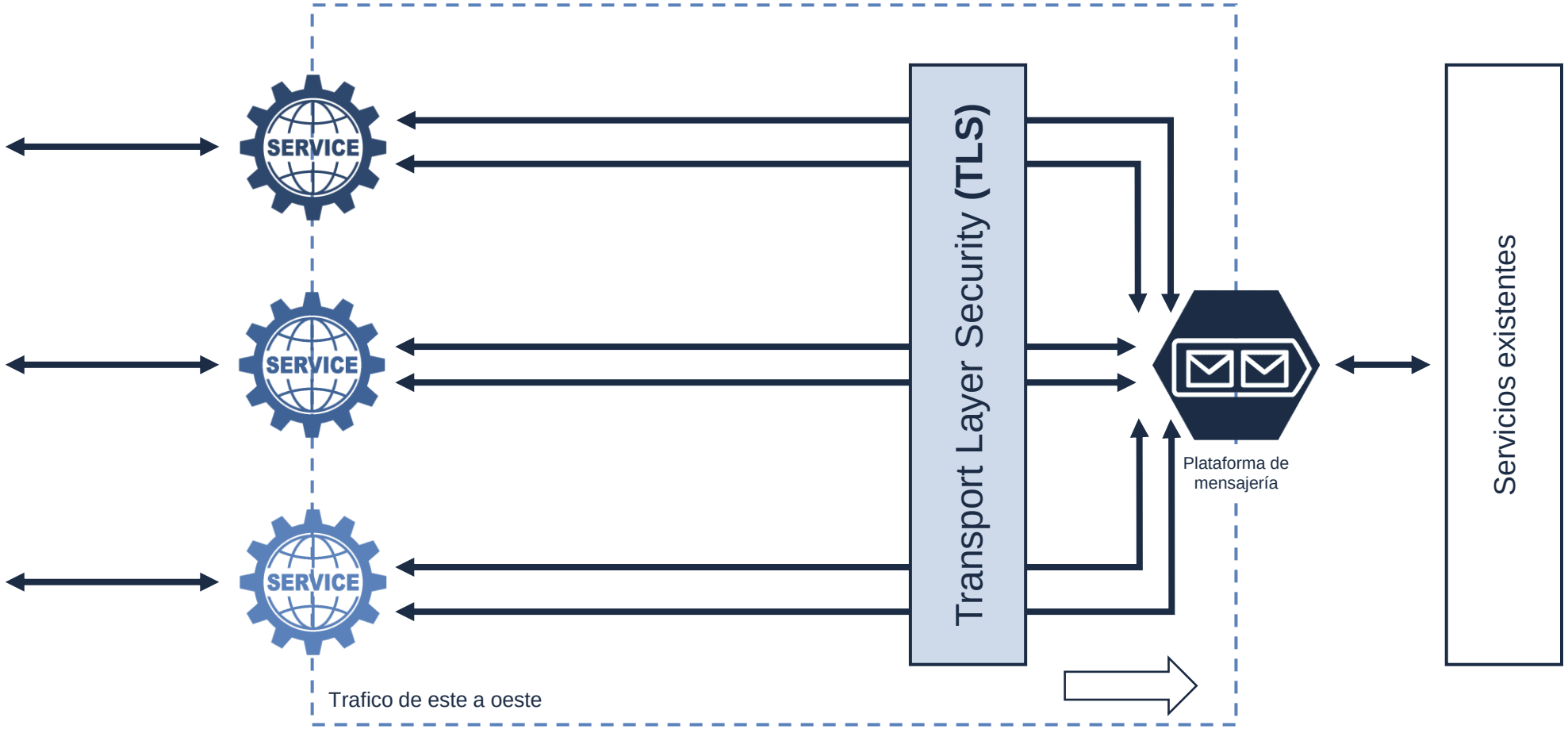
Seguridad de la capa de transporte con autenticación y autorización

El enfoque más seguro para implementar la seguridad de la comunicación entre servicios es **controlar el acceso mediante un esquema de autorización** que defina qué microservicios pueden acceder a qué otros microservicios y canales.

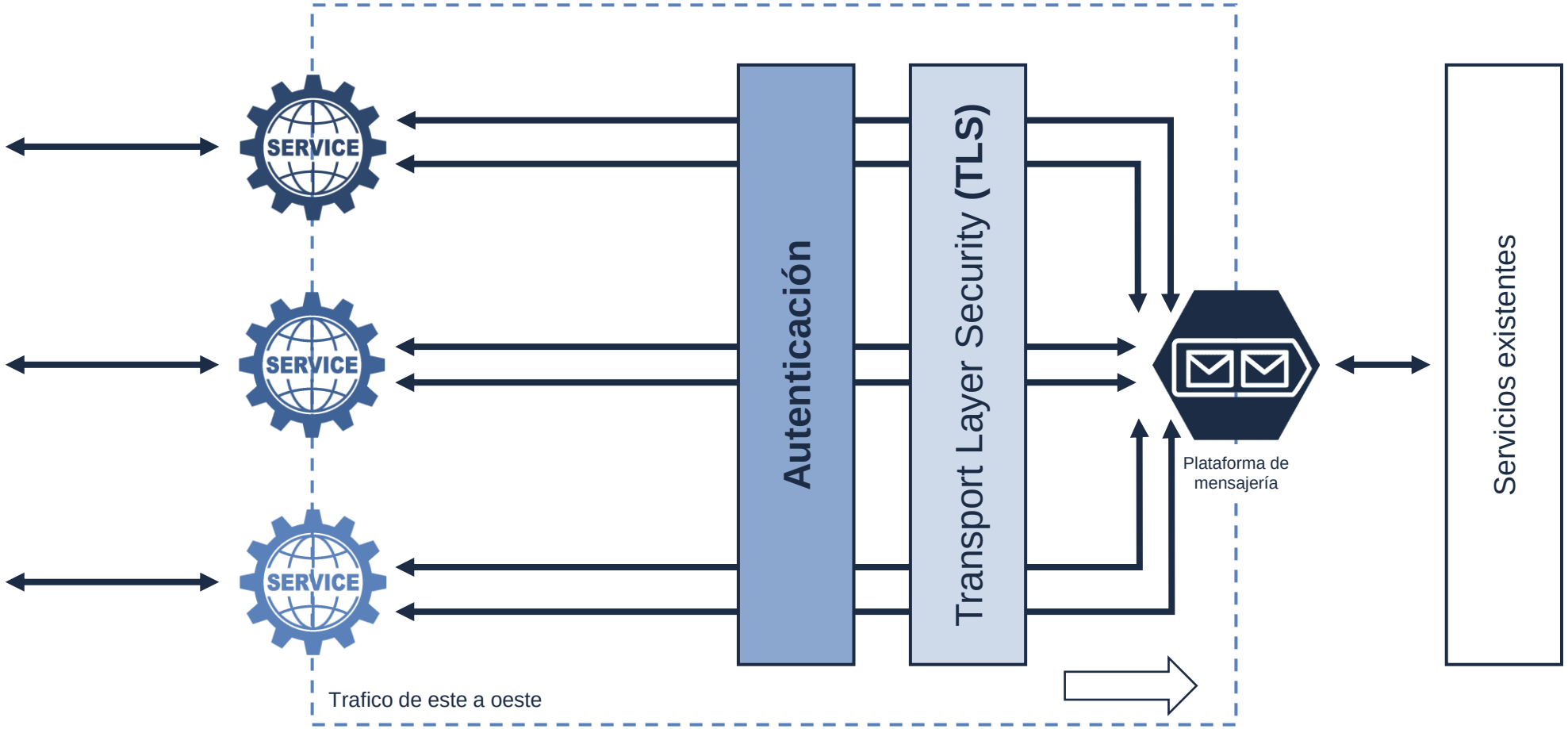
Seguridad de la capa de transporte con autenticación y autorización



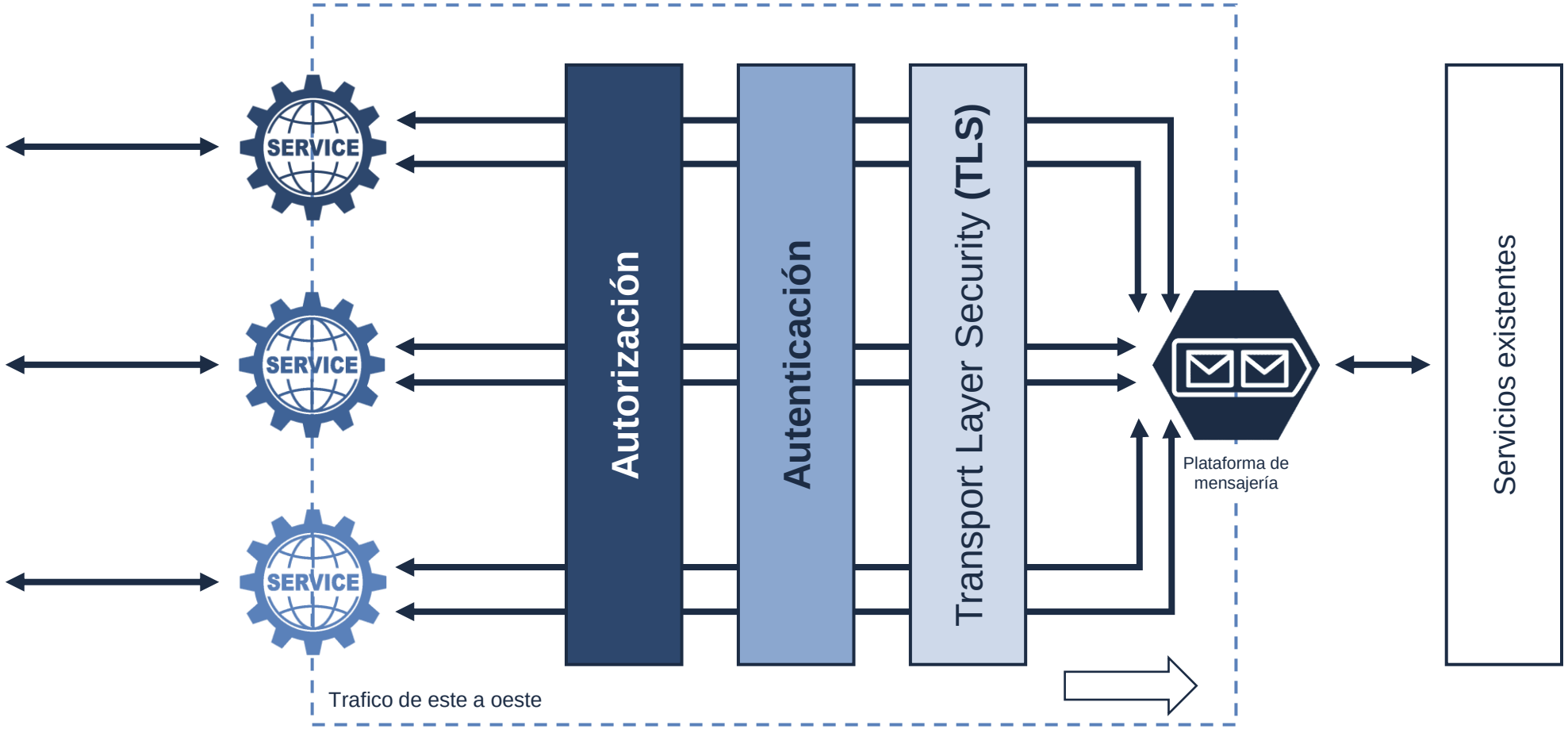
Seguridad de la capa de transporte con autenticación y autorización



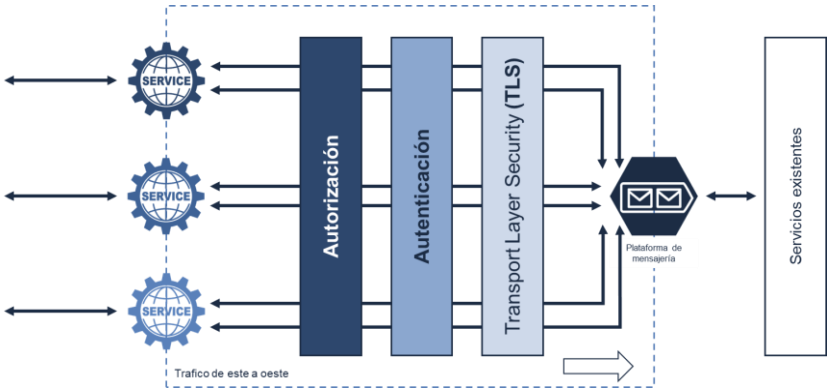
Seguridad de la capa de transporte con autenticación y autorización



Seguridad de la capa de transporte con autenticación y autorización



Seguridad de la capa de transporte con autenticación y autorización



Con este enfoque, los microservicios están equipados con las credenciales que tienen tanto la identidad del servicio como los derechos para acceder a los datos de otros servicios.

En una plataforma de mensajería como **NATS** (Neural Autonomic Transport System), tiene mecanismos para controlar el acceso mediante suscripciones usando wildcard para controlar quién puede acceder a qué.

Este es el nivel más alto de seguridad que podemos implementar para la comunicación entre servicios

02

GALAXY
TRAINING

Pilares de observabilidad (metrics, tracing y logs).

El dolor es un claro indicador de que algo no está bien. A veces las enfermedades silenciosas llegaron mucho antes que el dolor. Esa es la razón por la que visitamos a un médico de vez en cuando: para conocer nuestro estado de salud.

Pilares de observabilidad

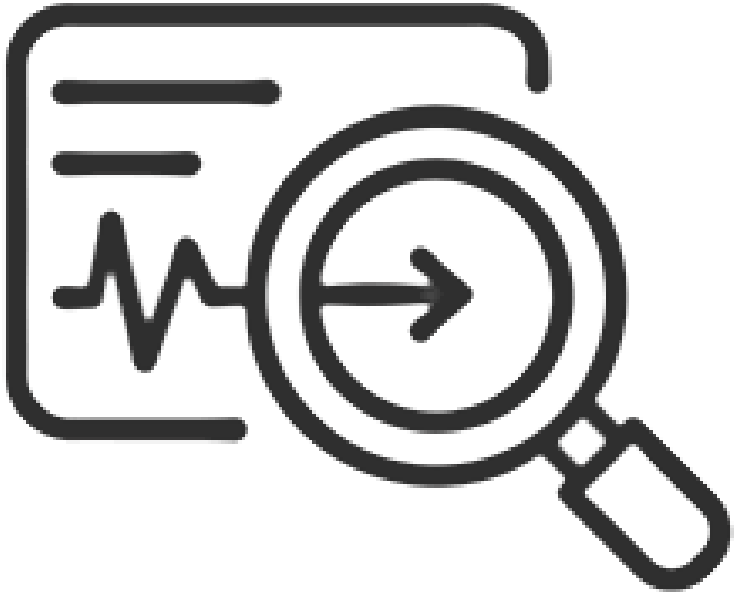
¿Cómo sabemos si nuestra aplicación es saludable?

¿Cómo podemos asegurarnos de que funciona como se espera, haciendo lo que se suponía que debía hacer y no desperdiciando recursos?

¿Cómo podemos evitar posibles incidentes antes de que ocurran?

Pilares de observabilidad

Si, adivinaste, "por monitoreo“, un concepto fundamental cuando se trata de monitoreo: Observabilidad.



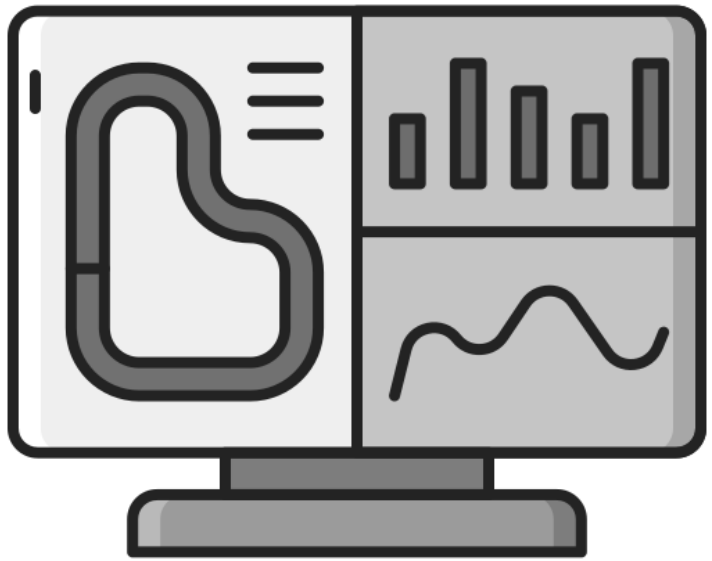
Pilares de observabilidad

¿Qué es la observabilidad?

La observabilidad es la capacidad de **conocer o inferir el estado interno de algo desde el exterior**. En otras palabras, es cómo sabemos que nuestra aplicación está funcionando bien y haciendo lo que se supone que debe hacer sin tener que mirar dentro o acceder a su código fuente.

La observabilidad es la capacidad de conocer o inferir el estado interno de algo desde el exterior.

Datos de telemetría

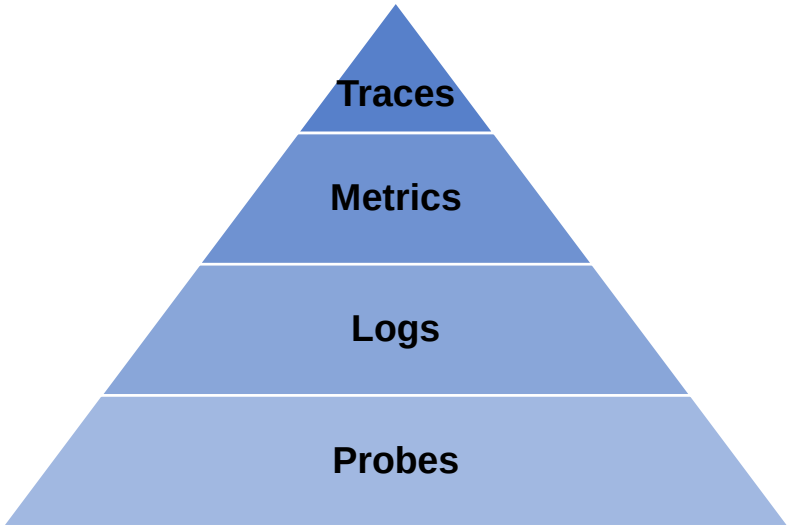


Para inferir nuestro estado de salud, los médicos confían en los datos recopilados. Escuchan los latidos de nuestro corazón, miden nuestra presión arterial, realizan análisis de sangre de laboratorio y solicitan tantas pruebas como sea necesario.

Inferir el estado de mantenimiento de una aplicación no es muy diferente. Recopilamos todos los datos que necesitamos para averiguar cómo está funcionando. **Los datos recopilados con fines de monitoreo los llamamos telemetría**.

Los datos de telemetría son los que nos permiten inferir el estado de una aplicación a distancia.

La Pirámide de Observabilidad

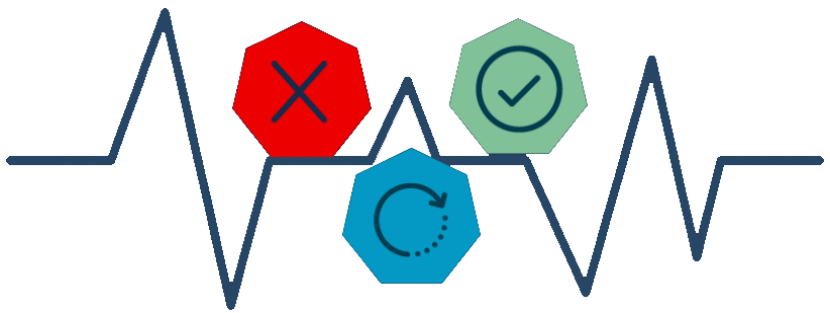


La cantidad y variedad de datos telemétricos influyen en el nivel de observabilidad final.

Si bien la recopilación de registros y métricas puede ser suficiente para los monolitos, la supervisión de sistemas distribuidos como los microservicios requiere más.

Por lo tanto, la pirámide de observabilidad comprende **probes, logs, mestrics y traces**. Cada capa piramidal aumenta el nivel de observabilidad y sirve para diferentes propósitos.

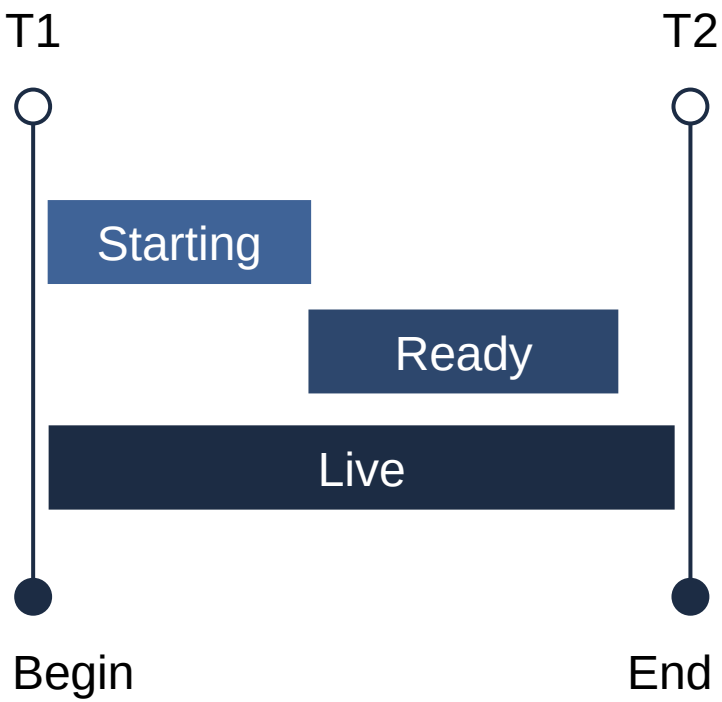
Probes (Sondas)



Sabemos si alguien está vivo, despierto y listo a partir de señales externas. Las sondas funcionan de la misma manera. **Son los indicadores de observabilidad más simples y nos dan una idea del estado del componente interno**.

Los sondeos a menudo se exponen como puntos finales y son utilizados por agentes externos como balanceadores de carga, orquestadores de contenedores, servidores de aplicaciones y administradores de sistemas antes de decidir qué hacer a continuación.

Probes (Sondas)



Las sondas más comunes son: **iniciando, listo y en vivo**.

La sonda de inicio está activada durante la fase de arranque del componente.

La sonda de preparación está activada durante el período en que el componente está listo/abierto para recibir solicitudes o procesar la carga de trabajo.

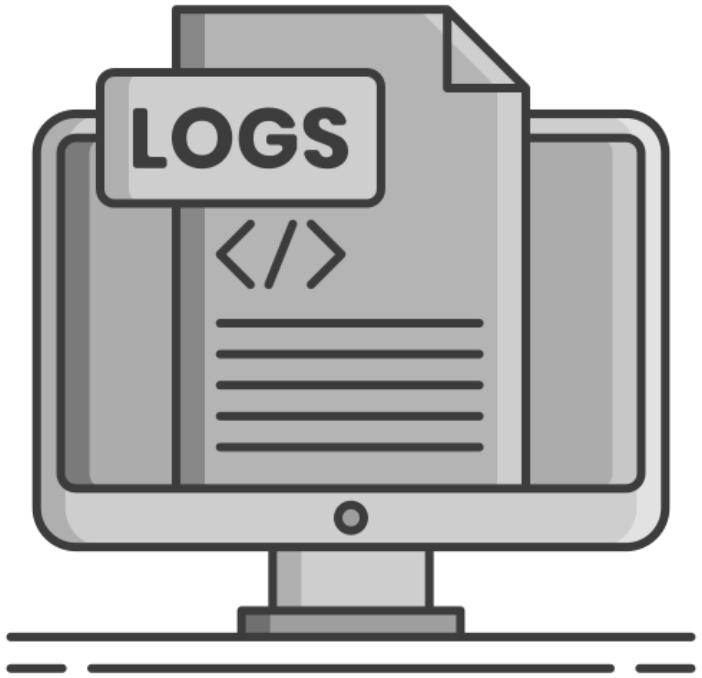
La sonda de vida está encendida desde el momento en que se inicia el componente hasta que deja de funcionar.

Pilares de observabilidad

Las sondas elevan el nivel de observabilidad, dándonos una idea del estado de un componente. Son ampliamente utilizados por monolitos, dispositivos IoT, aplicaciones por lotes y sistemas distribuidos.

Las sondas nos dan una idea del estado de un componente.

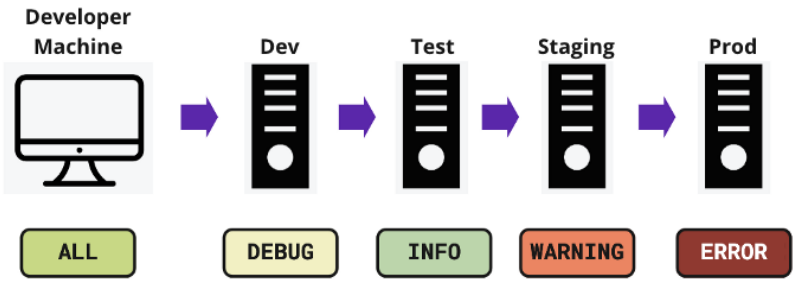
Logs



Los logs son mensajes producidos durante la ejecución del componente que capturan una pieza de información relevante. Por lo general, clasificamos los mensajes de log en **all, debug, info, warn, error, and fatal**.

Los registros pueden producir una gran cantidad de datos y contener información confidencial. Una buena práctica es aplicar diferentes niveles de registro dependiendo del entorno. De esta manera, los mensajes de registro por debajo del nivel establecido no se registran.

Logs



En entornos de producción, normalmente establecemos el nivel de registro en error o fatal.

También evitamos escribir información personal y confidencial en los registros. Cuando eso no sea posible, al menos deberíamos redactarlos. **La regla general es escribir solo lo que necesitamos para solucionar problemas**. De esa manera **no comprometemos la seguridad.**

En la máquina desarrolladora, dado que tratamos con datos falsos en una proporción manejable, somos libres de registrar todo.

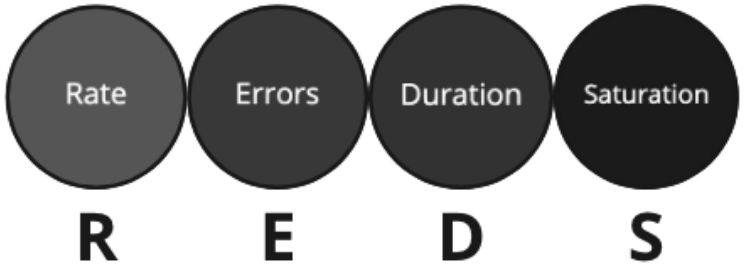
Metrics (Métricas)

Las métricas nos dan la visibilidad de los eventos buenos y malos que ocurren dentro de una aplicación. Las métricas nos ayudan a distinguir entre el comportamiento normal y anormal de la aplicación.

Por lo general, recopilamos, contamos, resumimos y comparamos eventos relacionados con **Rate** (tasa), **Errors** (errores), **Duration** (duración) y **Saturation** (saturación) para producir métricas.

El famoso acrónimo **REDS**:

Metrics (Métricas)



Rate: Tasa de un evento determinado en un momento específico. Ejemplo: Solicitudes por minuto (RPM) a las 10 pm.

Errors: Recuento de eventos de error en un intervalo de tiempo determinado. Ejemplo: Cantidad de error HTTP 5xx entre las 3 am y las 8 am.

Duration: Recuento de tiempo entre el inicio y el final de una acción específica. Ejemplo: porcentaje de solicitudes con una latencia inferior a 800 ms.

Saturation: Relación entre capacidad y consumo. Ejemplo: Componente con 530MB de 1GB disponible.

Pilares de observabilidad

Los REDS son un buen comienzo desde un punto de vista técnico. Sin embargo, podemos generar métricas a partir de cualquier otro evento contable. Por ejemplo, podemos identificar y contar el patrón "¡Fondos insuficientes!" dentro de los mensajes de log y generar una métrica de negocio a partir de eso.

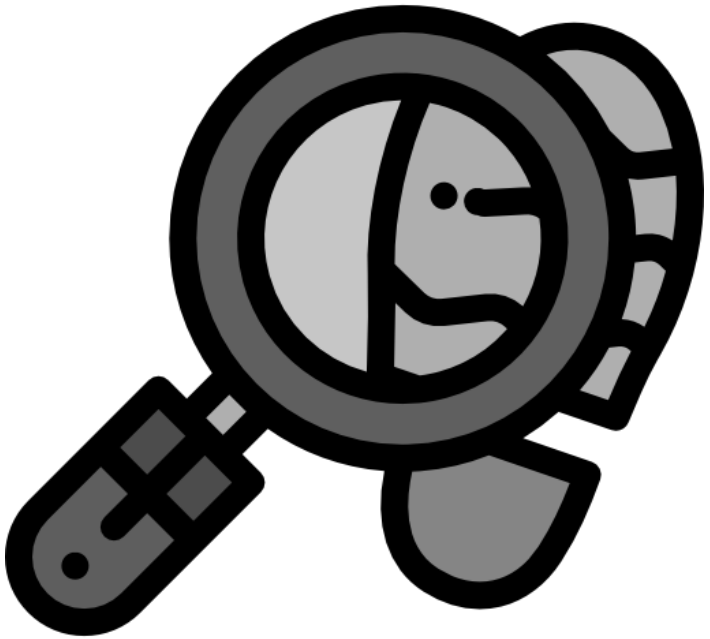
Las métricas nos dan visibilidad de los eventos buenos y malos que suceden dentro de una aplicación.

Trace (Seguimiento)

Nuestra sangre pasa a través de diferentes órganos en el sistema circulatorio. Una sola falla orgánica en este delicado sistema afecta a todo el cuerpo y puede determinar si una persona vive o muere.

Los sistemas distribuidos se componen de cientos o miles de componentes más pequeños y sus réplicas. Como ocurre en el sistema circulatorio, una falla de un solo componente puede desencadenar un evento en cascada que interrumpe todo el sistema. Por lo tanto, **los traces son imprescindibles para los microservicios**.

Trace (Seguimiento)



Los traces registran el flujo de datos entre diferentes componentes y los detalles de ejecución de cada uno a lo largo del camino.

Mediante el rastreo podemos descubrir las relaciones, las dependencias entre los componentes, identificar cuellos de botella, comprender el flujo de datos y el lapso de tiempo de cada componente.

Pilares de observabilidad

Los traces registran el flujo de datos entre los diferentes componentes y los detalles de ejecución de cada uno a lo largo del camino.

03

Monitoreo y estado de salud de los microservicios.

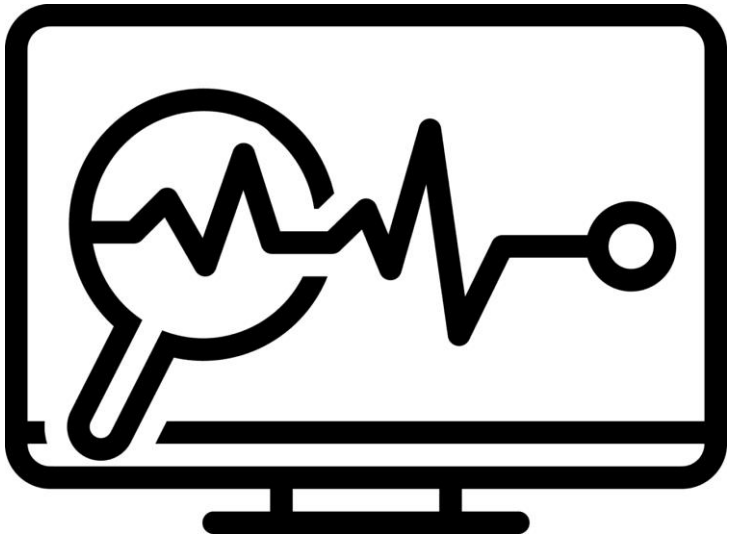
GALAXY
TRAINING

Health monitoring

El seguimiento de estado de salud puede permitir información prácticamente en **tiempo real sobre el estado de los contenedores y los microservicios**.

El seguimiento de estado de salud es fundamental para varios aspectos del funcionamiento de los microservicios y es especialmente importante cuando los orquestadores realizan actualizaciones de aplicación parcial en fases.

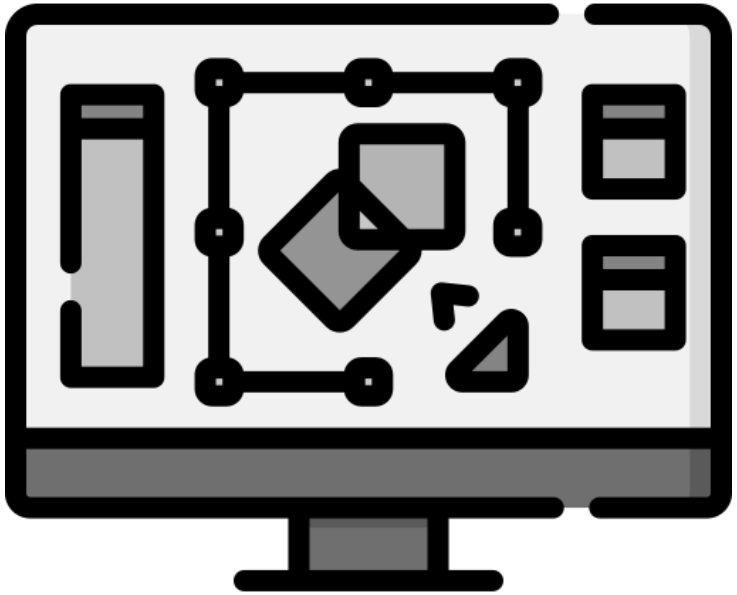
Health monitoring



Las aplicaciones basadas en microservicios suelen **usar latidos o comprobaciones** de estado para que sus monitores de rendimiento, programadores y orquestadores puedan realizar el seguimiento de gran cantidad de servicios.

Si los servicios no pueden enviar algún tipo de señal **"estoy activo"**, ya sea a petición o siguiendo una programación, la aplicación podría correr riesgos al implementar las actualizaciones, o podría simplemente detectar los errores demasiado tarde y no poder detener errores en cascada que pueden dar lugar a interrupciones importantes.

Health monitoring



En el modelo típico, **los servicios envían informes sobre su estado.** Esa información se agrega para proporcionar una visión general del estado de la aplicación.

Si se utiliza un orquestador, se puede proporcionar información de estado al clúster del orquestador a fin de que el clúster pueda actuar en consecuencia.

Si se invierte en informes de estado de alta calidad personalizados para la aplicación, **se pueden detectar y corregir mucho más fácilmente** los problemas de la aplicación que se está ejecutando.

Supervisión avanzada: visualización, análisis y alertas

La parte final de la supervisión es visualizar la secuencia de eventos, **generar informes sobre rendimiento de los servicios y emitir alertas cuando se detecta un problema**. Para este aspecto de la supervisión se pueden usar diferentes soluciones.

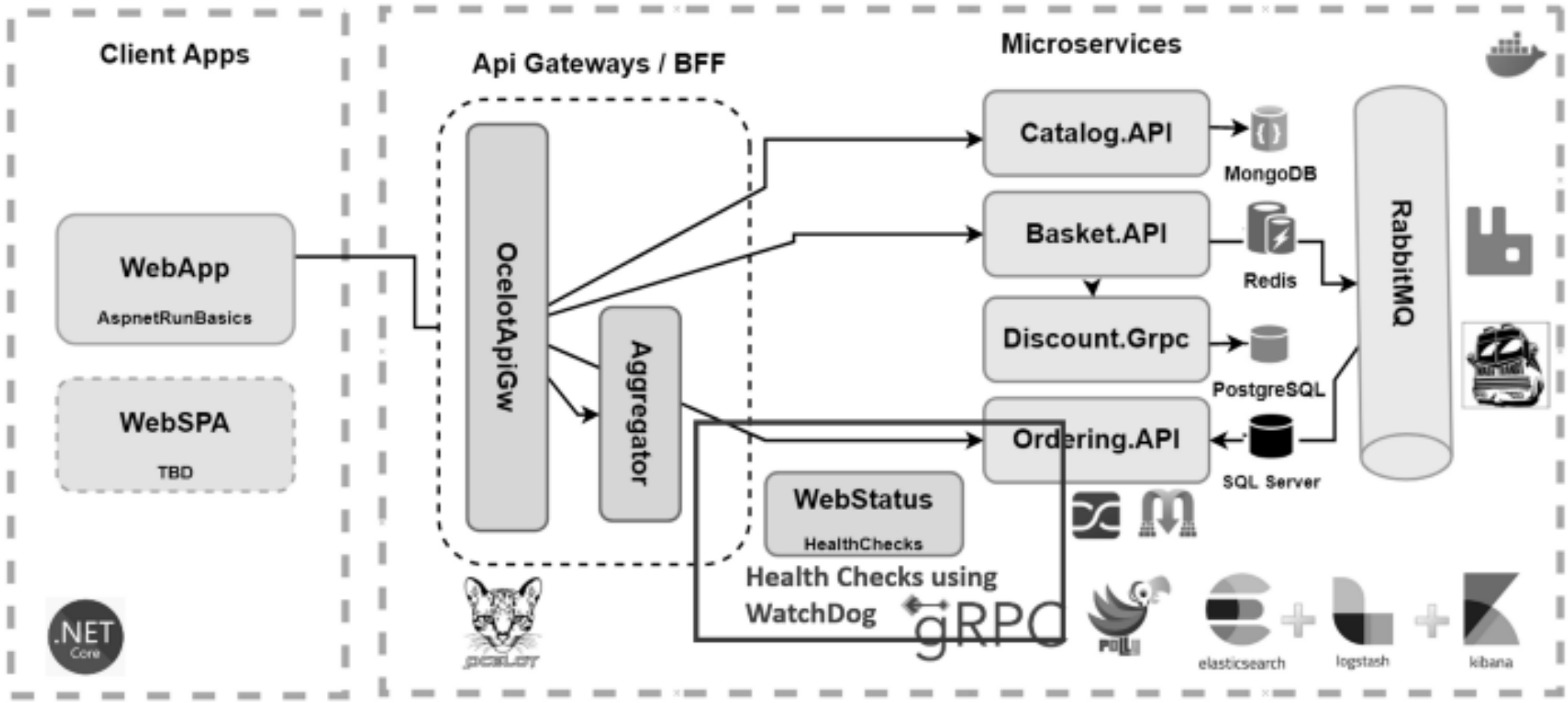
Se pueden utilizar **aplicaciones personalizadas simples** que muestren el estado de los servicios, como una página personalizada **o bien, podría usar herramientas más avanzadas como Azure Monitor** para generar alertas basadas en el flujo de eventos.

Por último, si almacena todos los flujos de eventos, se puede utilizar Microsoft Power BI u otras soluciones como Kibana o Splunk para **visualizar los datos**.

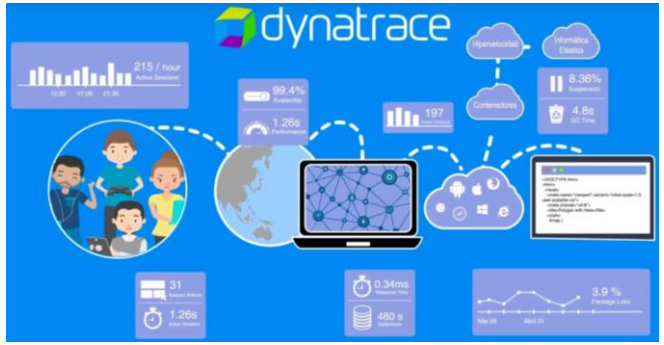
Monitoreo y estado de salud de los microservicios

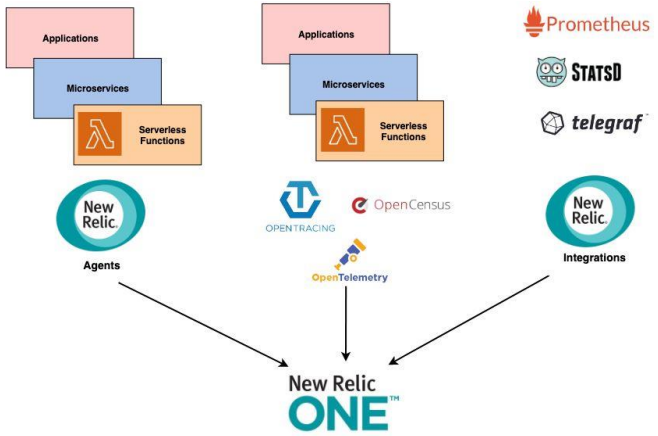
GALAXY TRAINING

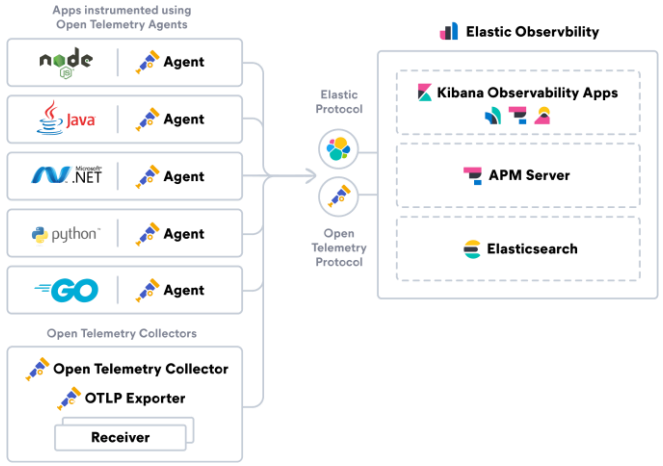
Health monitoring

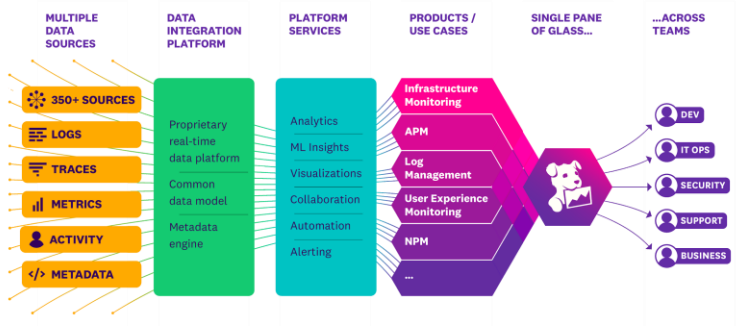


Health monitoring







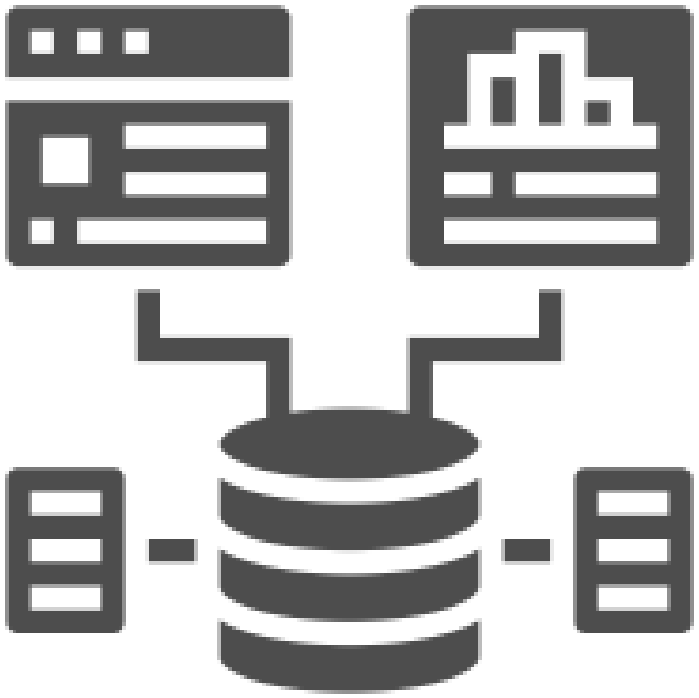


04

GALAXY
TRAINING

Centralización de logs de microservicios

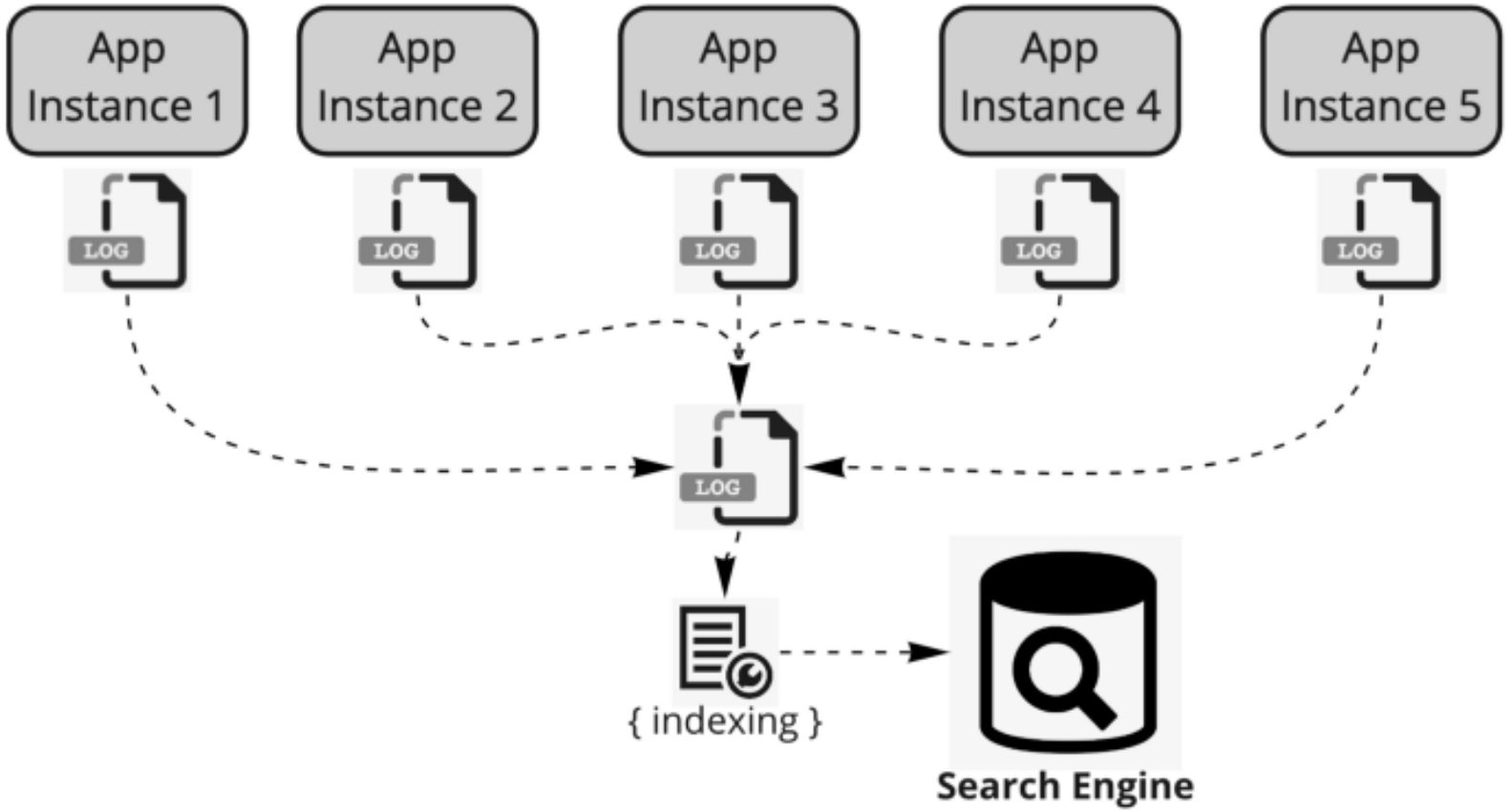
Log Aggregation



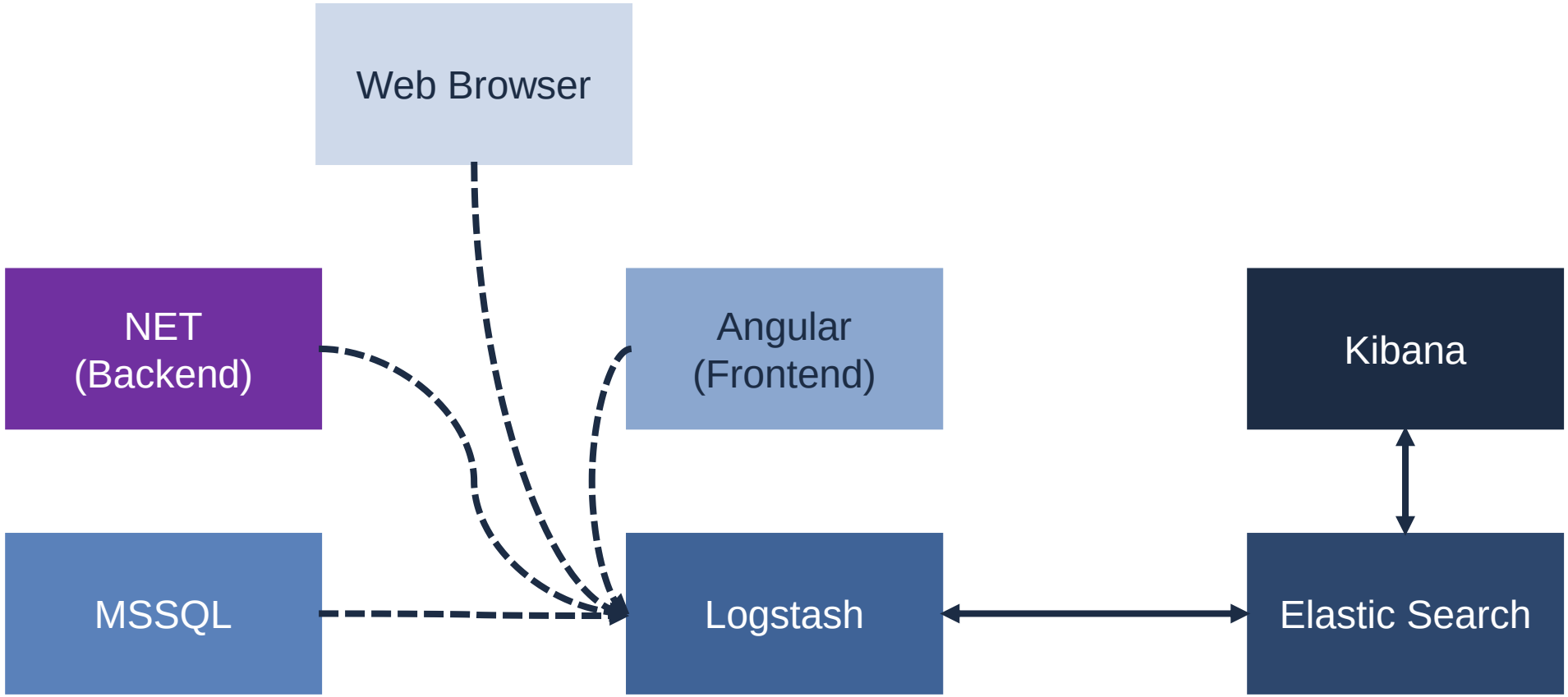
Cada componente genera uno o más archivos de log. Dado que los sistemas distribuidos se componen de múltiples componentes, es una tarea desalentadora y miserable profundizar en un montón de archivos de registro durante la solución de problemas. **Por lo tanto, la centralización de logs es muy recomendable para los microservicios.**

El Log Aggregation consiste en **consolidar e indexar los mensajes registrados en un lugar central**, normalmente un motor de búsqueda. De esa manera se vuelve más fácil y rápido buscar y analizar los mensajes.

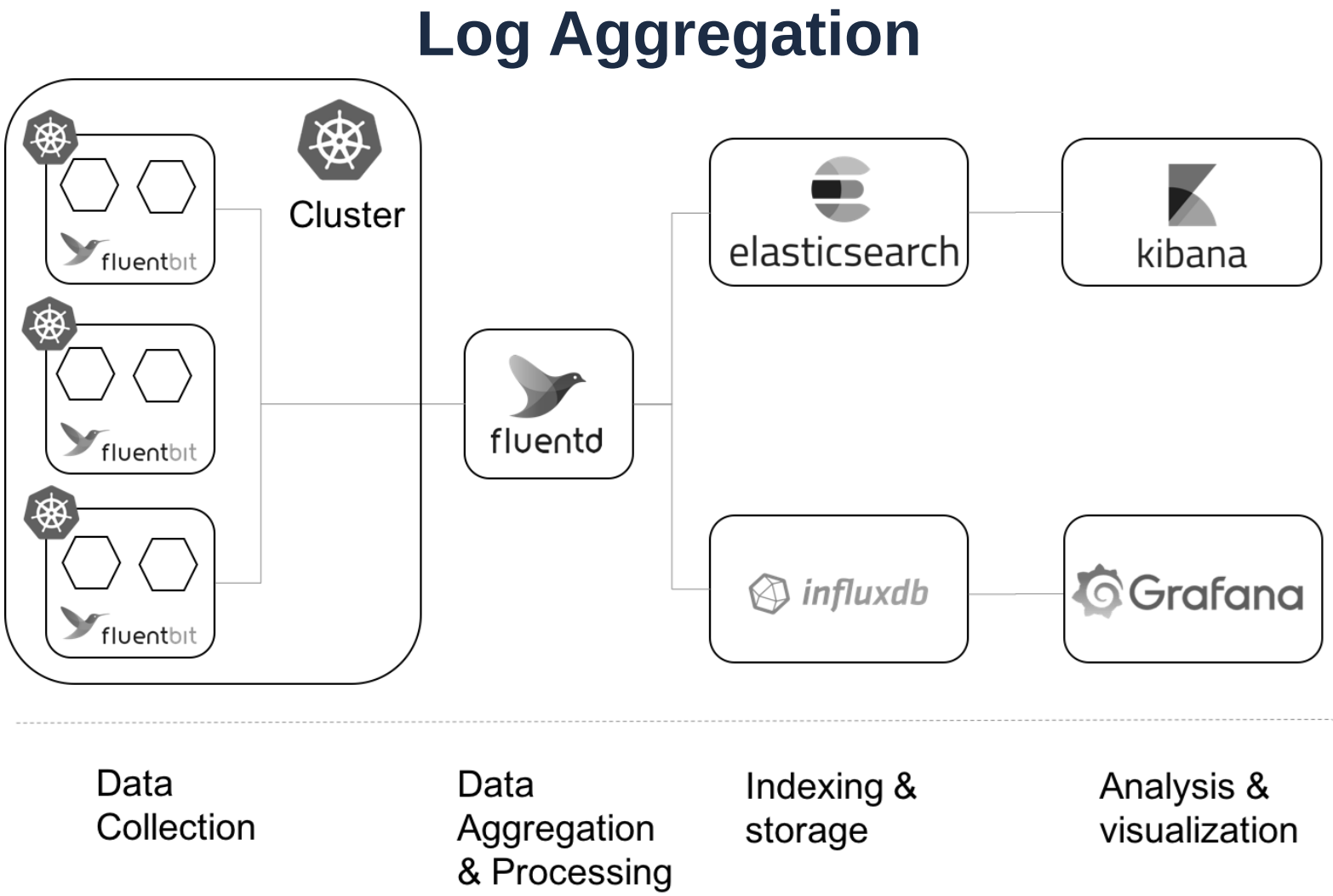
Log Aggregation



Log Aggregation

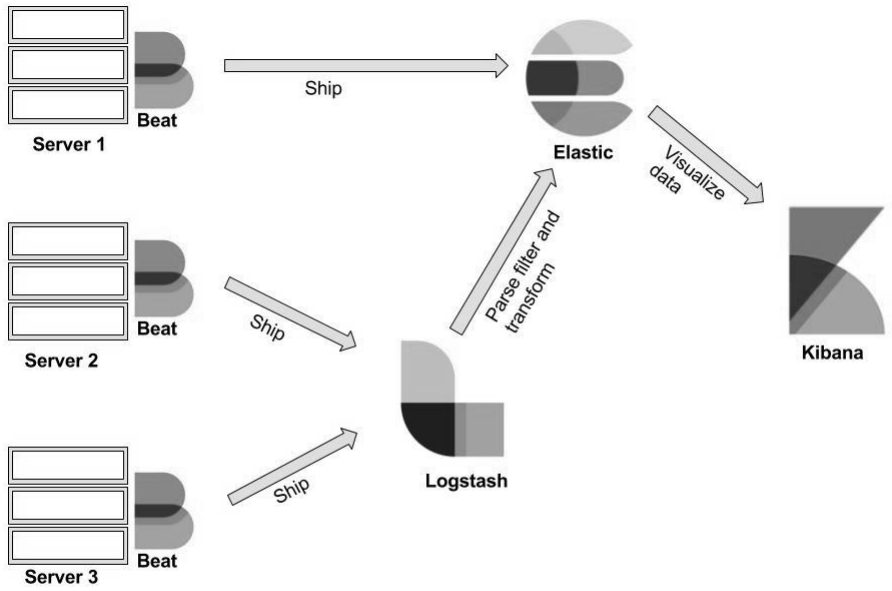


Centralización de logs de microservicios



Centralización de logs de microservicios

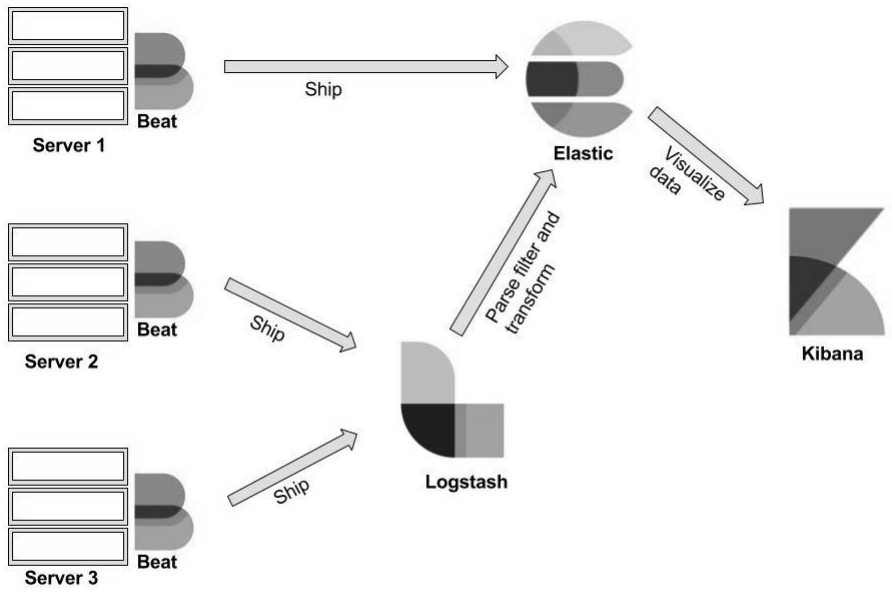
ELK



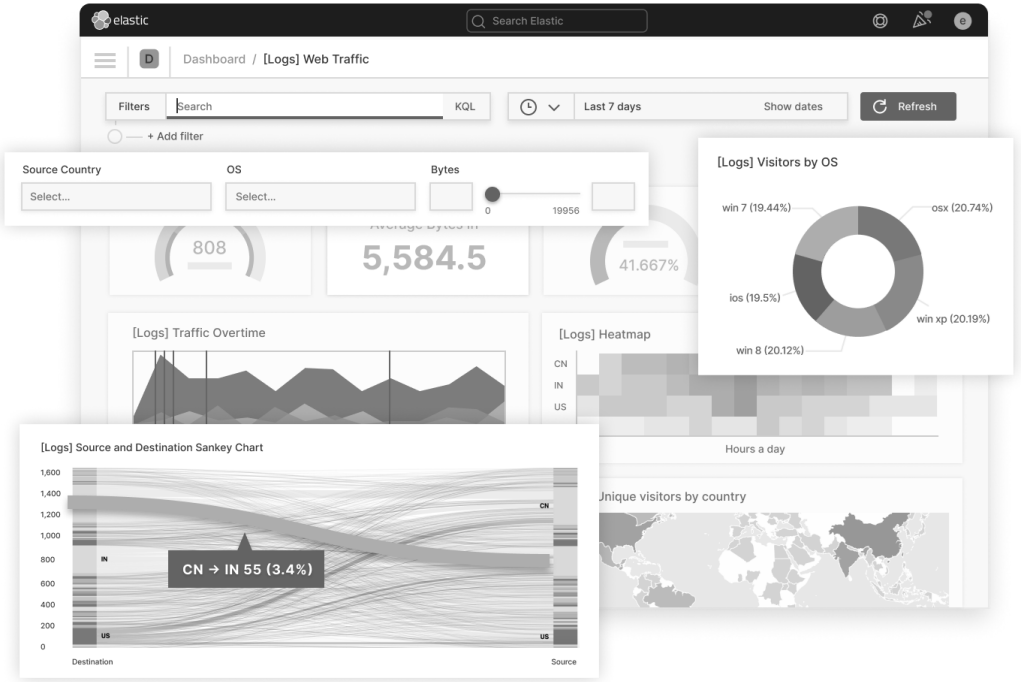
Log and System Metrics Management with Elastic Stack

Centralización de logs de microservicios

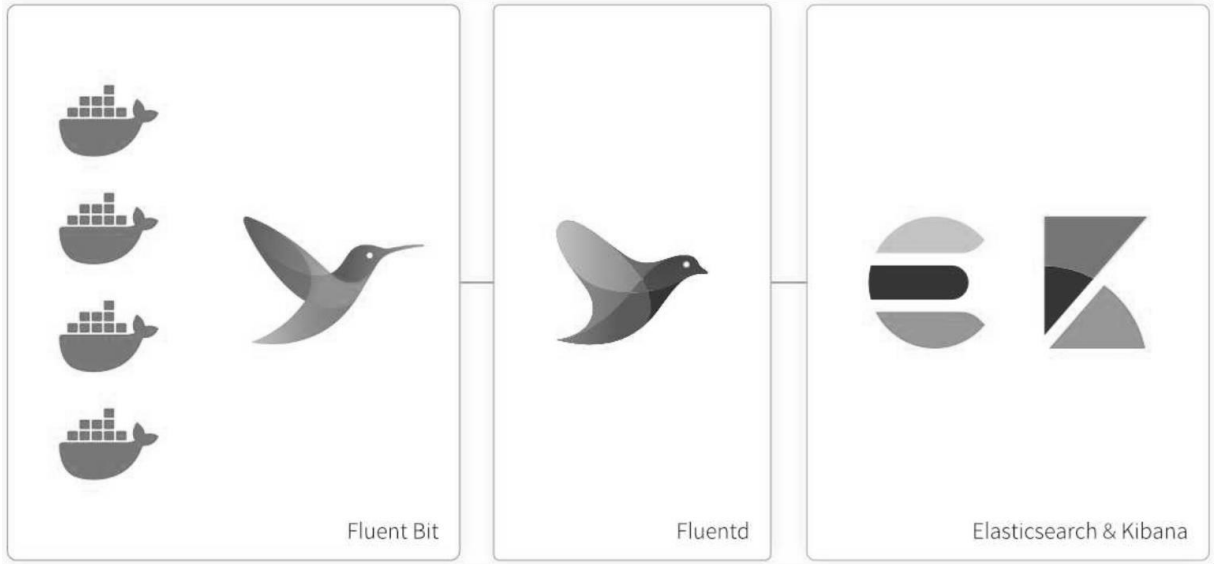
ELK



Log and System Metrics Management with Elastic Stack

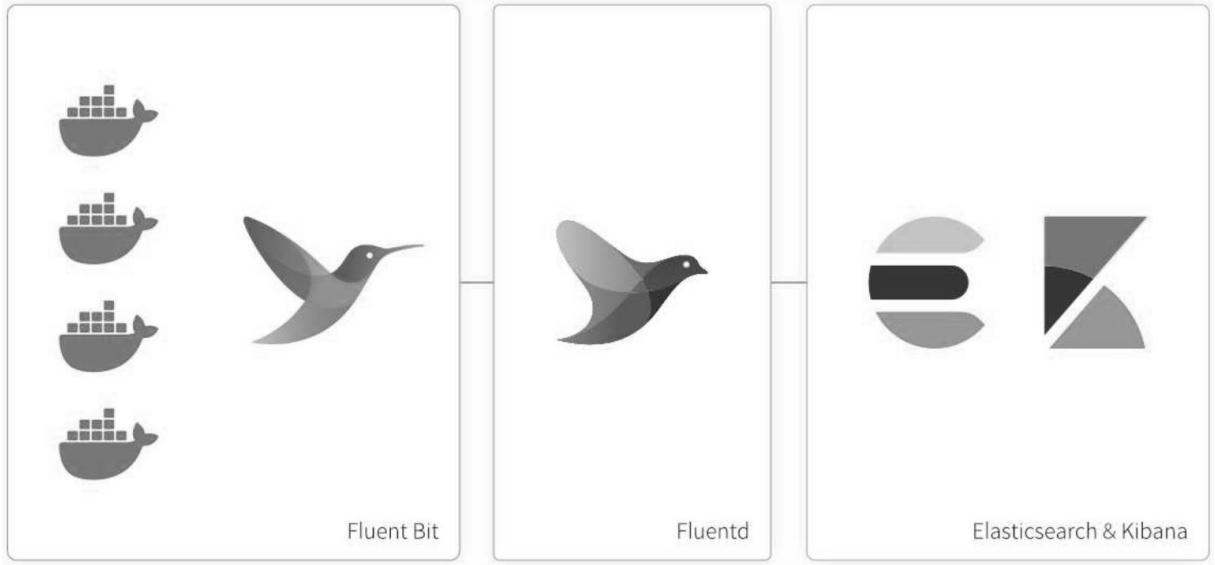


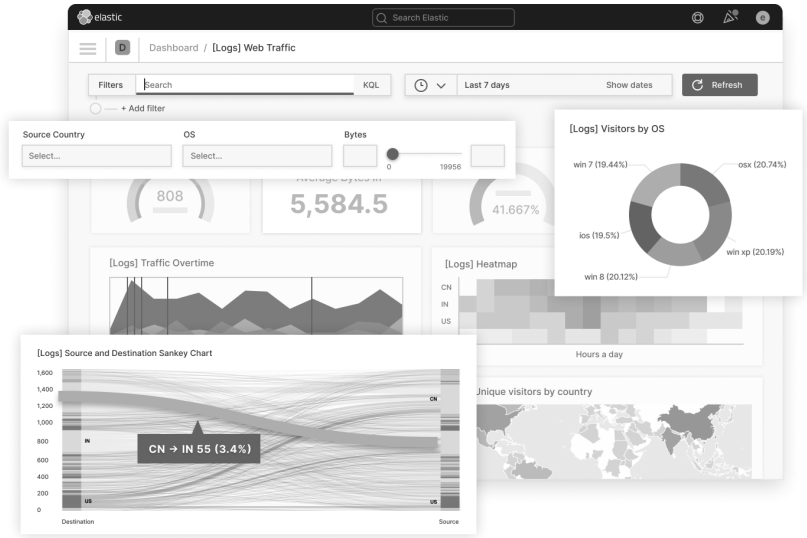
EFK



Centralización de logs de microservicios

EFK



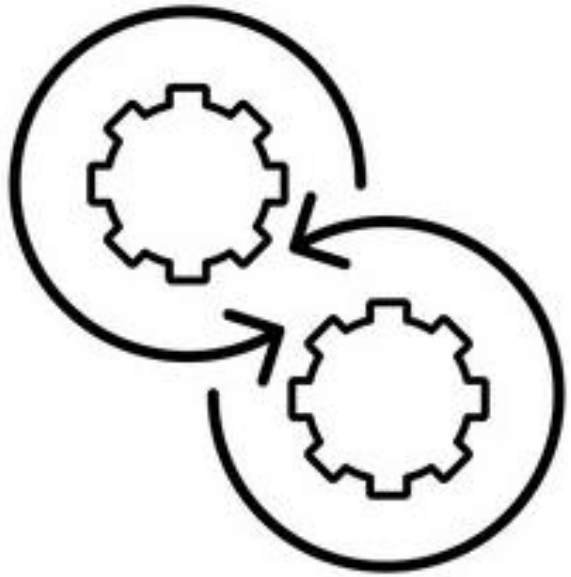


05

GALAXY
TRAINING

Trazabilidad de proceso en microservicios

Problemas comunes con los microservicios.



Pérdida de coherencia: Para cumplir con una sola solicitud del usuario final ahora se divide en múltiples procesos, posiblemente escritos en múltiples marcos y lenguajes de implementación, es mucho más difícil para los miembros del equipo entender qué sucedió exactamente en el curso del procesamiento de una solicitud.

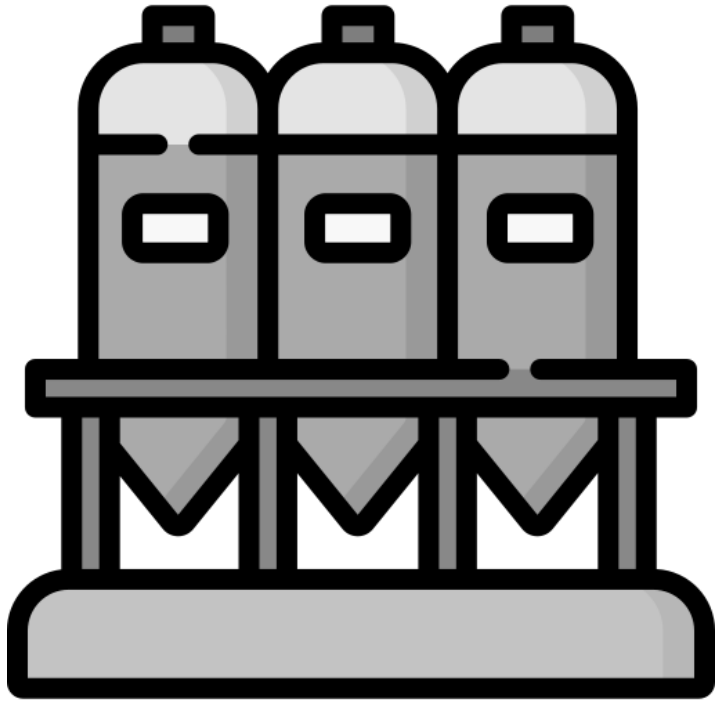
A diferencia de un proceso monolítico, donde podíamos recopilar la historia completa de cómo se manejaba una solicitud desde un solo proceso escrito en un solo lenguaje, ya no tenemos una manera fácil de hacerlo en un entorno de microservicios.

Problemas comunes con los microservicios.

Aumento de los costos de depuración y solución de problemas: El acto de rastrear y corregir las fuentes de errores dentro de las arquitecturas de microservicios puede ser tremendamente más costoso y llevar mucho tiempo.

En la mayoría de los casos, los datos de error no se propagan de manera inmediatamente útil o clara dentro de los microservicios; En lugar de un seguimiento de pila inmediatamente comprensible, tenemos que trabajar hacia atrás a partir de códigos de estado y mensajes de error vagos propagados a través de la red.

Problemas comunes con los microservicios.



Silos de datos y comunicación entre equipos: Dado que una solicitud tiene que hacer múltiples saltos a través de la red y tiene que ser manejado por múltiples.

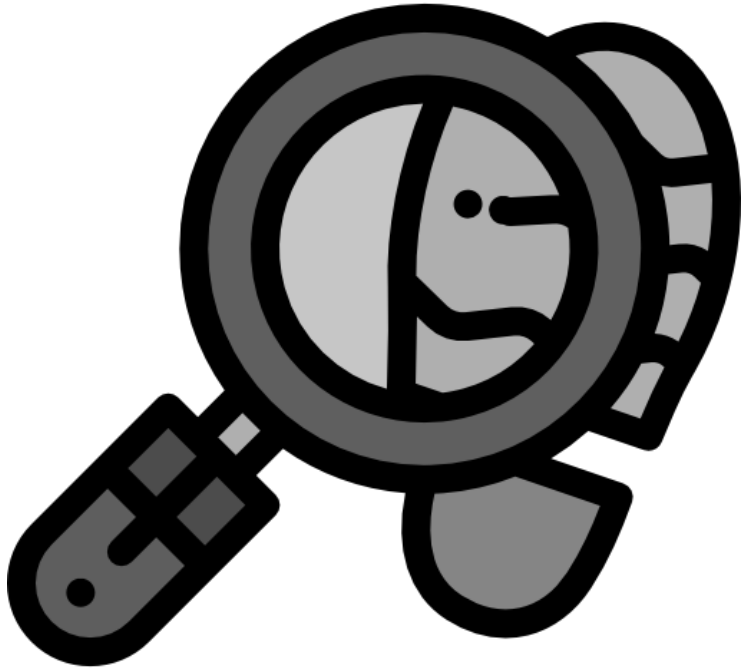
Los procesos desarrollados por equipos independientes, averiguar exactamente dónde ocurrió un error y de quién es la responsabilidad de corregirlo, pueden convertirse en un ejercicio de futilidad y frustración.

Trazabilidad de proceso en microservicios

El seguimiento distribuido es el proceso de SEGUIMIENTO Y ANÁLISIS de lo que sucede con una solicitud (transacción) en todos los servicios que toca.

¿Qué significa "rastrear" y "analizar"?

Seguimiento



Significa **generar los datos sin procesar en cada servicio** que dice: "**Hice un procesamiento para una solicitud** con un ID abc123 de rastreo: **esto es lo que hice**, **con qué otros servicios hablé** y **cuánto tiempo tomó** cada parte del trabajo".

Analizar

Significa usar cualquiera de las diversas herramientas de búsqueda, agregación, visualización y otras herramientas de análisis que lo ayudan a **dar sentido a los datos de seguimiento** sin procesar.

Trazabilidad de proceso en microservicios

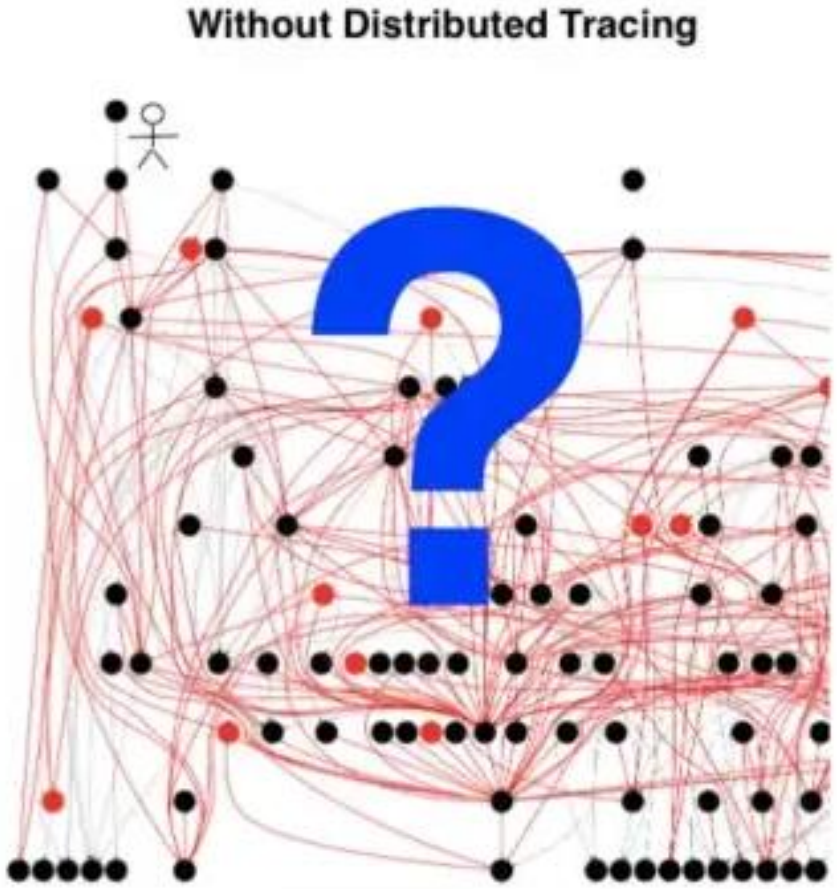
- **¿Por qué servicios pasó una solicitud?** Tanto para peticiones individuales como para la arquitectura distribuida en su conjunto (mapas de servicio).

- **¿Dónde están los cuellos de botella? ¿Cuánto tiempo tardó cada salto?** Una vez más, DT responde a esto para solicitudes individuales y ayuda a señalar patrones generales y anomalías intermitentes entre servicios en conjunto.

- **¿Cuánto tiempo se pierde debido al retraso de la red** durante la comunicación entre servicios (a diferencia del trabajo en servicio)?

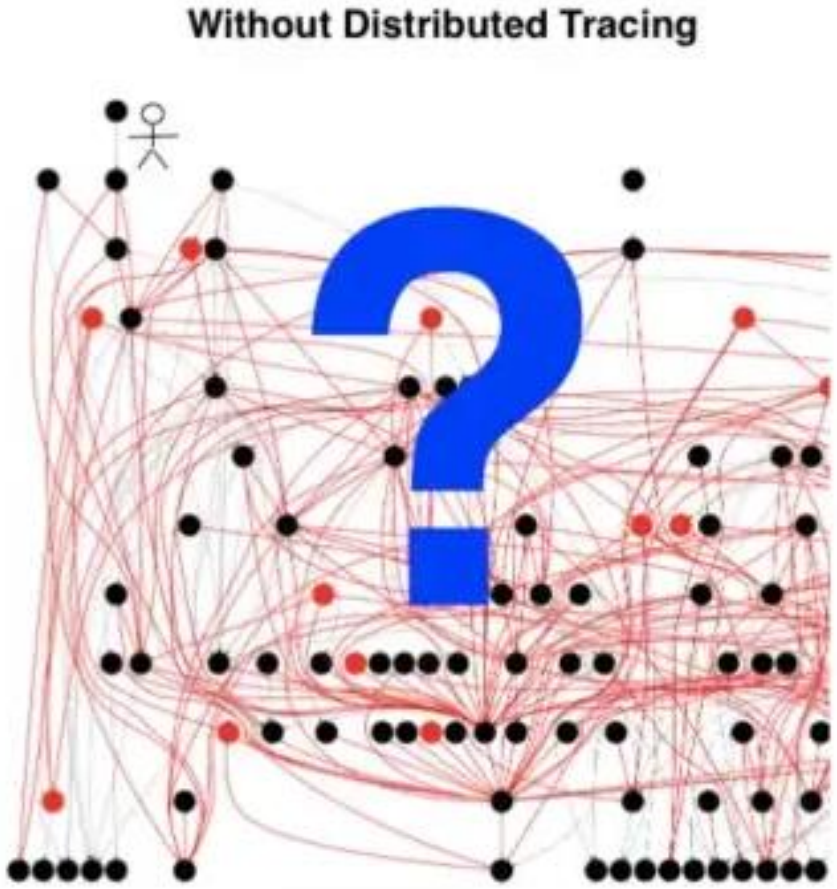
GALAXY TRAINING

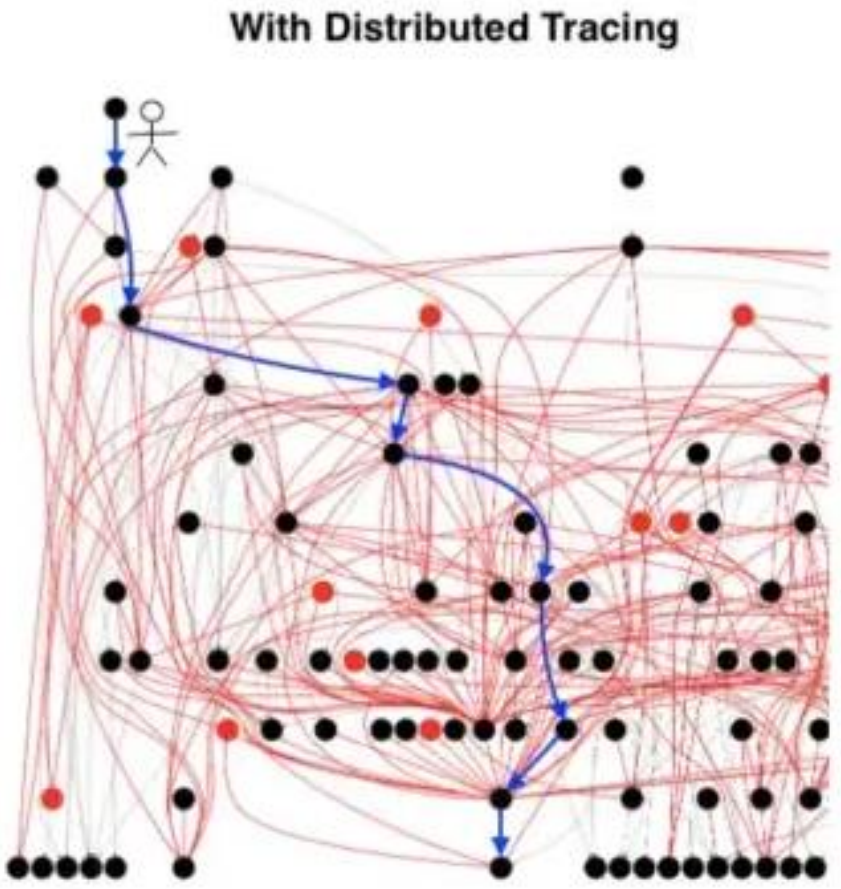
¿Qué paso con mi solicitud?



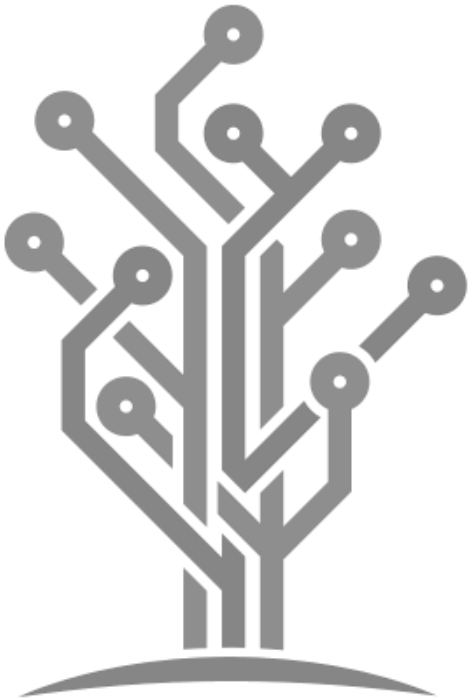
GALAXY TRAINING

¿Qué paso con mi solicitud?



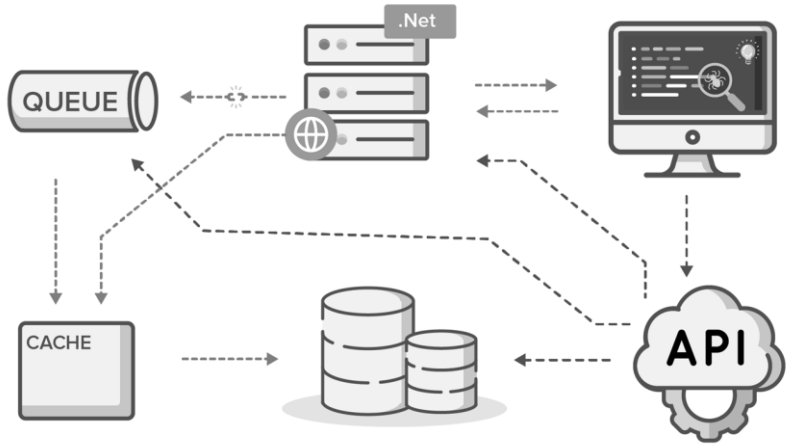


Componentes clave del sistema de rastreo



1. **Trace exposes,** la ruta de ejecución a través de un sistema distribuido. La traza se compone de uno o más tramos.
2. **Span**, en el seguimiento representa un microservicio en la ruta de ejecución.
3. **Request,** la forma en que las aplicaciones, los microservicios y las funciones se comunican entre sí.
4. **Root span**, es el primer componente de un seguimiento.
5. **Child span**, un componente posterior, que se puede anidar.

Selección de la herramienta adecuada para la implementación



Cualquiera que sea la herramienta que decida implementar debe tener algunas características básicas que se enumeran a continuación:

- En un esquema de seguimiento distribuido, debe haber algún tipo de "**recopilador de span**" que recopile los datos de span de los diversos servicios en una arquitectura de sistemas distribuidos.

- Otro componente importante que necesitamos es **una buena interfaz de usuario de visualización**. Esto será utilizado por los desarrolladores para escribir consultas complejas para depurar los problemas.

GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING