Examen Progreso 2

Juan Aristizábal

Integración de Sistemas

Facultad de Ingeniería y Ciencias Aplicadas, Universidad De Las Américas

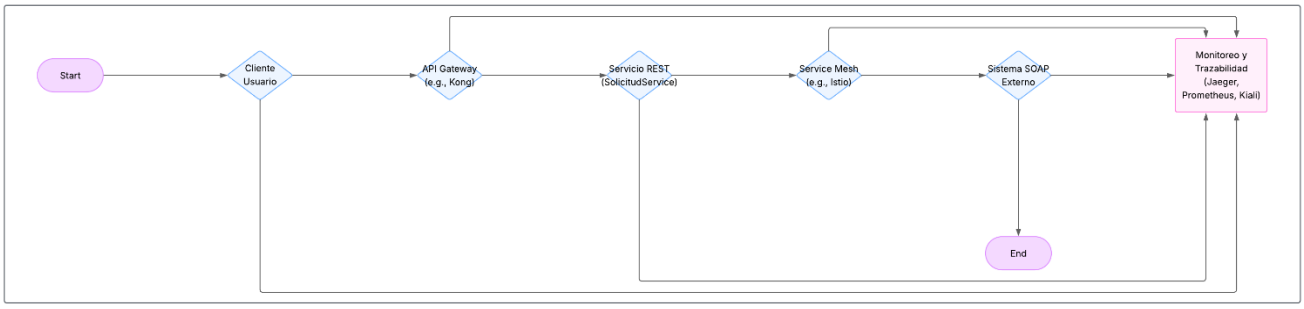
Mayo 28, 2025

Examen Progreso 2

Diseño de Arquitectura

- **Diseña un diagrama de alto nivel en el que se muestre:**
  - Los servicios involucrados.
  - El rol del API Gateway.
  - El flujo entre componentes.
  - Los puntos donde aplicarás Circuit Breaking, seguridad y trazabilidad.
- **Entregable:** Diagrama (usando la herramienta de diagramación con la que estes familiarizado) y breve descripción escrita.

Descripción



- **Servicios Involucrados:**
  - Cliente (quien hace la petición).
  - API Gateway **(Kong)** que expone el endpoint /solicitudes.
  - Servicio REST **(SolicitudService)** que procesa solicitudes y llama al sistema SOAP.
  - Sistema **SOAP** externo simulado para registrar certificaciones.
  - Sistema de seguridad y roles mediante JWT integrado en API Gateway y Servicio REST.
  - Service Mesh **(Istio)** para resiliencia: circuit breaking y retry en llamadas al SOAP
  - Herramientas de monitoreo y trazabilidad (**Jaeger** para tracing, **Prometheus** para métricas, **Kiali** para visualizar mesh)
- **Rol del API Gateway:** El API Gateway actúa como el punto único de entrada para todas las solicitudes externas que llegan a la plataforma. Su función principal es gestionar y

controlar el acceso a los microservicios internos. Entre sus responsabilidades destaca la validación y autenticación de los tokens JWT, asegurando que solo usuarios autorizados puedan interactuar con los servicios. Además, implementa políticas de rate limiting para proteger el sistema de sobrecargas o ataques de denegación de servicio. Finalmente, el API Gateway enruta las solicitudes válidas hacia el microservicio adecuado, en este caso, el Servicio REST llamado SolicitudService.

- **Flujo entre Componentes:** Cuando un cliente realiza una solicitud, envía un token JWT junto con la petición al API Gateway. Este componente valida el token y verifica que el cliente no haya excedido los límites de uso definidos por el rate limiting. Si la validación es exitosa, la solicitud es encaminada hacia el Servicio REST encargado de gestionar las solicitudes académicas, llamado **SolicitudService**. Dentro del Servicio REST, se realiza una segunda validación del JWT para reforzar la seguridad interna. Posteriormente, el servicio invoca al sistema SOAP externo, que gestiona las certificaciones académicas, para registrar la solicitud. Esta comunicación se realiza a través del **Service Mesh**, que añade una capa adicional de resiliencia mediante el manejo automático de reintentos (**retry**) y **circuit breaking** en caso de fallos. Mientras todo este proceso ocurre, las herramientas de monitoreo y trazabilidad capturan datos sobre el flujo de la solicitud, los tiempos de respuesta y posibles errores, lo que permite una mejor **observabilidad** y **diagnóstico** del sistema.
- **Puntos Clave donde se Aplican:**
    - **Seguridad:** Se implementa tanto en el **API Gateway** como en el Servicio **REST**, mediante la validación estricta de los tokens **JWT** para garantizar que sólo usuarios autenticados y autorizados puedan acceder a los servicios.
    - **Rate Limiting**: Controlado en el API Gateway, limita la cantidad de solicitudes que un cliente puede hacer en un período determinado, protegiendo así la infraestructura de posibles abusos.
    - **Circuit Breaking y Retry:** Estas políticas de resiliencia se aplican en la capa del **Service Mesh (Istio),** específicamente en la comunicación entre el Servicio REST y el sistema SOAP externo, asegurando que el sistema pueda recuperarse automáticamente de fallos temporales o evitar saturaciones.

- **Trazabilidad**: Se utiliza **Istio** junto con herramientas como Jaeger para el seguimiento distribuido, **Prometheus** para la recolección de métricas y **Kiali** para la visualización, facilitando el monitoreo en tiempo real y la detección rápida de problemas.

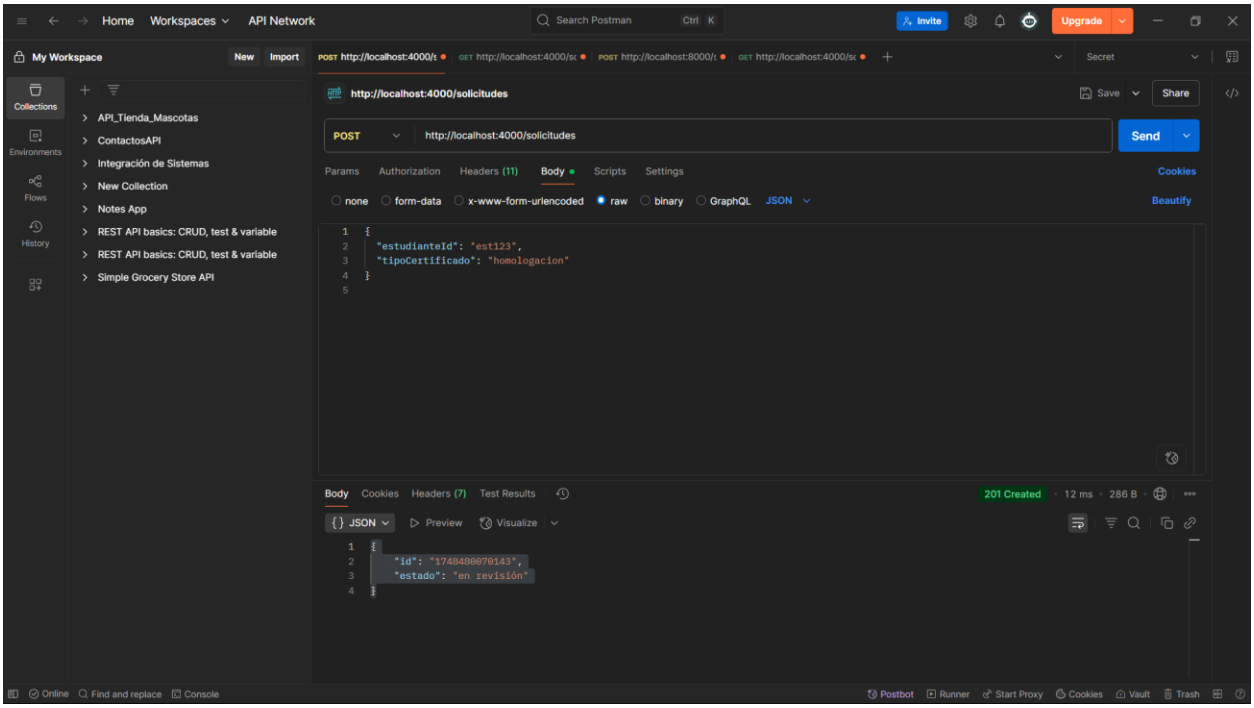
Diseño de Arquitectura

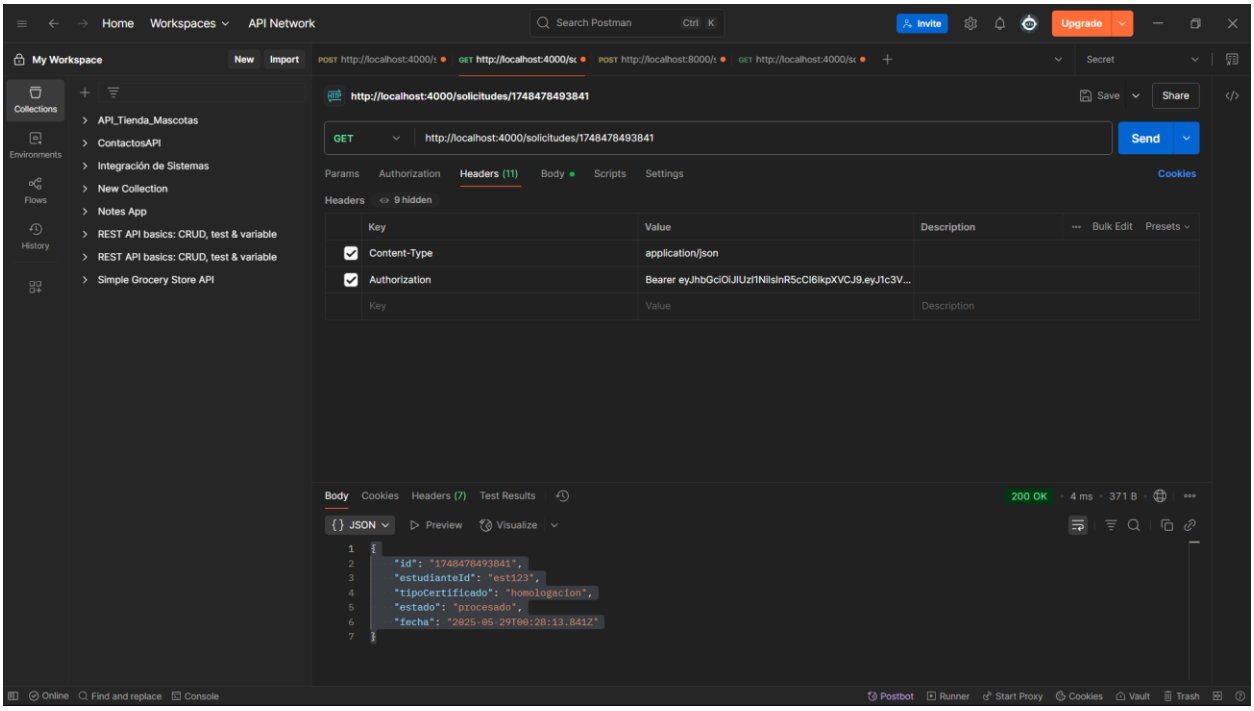
- **Implementa un microservicio REST usando la tecnología a tu elección llamado SolicitudService, con los siguientes endpoints:**
  - POST /solicitudes
  - GET /solicitudes/{id}
- **Este servicio debe:**
  - Validar el JWT recibido en la cabecera (Authorization: Bearer <token>).
  - Llamar al sistema SOAP externo para registrar la certificación, puedes usar un mock del servicio SOAP con una herramienta como SoapUI para simplemente simularlo.
  - Retornar el estado final de la solicitud (procesado, en revisión, rechazado).
- **Entregable:** Link al repositorio del código fuente funcional y archivo README con instrucciones claramente especificadas.

Repositorio

https://github.com/Aristox173/ExamenProgreso2

Respuestas de Postman





Exposición del servicio a través del API Gateway

- Usa una herramienta como WSO2 API Manager, Kong Gateway, o la de tu preferencia, o a su vez un mock si el entorno no lo permite.
- Registra el endpoint /solicitudes y aplica:
    - Una política de seguridad por token (API Key o JWT).
    - Una política de rate limiting.
- **Entregable:** capturas de configuración o archivo exportado del gateway.

Crear un servicio en Kong para el microservicio REST

Descripción: Registra un servicio llamado solicitud-service en Kong, apuntando a la URL local del microservicio REST (puerto 4000). Esto permite que Kong conozca el backend al que se debe enrutar el tráfico.

Crear una ruta para el servicio

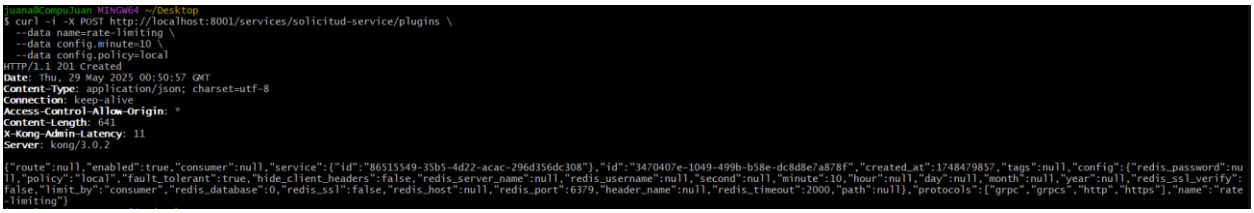
Descripción: Define una ruta asociada al servicio anterior que expone el endpoint /solicitudes. Así, cuando se haga una petición a /solicitudes en Kong, esta se redirige al microservicio registrado.

Agregar plugin de validación JWT al servicio

Descripción: Aplica el plugin JWT para que Kong valide los tokens JWT en las solicitudes dirigidas a solicitud-service, asegurando que solo usuarios autenticados puedan acceder.

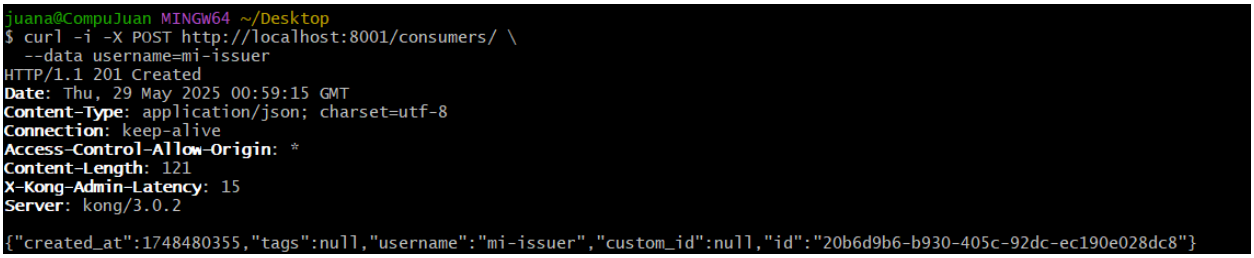
Agregar plugin de rate limiting al servicio

Descripción: Aplica una política de limitación de solicitudes (rate limiting) que permite hasta 10 solicitudes por minuto por consumidor, protegiendo el servicio de abusos o picos de tráfico.



Crear un consumidor (cliente autorizado) en Kong

Descripción: Registra un consumidor llamado mi-issuer que representa a un usuario o aplicación cliente autorizada para consumir el servicio mediante JWT.



Asociar credenciales JWT al consumidor

Descripción: Vincula una clave (key) y un secreto (secret) JWT al consumidor, para que Kong pueda validar correctamente los tokens firmados por ese emisor.

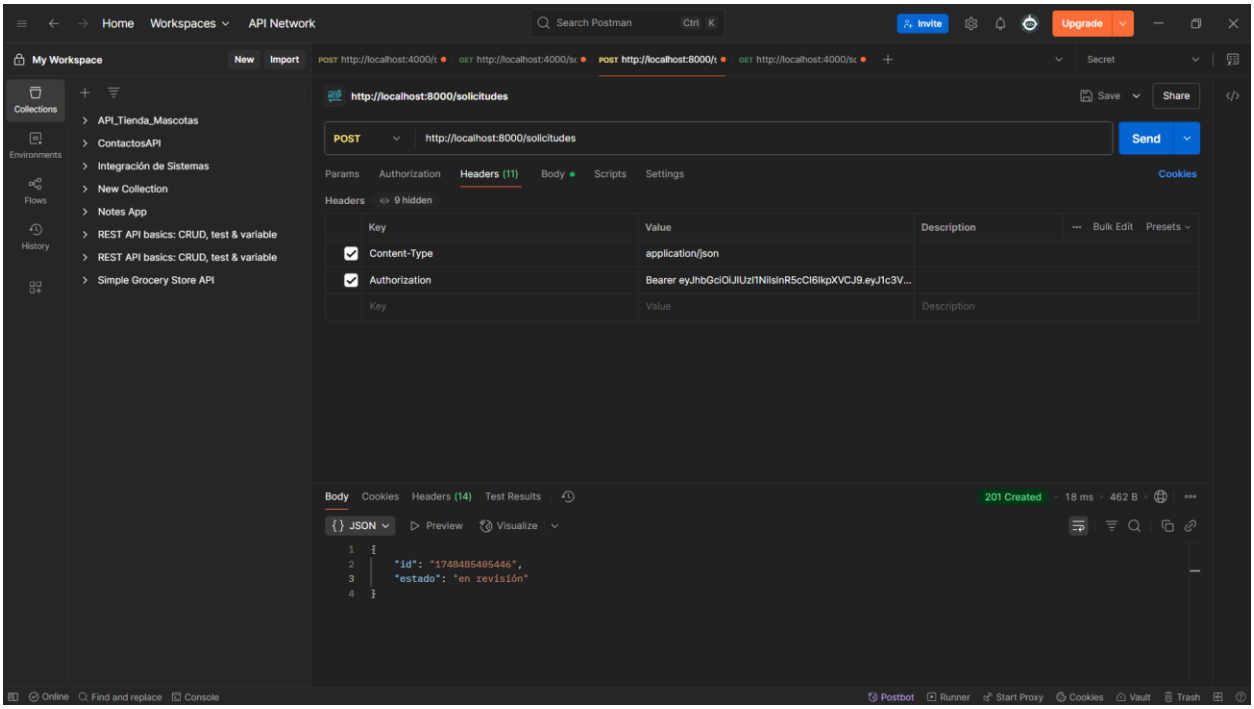
Consultar las rutas configuradas en Kong

Descripción: Obtiene un listado de todas las rutas configuradas en Kong, permitiendo verificar que /solicitudes esté correctamente creada y asociada al servicio.

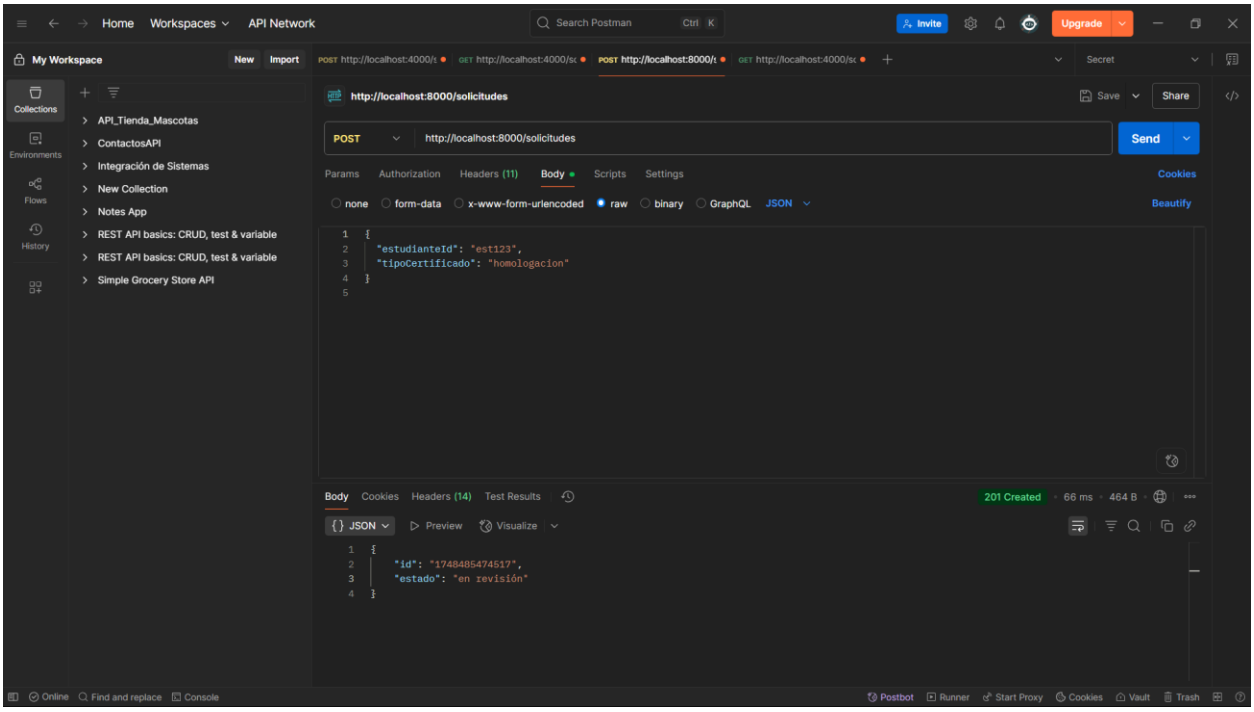
Actualizar la configuración de la ruta para no eliminar el path

Descripción: Cambia la configuración para que Kong no elimine el prefijo /solicitudes al reenviar la petición al servicio backend. Esto asegura que el servicio reciba el path completo.

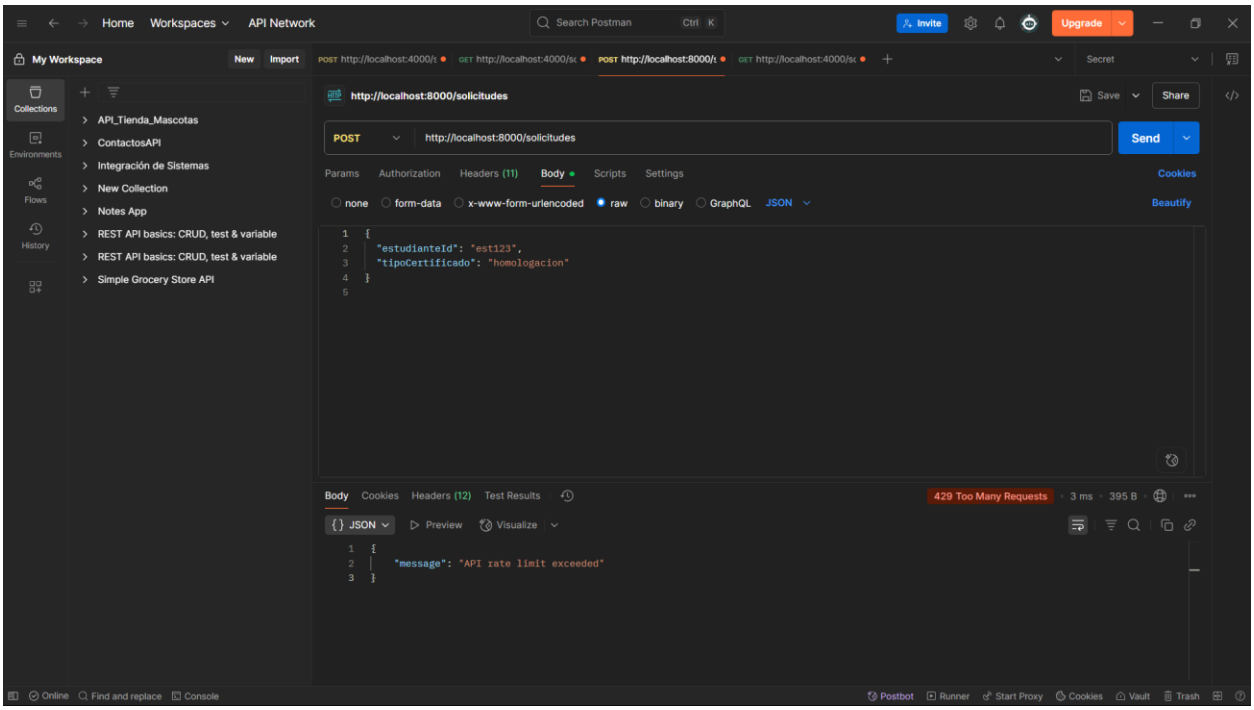
Uso de Token



Petición



Rate Limiting



Implementación de Circuit Breaking y Retry

- **Define una configuración (real o pseudocódigo YAML) para aplicar:**
  - Retry automático al servicio SOAP (máximo 2 intentos).
  - Circuit Breaker si hay más de 3 fallos en 60 segundos.
- Puedes usar herramientas como Istio o Spring Cloud Gateway
- **Entregable:** archivo YAML o fragmento de código con explicación.

Archivo YAML

destinationrule-cb-retry.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: soap-service-destination
  namespace: default
spec:
  host: soap-service
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        http1MaxPendingRequests: 100
        maxRequestsPerConnection: 100
    outlierDetection:
      consecutive5xxErrors: 3 # Circuit Breaker: 3 errores HTTP 5xx consecutivos
```

```
    interval: 60s # Ventana de 60 segundos
    baseEjectionTime: 30s # Tiempo que el host queda excluido del pool
    maxEjectionPercent: 100 # Puede excluir hasta el 100% de hosts
```

virtualservice-retry.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: soap-service-vs
  namespace: default
spec:
  hosts:
    - soap-service
  http:
    - route:
        - destination:
            host: soap-service
            port:
              number: 80
      retries:
        attempts: 2 # Máximo 2 intentos de retry
        perTryTimeout: 2s # Timeout de 2 segundos por intento
        retryOn: gateway-error,connect-failure,refused-stream
```

Explicación

- **Circuit Breaking:** La configuración en DestinationRule detecta si el servicio SOAP responde con 3 errores HTTP 5xx consecutivos en un intervalo de 60 segundos, y automáticamente "expulsa" (eject) el host durante 30 segundos para evitar seguir enviando tráfico a un servicio inestable. Esto protege al sistema de fallos prolongados o cascadas de errores.
- **Retry Automático:** La configuración en VirtualService permite que Istio reintente automáticamente la llamada al servicio SOAP hasta 2 veces, con un timeout de 2 segundos por intento, cuando detecta ciertos errores (fallos de gateway, conexión o streams rechazados). Esto aumenta la resiliencia frente a fallos temporales.

Uso

kubectl apply -f destinationrule-cb-retry.yaml

kubectl apply -f virtualservice-retry.yaml

Monitoreo y Trazabilidad

- **Explica brevemente (en texto) cómo implementarías monitoreo en esta arquitectura.**
    - ¿Qué herramientas utilizarías?
    - ¿Qué métricas y trazas capturarías?
- **Entregable:** documento breve explicativo.

Para implementar monitoreo y trazabilidad en esta arquitectura que involucra un microservicio REST (SolicitudService), un sistema SOAP externo, un API Gateway (Kong) y un Service Mesh (Istio), es fundamental contar con herramientas que permitan visualizar y analizar el comportamiento de los componentes en tiempo real. Como herramientas principales, usaría Prometheus para la recolección de métricas, ya que es ampliamente utilizado en entornos Kubernetes y Service Mesh. Prometheus puede extraer métricas de latencia, tasas de error y uso de recursos de los pods y servicios. Para la visualización de estas métricas, integraría Grafana, que

permite crear dashboards intuitivos y personalizados para monitorear el estado y la performance del sistema. Para la trazabilidad distribuida, implementaría Jaeger, que captura las trazas de las solicitudes que atraviesan los diferentes componentes, desde el API Gateway, pasando por el microservicio REST, hasta la llamada al sistema SOAP. Esto facilita identificar cuellos de botella, fallos y retrasos específicos en la cadena de llamadas, ayudando a diagnosticar problemas con mayor rapidez. Además, usaría Kiali para la observabilidad del Service Mesh de Istio. Kiali proporciona una representación visual de la topología de los servicios y sus interacciones, además de mostrar métricas específicas del mesh, como el estado de los circuit breakers, reintentos y latencias entre servicios. Esto es útil para monitorear la salud de la red de servicios y la efectividad de políticas como circuit breaking y retry. Las métricas que capturaría incluyen tiempos de respuesta, tasas de éxito y error (especialmente HTTP 5xx), número de solicitudes, uso de CPU y memoria, y estado de los circuitos (abiertos o cerrados). También es importante capturar datos de validación de seguridad, como rechazos por tokens inválidos en el API Gateway, y las limitaciones aplicadas por políticas de rate limiting.