

Progreso 2 – Integración de Sistemas
Caso Plataforma de Servicios Estudiantiles

Martín Vargas

Repositorio Github: https://github.com/MartinVargas07/Examen_IDS_P2_Vargas.git

La universidad desea construir una nueva plataforma integrada para atender solicitudes académicas de los estudiantes (solicitudes de certificados, legalizaciones, homologaciones y equivalencias). Actualmente, existen 3 sistemas independientes:

1. Sistema Académico (REST API – gestionado internamente)
2. Sistema de Certificación (SOAP – externo, expuesto por un proveedor estatal)
3. Sistema de Seguridad y Roles (interno, usa tokens JWT)

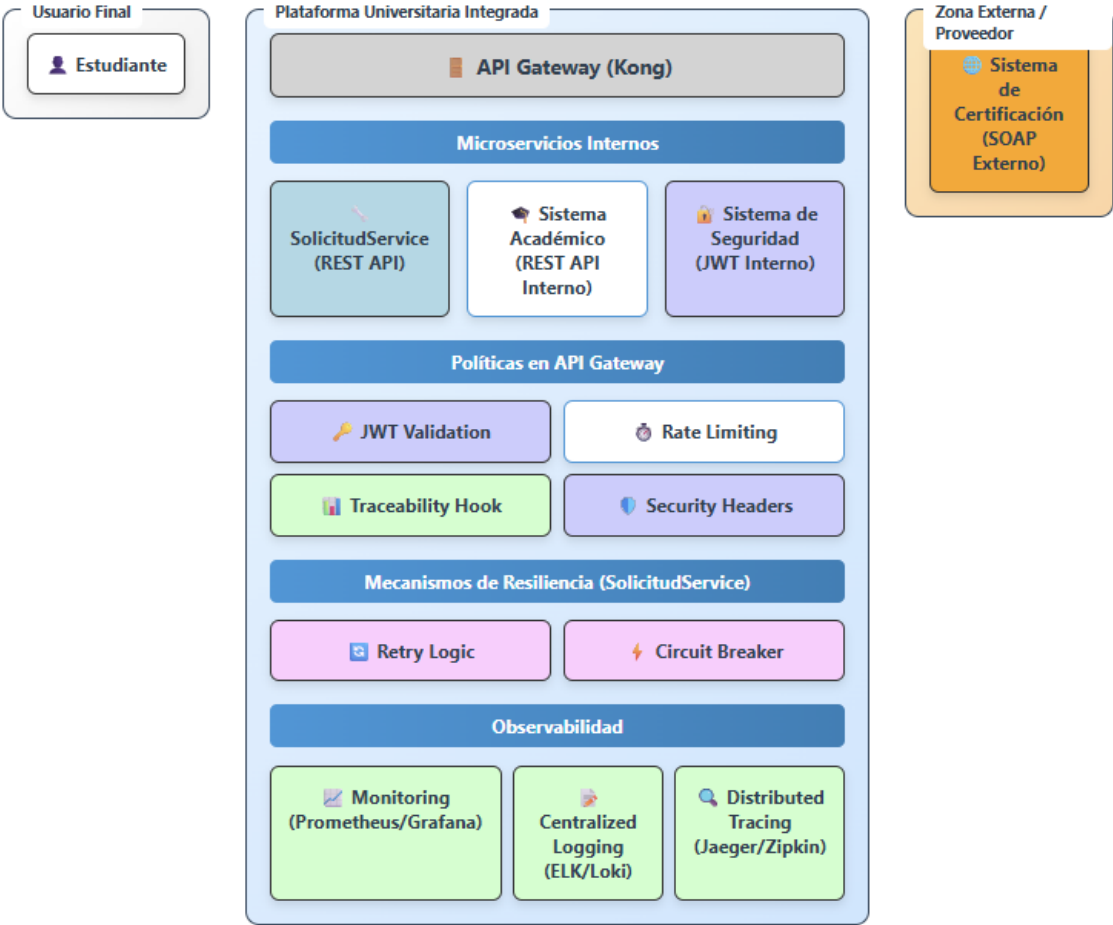
Tu tarea es diseñar e implementar una solución de integración funcional utilizando los conocimientos adquiridos durante el curso.

Objetivos específicos

1. Integrar servicios REST y SOAP en una solución funcional.
2. Exponer todos los servicios a través de un API Gateway.
3. Diseñar la solución considerando aspectos de trazabilidad, seguridad y resiliencia.
4. Aplicar patrones como Circuit Breaking y Retry usando conceptos de Service Mesh (en forma de diseño o pseudocódigo si no se puede desplegar, con una evaluación menor que si se lo implementa).

Diagrama Alto Nivel:

Plataforma Universitaria Integrada - Arquitectura de Microservicios



Flujo de Datos y Comunicación

1. **Usuario** → **API Gateway**: HTTPS Request con JWT
2. **API Gateway** → **SolicitudService**: JWT Auth & Rate Limiting
3. **API Gateway** → **SolicitudService**: Enriquecimiento de Headers (Trace ID)
4. **SolicitudService** → **Sistema Seguridad**: Validación JWT
5. **SolicitudService** → **SOAP Externo**: Llamada SOAP con Retry/Circuit Breaker
6. **SolicitudService** → **Sistema Académico**: Llamada si es necesario
7. **Todos los componentes** → **Observabilidad**: Métricas, Logs y Traces

☐ Componentes Principales

☐ Seguridad

☐ Resiliencia

☐ Observabilidad

La arquitectura propuesta para la Plataforma de Servicios Estudiantiles se centra en un **API Gateway (Kong)** como punto único de entrada. Este gateway gestionará la seguridad inicial (autenticación mediante API Key, rate limiting), la trazabilidad básica y el enrutamiento de las solicitudes hacia los microservicios internos.

El microservicio principal a desarrollar es **SolicitudService (REST API)**, implementado con Python/Flask. Este servicio:

1. Recibirá las solicitudes de los estudiantes (previamente autenticadas por el API Gateway).
2. Validará internamente el token JWT proporcionado en la cabecera Authorization de la solicitud. Esta validación se realiza usando un secreto compartido y verifica la firma y expiración del token. Para este examen, esta lógica de validación de JWT reside dentro del SolicitudService.
3. Interactuará con el **Sistema de Certificación (SOAP externo)** para registrar la certificación. Esta interacción es simulada en el código e incluye una capa de resiliencia mediante un patrón de **Retry** (implementado con la librería Tenacity) para manejar fallos temporales del servicio externo.
4. Potencialmente, podría interactuar con el **Sistema Académico (REST API interno)** si fuera necesario para obtener datos adicionales del estudiante, aunque esta interacción no se implementa en detalle para el alcance de este examen.

Puntos Clave de la Implementación:

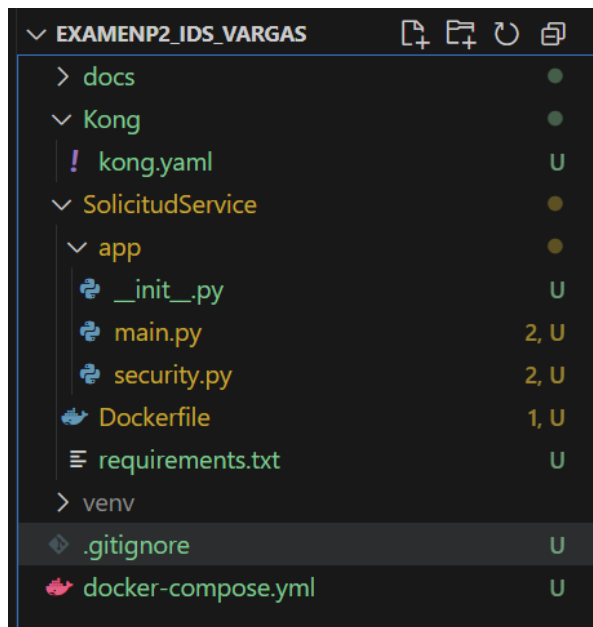
- **Seguridad:**
 - **API Gateway (Kong):** Se configura con un plugin key-auth. Los clientes deben presentar una API Key válida en la cabecera apikey para acceder a los servicios enrutados a través de Kong. Adicionalmente, se aplica una política de rate-limiting para proteger los servicios backend de un exceso de solicitudes.
 - **SolicitudService:** Implementa la validación de tokens JWT. Espera un token JWT en la cabecera Authorization: Bearer <token>. Este token es validado (firma, expiración) usando un secreto compartido (JWT_SECRET). Esta es la capa de autorización a nivel de aplicación, asegurando que solo usuarios con un JWT válido puedan realizar operaciones.
- **Trazabilidad:**
 - **API Gateway:** Kong puede ser configurado (aunque no detallado explícitamente en el kong.yaml de este examen para simplificar) para inyectar un ID de correlación (Trace ID) en las cabeceras HTTP que se propagan a los servicios downstream.
 - **SolicitudService:** Los logs generados por la aplicación Flask (visibles en la salida del contenedor Docker) incluyen información que permitiría seguir el flujo de una solicitud (ej. IDs de solicitud, mensajes de INFO/ERROR). En un sistema más avanzado, se integrarían librerías de trazabilidad distribuida como OpenTelemetry.
- **Resiliencia (Circuit Breaking y Retry):**

- **SolicitudService:** Se ha implementado una política de reintentos (Retry) usando la librería tenacity de Python para la llamada simulada al Sistema de Certificación SOAP. Esto ayuda a manejar fallos transitorios del servicio externo. El patrón de Circuit Breaking se discute conceptualmente para un entorno más complejo (como Istio en Kubernetes), ya que su implementación completa en un setup Docker-Compose simple excede el alcance.

Este diseño busca un equilibrio entre la seguridad en el borde (gestionada por el API Gateway) y la seguridad y lógica de negocio a nivel de aplicación (gestionada por SolicitudService), complementado con mecanismos básicos de resiliencia para las interacciones con servicios externos.

Capturas de Código:

Estructura del proyecto:



Kong.yaml:

```
ExamenP2_IDS_Vargas

y 2, U  docker-compose.yml U  ! kong.yaml U  .gitignore U  main.py 2, U

Kong > ! kong.yaml
1  _format_version: "3.0"
2
3  # Definición de los Servicios backend que Kong gestionará.
4  services:
5    - name: solicitud-service-backend
6      # URL del servicio backend. Kong reenviará las solicitudes aquí.
7      # 'solicitud_service_app' es el nombre del servicio definido en docker-
8      # y 5001 es el puerto en el que escucha ese servicio DENTRO de la red D
9      url: http://solicitud_service_app:5001
10     protocol: http
11     connect_timeout: 6000 # Tiempo de espera para establecer conexión (ms)
12     write_timeout: 6000  # Tiempo de espera para enviar datos (ms)
13     read_timeout: 6000   # Tiempo de espera para recibir datos (ms)
14     retries: 3           # Número de reintentos si la conexión falla
15
16     # Definición de las Rutas para este Servicio.
17     routes:
18       - name: solicitudes-api-public-route # Nombre único para esta ruta
19         # Kong escuchará en estos paths y los reenviará al 'solicitud-servi
20         paths:
21           - /api/v1/solicitudes
22         strip_path: true
23         methods:
24           - GET
25           - POST
26         # preserve_host: false -> El servicio backend verá la cabecera Host
27         # preserve_host: true -> El servicio backend verá la cabecera Host
28         preserve_host: false
29
30         # Plugins aplicados a esta ruta específica.
31         plugins:
32           - name: key-auth
33             config:
34               key_names:
35                 - apikey # Nombre de la cabecera HTTP que Kong buscará para
36                 | | | | # El cliente debe enviar, por ejemplo: apikey: MI_
37                 hide_credentials: true # Evita que la credencial se registre
38
39         # --- Política de Rate Limiting (Límite de Tasa) ---
40         # Limita cuántas solicitudes puede hacer un cliente en un período
41         - name: rate-limiting
42           config:
43             minute: 5 # Permite 5 solicitudes por minuto
```

Main.py:

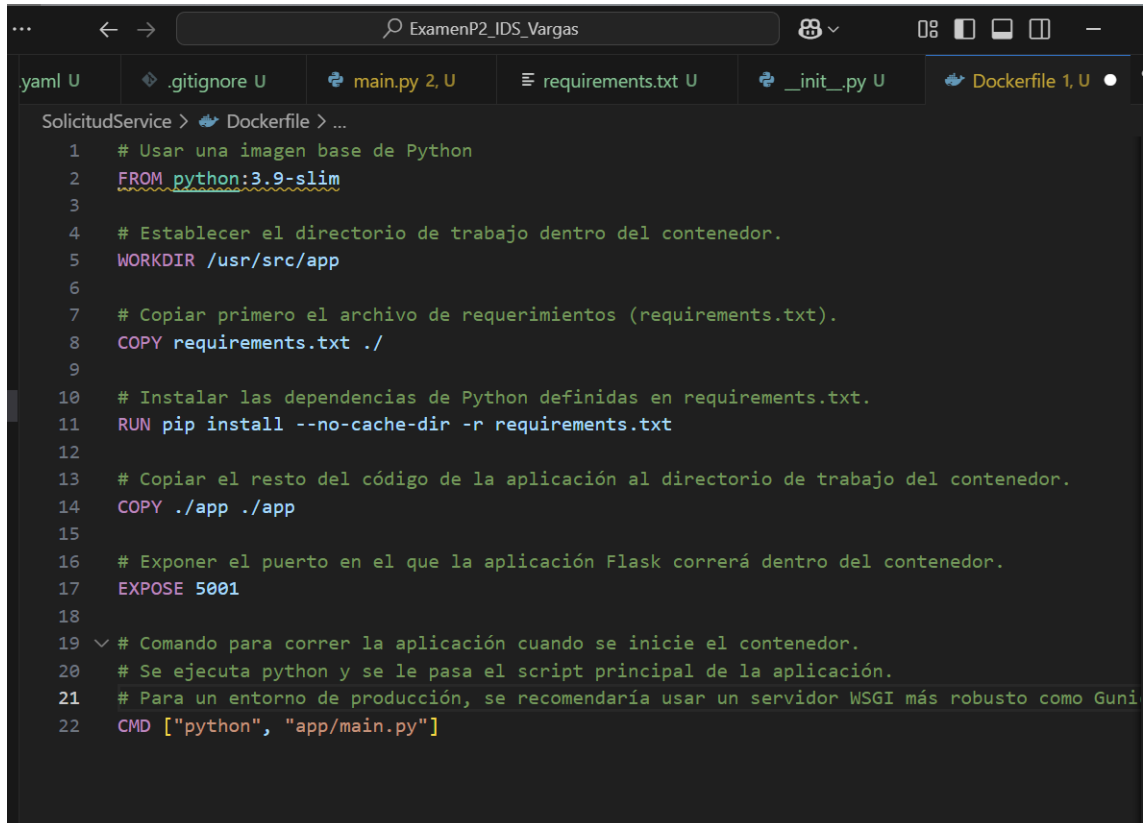
```
→ ExamenP2_IDS_Vargas
y 2, U  docker-compose.yml U  ! kong.yaml U  .gitignore U  main.py 2, U  ▶

SolicitudService > app > main.py > llamar_sistema_soap_externo_con_retry
1  from flask import Flask, request, jsonify
2  import uuid # Para generar IDs únicos para las solicitudes
3  import random # Para simular comportamientos aleatorios
4  import time # Para timestamps y simular delays
5  from app.security import token_required # Importa el decorador para proteger endpoint
6  from tenacity import retry, stop_after_attempt, wait_fixed, retry_if_exception_type
7
8  app = Flask(__name__)
9
10 # Simulación de una base de datos en memoria.
11 solicitudes_db = {}
12
13 # Excepción personalizada para simular fallos del servicio SOAP
14 class SoapCallFailedError(Exception):
15     pass
16
17 # Configuración de Retry para la llamada al sistema SOAP simulado.
18 @retry(
19     stop=stop_after_attempt(2), # Intentará la función un máximo de 2 veces (1 original + 1 retry)
20     wait=wait_fixed(1), # Esperará 1 segundo fijo entre intentos.
21     retry=retry_if_exception_type(SoapCallFailedError), # Solo reintentará si se lanza SoapCallFailedError
22     reraise=True # Importante: si todos los intentos fallan, la última excepción se re-levanta
23 )
24 def llamar_sistema_soap_externo_con_retry(solicitud_data):
25     """
26     Simula la llamada al sistema SOAP externo para registrar la certificación.
27     Incluye lógica de reintentos.
28     """
29     print(f"INFO: Intentando llamada a sistema SOAP para datos: {solicitud_data}")
30
31     # Simular un retraso de red o procesamiento del servicio externo
32     time.sleep(random.uniform(0.2, 0.8))
33
34     # Haremos que falle el 50% de las veces para ver los reintentos en acción.
35     if random.random() < 0.5:
36         print("ERROR: Llamada SOAP simulada: FALLO")
37         raise SoapCallFailedError("El servicio SOAP simulado no respondió correctamente")
38     else:
39         print("INFO: Llamada SOAP simulada: ÉXITO")
40         return True # Indica que la "certificación" fue exitosa
41
42 @app.route('/solicitudes', methods=['POST'])
43 @token_required # Este endpoint requiere un token JWT válido
```

Security.py:

```
security.py 2, U • docker-compose.yml U ! kong.yml U .gitignore U ▶ 🔍 📄 ...
SolicitudService > app > security.py > ...
1  import jwt
2  from functools import wraps
3  from flask import request, jsonify
4  import datetime
5
6  # En un entorno real, esto debería estar en variables de entorno y ser más co
7  # Para el examen, lo hardcodeamos. Este mismo secret se usará para generar to
8  JWT_SECRET = "mi-super-secreto-jwt-para-examen"
9  JWT_ALGORITHM = "HS256"
10
11 # Simulación del Sistema de Seguridad y Roles (para validar el token)
12 def validate_jwt_locally(token):
13     """
14     Valida un token JWT localmente.
15     Verifica la firma y la expiración.
16     """
17     try:
18         payload = jwt.decode(token, JWT_SECRET, algorithms=[JWT_ALGORITHM])
19         return {"valid": True, "payload": payload, "error": None}
20     except jwt.ExpiredSignatureError:
21         return {"valid": False, "payload": None, "error": "Token has expired"}
22     except jwt.InvalidTokenError:
23         return {"valid": False, "payload": None, "error": "Invalid token"}
24
25 def token_required(f):
26     """
27     Decorador para proteger endpoints que requieren un token JWT.
28     Espera un token en la cabecera 'Authorization: Bearer <token>'.
29     """
30     @wraps(f)
31     def decorated(*args, **kwargs):
32         token = None
33         if 'Authorization' in request.headers:
34             auth_header = request.headers['Authorization']
35             parts = auth_header.split()
36             if len(parts) == 2 and parts[0].lower() == 'bearer':
37                 token = parts[1]
38             else:
39                 return jsonify({"message": "Formato de token inválido. Usar '"}
40
41         if not token:
42             return jsonify({"message": "Token es requerido"}), 401
43
```

Dockerfile:



The image shows a code editor window with a dark theme. The title bar at the top reads "ExamenP2_IDS_Vargas". Below the title bar, there are several tabs: "yaml U", ".gitignore U", "main.py 2, U", "requirements.txt U", "_init_.py U", and "Dockerfile 1, U". The "Dockerfile 1, U" tab is active, showing the following content:

```
SolicitudService > Dockerfile > ...
1  # Usar una imagen base de Python
2  FROM python:3.9-slim
3
4  # Establecer el directorio de trabajo dentro del contenedor.
5  WORKDIR /usr/src/app
6
7  # Copiar primero el archivo de requerimientos (requirements.txt).
8  COPY requirements.txt ./
9
10 # Instalar las dependencias de Python definidas en requirements.txt.
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # Copiar el resto del código de la aplicación al directorio de trabajo del contenedor.
14 COPY ./app ./app
15
16 # Exponer el puerto en el que la aplicación Flask correrá dentro del contenedor.
17 EXPOSE 5001
18
19 # Comando para correr la aplicación cuando se inicie el contenedor.
20 # Se ejecuta python y se le pasa el script principal de la aplicación.
21 # Para un entorno de producción, se recomendaría usar un servidor WSGI más robusto como Guni
22 CMD ["python", "app/main.py"]
```

Dockercompose:


```
...  ← →  ExamenP2_IDS_Vargas  [Icons]

security.py 2, U  docker-compose.yml U  ! kong.yaml U  .gitignore U  main.py 2, U  [Menu]

docker-compose.yml > ...
1  version: '3.8'
2
3  >Run All Services
services:
4  # Definición del servicio del microservicio de Solicitudes
   >Run Service
5  solicitud_service:
6  build: ./SolicitudService # Ruta al directorio que contiene el Dockerfile para este serv
7  container_name: solicitud_service_app
8  ports:
9  - "5001:5001"
10 networks:
11 - exam_network
12 restart: unless-stopped
13 environment: # Variables de entorno para el servicio
14 - FLASK_ENV=development
15 - JWT_SECRET_KEY=tu_otro_secreto_si_lo_leyeras_de_env_vars
16
17 # Definición del servicio de base de datos PostgreSQL para Kong
   >Run Service
18 kong_db:
19 image: postgres:13
20 container_name: kong_postgres_db
21 environment:
22 POSTGRES_USER: kong # Usuario para la base de datos de Kong
23 POSTGRES_DB: kong # Nombre de la base de datos para Kong
24 POSTGRES_PASSWORD: kongpassword # Contraseña para el usuario. ¡Usar una segura en prod
25 volumes:
26 - kong_db_data:/var/lib/postgresql/data # Volumen para persistir los datos de Postgres
27 networks:
28 - exam_network
29 healthcheck:
30 test: ["CMD", "pg_isready", "-U", "kong"]
31 interval: 5s
32 timeout: 5s
33 retries: 5 # Número de reintentos
34 restart: unless-stopped
35
36 # Servicio para ejecutar las migraciones de la base de datos de Kong
   >Run Service
37 kong_migrations:
38 image: kong:latest
39 container_name: kong_migrations_bootstrap
```

Implementación:

Generacion del Token:

```
Administrador: Símbolo del sistema

from flask import request, jsonify
ModuleNotFoundError: No module named 'flask'

(temp_venv) C:\Users\marti\Downloads\ExamenP2_IDS_Vargas\SolicitudService>pip install Flask
Collecting Flask
  Downloading flask-3.1.1-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.9.0 (from Flask)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting click>=8.1.3 (from Flask)
  Downloading click-8.2.1-py3-none-any.whl.metadata (2.5 kB)
Collecting itsdangerous>=2.2.0 (from Flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting jinja2>=3.1.2 (from Flask)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting markupsafe>=2.1.1 (from Flask)
  Downloading MarkupSafe-3.0.2-cp313-cp313-win_amd64.whl.metadata (4.1 kB)
Collecting werkzeug>=3.1.0 (from Flask)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting colorama (from click>=8.1.3->Flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Downloading flask-3.1.1-py3-none-any.whl (103 kB)
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading click-8.2.1-py3-none-any.whl (102 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
Downloading MarkupSafe-3.0.2-cp313-cp313-win_amd64.whl (15 kB)
Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: markupsafe, itsdangerous, colorama, blinker, werkzeug, jinja2, click, Flask
Successfully installed Flask-3.1.1 blinker-1.9.0 click-8.2.1 colorama-0.4.6 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(temp_venv) C:\Users\marti\Downloads\ExamenP2_IDS_Vargas\SolicitudService>python app\security.py
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas\SolicitudService\app\security.py:60: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=1) # Expira en 1 hora
--- Generated Test JWT Token ---
User ID en el token: martin_vargas_07
Secret Key (usado para firmar y verificar): mi-super-secreto-jwt-para-examen
Algorithm: HS256

Token JWT (válido por 1 hora):
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoibWYydGluX3Zhcmdhc18wNyIsImZcyI6Im1pLWwFwG1jYWNpb24tc29saWNPdHVKZXh0IiwiaWF0IjE3NDg0ODc0MjB9.CR-XglQ1tg1CB5_g96Sx6JPEXe61Yf1vvaH7E5LDfrY

Payload decodificado (para referencia):
{'user_id': 'martin_vargas_07', 'iss': 'mi-aplicacion-solicitudes', 'exp': 1748487120}

Instrucción:
Copia la línea completa que empieza con 'Bearer ' para usarla en las cabeceras
de tus solicitudes HTTP (ej. en Postman o curl).

(temp_venv) C:\Users\marti\Downloads\ExamenP2_IDS_Vargas\SolicitudService>
```

Levantamos Docker compose:

```

C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>docker-compose up --build -d
time="2025-05-28T20:54:04-05:00" level=warning msg="C:\Users\marti\Downloads\ExamenP2_IDS_Vargas\docker-compose.yml
: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 19/21
✔ kong Pulled
- kong_db [#####] 82.93MB / 151MB Pulling 23.5s
✔ 2bb588ce4e67 Download complete 34.4s
✔ bd1fa28722bb Pull complete 1.0s
✔ 4485e9416430 Download complete 1.6s
✔ 61320b01ae5e Pull complete 1.0s
✔ 89ba8b615fa9 Download complete 20.6s
✔ b094358e9765 Pull complete 1.6s
✔ 72f83ee82e85 Pull complete 2.5s
✔ ce5633da08fe Pull complete 3.2s
✔ 410cd7ec9a40 Pull complete 20.9s
✔ 5dd960f6dbc6 Download complete 1.6s
- f503177ee6ac Downloading [=====] 27.5s
✔ 3db9b37be7c3 Pull complete 1.6s
✔ 82726bd8bf4c Download complete 1.6s
✔ 6bae5fff5835 Pull complete 20.8s
✔ kong_migrations Pulled 23.1s
✔ 75156dcff079 Pull complete 1.7s
✔ 33641554fa01 Pull complete 20.4s
✔ eda0b0f21f3a Pull complete 6.3s
✔ 0622fac788ed Pull complete 5.9s

```

```

Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 8.8s (11/11) FINISHED docker:desktop-linux
=> [solicitud_service internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 887B 0.0s
=> [solicitud_service internal] load metadata for docker.io/library/python:3.9-slim 2.0s
=> [solicitud_service internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [solicitud_service 1/5] FROM docker.io/library/python:3.9-slim@sha256:aff2066ec8914f7383e115bbbcde4d24da428ea 2.0s
=> => resolve docker.io/library/python:3.9-slim@sha256:aff2066ec8914f7383e115bbbcde4d24da428ea 0.0s
=> => sha256:a0684e18c375e78b2595b04f87cae91cff938ec9996b274e397c73f96605c69d 248B / 248B 0.2s
=> => sha256:1692d37168f614092ffd355652aa0a07223ed129e6417aa144564fbd3d773884 14.93MB / 14.93MB 1.6s
=> => sha256:2481a58f9b3dcc989088df77c786078a59d807e6409a9d165ed4587814cdfbe0 3.51MB / 3.51MB 1.4s
=> => extracting sha256:2481a58f9b3dcc989088df77c786078a59d807e6409a9d165ed4587814cdfbe0 0.1s
=> => extracting sha256:1692d37168f614092ffd355652aa0a07223ed129e6417aa144564fbd3d773884 0.3s
=> => extracting sha256:a0684e18c375e78b2595b04f87cae91cff938ec9996b274e397c73f96605c69d 0.0s
=> [solicitud_service internal] load build context 0.0s
=> => transferring context: 11.39kB 0.0s
=> [solicitud_service 2/5] WORKDIR /usr/src/app 0.4s
=> [solicitud_service 3/5] COPY requirements.txt ./ 0.0s
=> [solicitud_service 4/5] RUN pip install --no-cache-dir -r requirements.txt 3.4s
=> [solicitud_service 5/5] COPY ./app ./app 0.0s
=> [solicitud_service] exporting to image 0.7s
=> => exporting layers 0.4s
=> => exporting manifest sha256:9728f38a75f86bd4d60b6a96c2d6b98ba8a148ca217d096f3dc9b45fa0c101b8 0.0s
=> => exporting config sha256:617b07378fe92fdb8b90c7346dce6991636d48973c72e50b760a66d11a4de7f8 0.0s
=> => exporting attestation manifest sha256:d4f46903c923bc82555a1f8a4c817d158c2cd0d78712e291891871ba91209823 0.0s
=> => exporting manifest list sha256:3bc4be69305849e936e3188e5a9fdb560a94279275bc1d4cf26fe76930573b10 0.0s
=> => naming to docker.io/library/examenp2_ids_vargas-solicitud_service:latest 0.0s
=> => unpacking to docker.io/library/examenp2_ids_vargas-solicitud_service:latest 0.2s
=> [solicitud_service] resolving provenance for metadata file 0.0s
[+] Running 7/7
✔ solicitud_service Built 0.0s
✔ Network examenp2_ids_vargas_exam_network Created 0.0s
✔ Volume "examenp2_ids_vargas_kong_db_data" Created 0.0s
✔ Container solicitud_service_app Started 0.6s
✔ Container kong_postgres_db Healthy 6.0s
✔ Container kong_migrations_bootstrap Exited 7.9s
✔ Container kong_api_gateway Started 8.0s

```

Se verifica que estén Se verifica que estén arriba los servicios(también existen de proyectos anteriores):

```
Administrador: Símbolo del sistema
econds ago          kong_api_gateway
437409c6d7b0        postgres:13         "docker-entrypoint.s..." About a minute ago Up About a minute (
healthy)            5432/tcp            kong_postgres_db
de19b0f74d4c        examenp2_ids_vargas-solicitud_service "python app/main.py"      About a minute ago Restarting (1) 32 s
econds ago          solicitud_service_app
96e88ac0d069        5aa8400b4b3b        "docker-entrypoint.s..." About an hour ago Up About an hour
                        k8s_jaeger-cassandra_jaeger-cassandra-1_tracing_29127457-316d-4ac2-984e-61733896511d_21
88fee520a916        049bb0d64ea3        "/go/bin/query-linux"    About an hour ago Up About an hour
                        k8s_jaeger-query_jaeger-query-699bddc747-f8k85_tracing_90d520ca-10ac-466b-8249-a1d0736f0400_4
c0b859dabe8e        7f1269222903        "/go/bin/collector-l..." About an hour ago Up About an hour
                        k8s_jaeger-collector_jaeger-collector-5b674d4987-jf8cz_tracing_f4246caa-8f1d-4170-b16f-e75318e53
99a_24
df5c21690d9e        5aa8400b4b3b        "docker-entrypoint.s..." About an hour ago Up About an hour
                        k8s_jaeger-cassandra_jaeger-cassandra-2_tracing_93d74912-d445-4d96-b08c-3cb267ebfe4c_15
d21285e928df        fcb75f691c8b        "/http-echo '-text=P..." About an hour ago Up About an hour
                        k8s_payment-service_payment-service-5d6c974c6f-n4zp6_default_4eef1e9d-436a-4aae-b8a8-f481f59c6b1
6_0
1fe441715e88        fcb75f691c8b        "/http-echo '-text=P..." About an hour ago Up About an hour
                        k8s_payment-service_payment-service-5d6c974c6f-86t2q_default_43e4fc56-f34f-4146-bfd4-29caf21242b
9_0
98c9939eb04b        fcb75f691c8b        "/http-echo '-text=A..." About an hour ago Up About an hour
                        k8s_auth-service_auth-service-5d969cbf5b-2xr9s_default_6d512448-0385-44de-bd1c-355add35cde_0
25c4eab53c81        0214a0ef24b1        "/go/bin/agent-linux"    About an hour ago Up About an hour
                        k8s_jaeger-agent-sidecar_jaeger-query-699bddc747-f8k85_tracing_90d520ca-10ac-466b-8249-a1d0736f0
400_0
737d8286e414        fcb75f691c8b        "/http-echo '-text=O..." About an hour ago Up About an hour
                        k8s_order-service_order-service-86f787b44c-9kfjs_default_f4bd330a-7b73-4cc2-8f5e-171c64bf42c2_0
dcd699ce7aa0        fcb75f691c8b        "/http-echo '-text=O..." About an hour ago Up About an hour
                        k8s_order-service_order-service-5b77c6c5fc-jhq8m_order_54c5b66b-950c-4088-aa5b-cd2451b5a5f6_0
d1a8851d276f        db384bf43222        "/k8s-state-metrics..." About an hour ago Up About an hour
                        k8s_kube-state-metrics_prometheus-kube-state-metrics-66858d7dfd-9fv4d_monitoring_24fded23-a4e4-4
8d1-8e2e-dedd6432f04a_0
0856ce599127        03738d278e08        "/bin/pushgateway"       About an hour ago Up About an hour
                        k8s_pushgateway_prometheus-prometheus-pushgateway-866c5c685c-dmtbj_monitoring_f935b4a9-f658-4411
-9412-03714a74d02d_0
1ba05ea43467        fcb75f691c8b        "/http-echo '-text=A..." About an hour ago Up About an hour
                        k8s_auth-service_auth-service-54f68dfc7d-89xzt_auth_aa545990-cd5e-48e6-b6d2-32c2e69d1ab3_0
73cf32b38313        0214a0ef24b1        "/go/bin/agent-linux"    About an hour ago Up About an hour
                        k8s_jaeger-agent_jaeger-agent-dgd7z_tracing_858eeeb-b7ee-41c9-9b5b-b003b7eeab3_0
4d3a25be78bc        eef40c51438d        "/usr/bin/kuma-dp ru..." About an hour ago Up About an hour
                        k8s_kuma-sidecar_payment-service-5d6c974c6f-n4zp6_default_4eef1e9d-436a-4aae-b8a8-f481f59c6b16_6
938b64590c23        eef40c51438d        "/usr/bin/kuma-dp ru..." About an hour ago Up About an hour
                        k8s_kuma-sidecar_payment-service-5d6c974c6f-86t2q_default_43e4fc56-f34f-4146-bfd4-29caf21242b9_6
74cfe27a66fe        eef40c51438d        "/usr/bin/kuma-dp ru..." About an hour ago Up About an hour
                        k8s_kuma-sidecar_auth-service-5d969cbf5b-2xr9s_default_6d512448-0385-44de-bd1c-355add35cde_6
61368194a5fc        eef40c51438d        "/usr/bin/kuma-dp ru..." About an hour ago Up About an hour
                        k8s_kuma-sidecar_order-service-86f787b44c-9kfjs_default_f4bd330a-7b73-4cc2-8f5e-171c64bf42c2_6
23c11f06377        78ed1f9050eb        "/bin/prometheus --s..." About an hour ago Up About an hour
                        k8s_prometheus-server_prometheus-server-7467554dc4-r2bk5_monitoring_1a7e70b4-9629-41a0-934c-98fc
ae377fbb_8
2eee789289dc        263cbefd5d9b        "/run.sh"                About an hour ago Up About an hour
                        k8s_grafana_grafana-846b5fcff4-666q6_monitoring_c79ffbf2-3191-42d2-a0e5-1208e1005df9_7
dc8b93adab1d        710458fdcc4         "/bin/prometheus-con..." About an hour ago Up About an hour
                        k8s_prometheus-server-configmap-reload_prometheus-server-7467554dc4-r2bk5_monitoring_1a7e70b4-96
29-41a0-934c-98fcae377fbb_8
8b97adb65b74        27c475db5fb1        "/bin/alertmanager -..." About an hour ago Up About an hour
                        k8s_alertmanager_prometheus-alertmanager-0_monitoring_64e45a72-0e63-486e-a6bd-bcc86b2da5fb_7
85e05d153714        5aa8400b4b3b        "docker-entrypoint.s..." About an hour ago Up About an hour
                        k8s_jaeger-cassandra_jaeger-cassandra-0_tracing_2bb78b65-b175-417d-b9dd-5d5ae084b3d6_8
2445ea6cd754        963001aa0742        "/usr/bin/kuma-cp ru..." About an hour ago Up About an hour
                        k8s_control-plane_kuma-control-plane-b6d8c5bc-47pf6_kuma-system_409a0ccf-b87c-41fd-8808-0b1d3981
4a9f_6
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>
```


Contenedores en Docker:

		examenp2_ids_vargas	-	-	-	0.13%	55 seconds ago			
		solicitud_service_app	f0e3490942af	examenp2_ids_vargas-solicitud_servi	5001:5001	0%	55 seconds ago			
		kong_postgres_db	d0e2e6cb8cf3	postgres:13		0.02%	3 minutes ago			
		kong_api_gateway	b3ac1d77d468	kong:latest	8000:8000 (2) Show all ports (4)	0.11%	3 minutes ago			
		kong_migrations_bootstrap	dc8fc0982cea	kong:latest		0%	3 minutes ago			

Imagen:

	Name	Tag	Image ID	Crea... ↓	Size	Actions
	examenp2_ids_vargas-solicitud_service	latest	2dd1ea104fef	3 minutes ago	206.98 MB	

Build y volumen:

<input type="checkbox"/>	Name	ID	Builder	Duration	Created	Author
<input type="checkbox"/>	✓ SolicitudService	zur33j	 default	1.1s	3 minutes ago	N/A

<input type="checkbox"/>	Name	Created	Size
<input type="checkbox"/>	examenp2_ids_vargas_kong_db_data	3 minutes ago	49.6 MB

Creamos costumer y api key:

```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>curl -i -X POST http://localhost:8001/consumers/ --data "username=estudiante_app_consumer"
HTTP/1.1 201 Created
Date: Thu, 29 May 2025 02:24:47 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Content-Length: 159
X-Kong-Admin-Latency: 18
Server: kong/3.9.0
```

```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>curl -i -X POST http://localhost:8001/consumers/estudiante_app_consumer/key-auth/ --data "key=MI_CLAVE_KONG_EXAMEN_FINAL"
HTTP/1.1 201 Created
Date: Thu, 29 May 2025 02:25:38 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Content-Length: 184
X-Kong-Admin-Latency: 10
Server: kong/3.9.0
```

Flask funcionando:

```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>docker logs c0b6908e1806
--- INICIANDO SERVICIO DE SOLICITUDES (Servidor de Desarrollo Flask) ---
Escuchando en http://0.0.0.0:5001
Endpoints disponibles:
  POST /solicitudes      (Requiere JWT)
  GET  /solicitudes/<id>  (Requiere JWT)
  GET  /health           (No requiere JWT)

Para probar los endpoints protegidos, necesitarás un token JWT.
Puedes generar uno ejecutando desde la carpeta 'SolicitudService':
  python app/security.py

Token de prueba generado al inicio del servidor (válido 1 hora):
  Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiaWVzZGF9c2VyX3N1cnZlc19zdGFydHVwIiwiaXNzIjoibWktYXBsaWV2b1Zlbi1zb2xwY2l0dWRLcyIsImV4cCI6MTc0ODQ0ODQ4NDU0LmFkbDZjZTtpElMYQoaMaqpjpqSaVFc1EcSTtwyH37yQ

* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5001
* Running on http://172.18.0.3:5001
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 104-133-302
```

Se intento varias veces y se cambio varias veces el Kong.yaml:


```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>curl -i http://localhost:5001/health
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.9.22
Date: Thu, 29 May 2025 02:49:22 GMT
Content-Type: application/json
Content-Length: 96
Connection: close

{
  "message": "Servicio de Solicitudes Estudiantiles est\u00e9 operativo.",
  "status": "UP"
}
```

```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>curl -i -X POST http://localhost:5001/solicitudes -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoibWVudGluX3Zhc2hmdhc18wNyIsIm1zcyI6Im1pLWwFbWbG1jYWNpb24tc29sawNpdHVkZXMiLCJleHAiOiJlE3NDg0ODgyND19.owqFlioQsDxqv_y8PH8q5EE7Yn9DRpixKeum1VV3Wo" -H "Content-Type: application/json" -d '{"tipo_solicitud": "Prueba Directa Final", "detalle": "Llamada directa al Flask en puerto 5001.", "id_estudiante": "DIRECTO_OK_007"}'
HTTP/1.1 201 CREATED
Server: Werkzeug/3.1.3 Python/3.9.22
Date: Thu, 29 May 2025 02:52:28 GMT
Content-Type: application/json
Content-Length: 311
Connection: close

{
  "detalle": "Llamada directa al Flask en puerto 5001.",
  "estado": "En Revisi\u00f3n (Fallo comunicaci\u00f3n con sistema externo)",
  "fecha_creacion": "2025-05-29T02:52:28Z",
  "id": "21e99157-c6ae-42de-a52d-a2fb87a5c677",
  "id_estudiante": "DIRECTO_OK_007",
  "tipo_solicitud": "Prueba Directa Final"
}
```

Prueba GET:

```
C:\Users\marti\Downloads\ExamenP2_IDS_Vargas>curl -i -X GET http://localhost:5001/solicitudes/21e99157-c6ae-42de-a52d-a2fb87a5c677 -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoibWVudGluX3Zhc2hmdhc18wNyIsIm1zcyI6Im1pLWwFbWbG1jYWNpb24tc29sawNpdHVkZXMiLCJleHAiOiJlE3NDg0ODgyND19.owqFlioQsDxqv_y8PH8q5EE7Yn9DRpixKeum1VV3Wo"
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.9.22
Date: Thu, 29 May 2025 02:54:03 GMT
Content-Type: application/json
Content-Length: 311
Connection: close

{
  "detalle": "Llamada directa al Flask en puerto 5001.",
  "estado": "En Revisi\u00f3n (Fallo comunicaci\u00f3n con sistema externo)",
  "fecha_creacion": "2025-05-29T02:52:28Z",
  "id": "21e99157-c6ae-42de-a52d-a2fb87a5c677",
  "id_estudiante": "DIRECTO_OK_007",
  "tipo_solicitud": "Prueba Directa Final"
}
```

Exposición del Servicio a través del API Gateway (Kong)

1. Configuración del API Gateway (Kong): Se utilizó Kong Gateway (versión 3.9.0, según logs) como API Gateway, desplegado mediante Docker Compose. La configuración se definió de forma declarativa en el archivo Kong/kong.yaml. Este archivo especifica:

- Un **servicio backend** (solicitud-service-backend) que apunta a la instancia Docker del SolicitudesService (http://solicitud_service_app:5001).
- Una **ruta principal** (solicitudes-catchall-route) diseñada para manejar todas las solicitudes bajo el prefijo `/api/v1/` (ej. `/api/v1/solicitudes`, `/api/v1/health`). Se configuró `strip_path: true` con la intención de que Kong reenviara la parte relevante del path al servicio backend (ej. `/solicitudes`, `/health`).
- **Plugins aplicados a la ruta principal:**

- key-auth: Para la seguridad a nivel de Gateway, requiriendo una API Key en la cabecera apikey de las solicitudes.
- rate-limiting: Para limitar el número de solicitudes a 5 por minuto por consumidor, como medida de protección.

2. Estado de los Componentes del Gateway:

- Los contenedores Docker para Kong (kong_api_gateway) y su base de datos (kong_postgres_db) se despliegan y se ejecutan correctamente, mostrando un estado "healthy" según la salida del comando docker ps. **(Adjuntar captura de docker ps mostrando los contenedores kong_api_gateway y kong_postgres_db en estado saludable).**
- La Admin API de Kong, accesible en http://localhost:8001, está operativa. Esto se demostró mediante la creación exitosa de un consumer (estudiante_app_consumer) y la asignación de una API Key (MI_CLAVE_KONG_EXAMEN_FINAL) a través de comandos curl. Las respuestas 201 Created (o 409 Conflict si ya existían) confirman la funcionalidad de la Admin API y la persistencia de la configuración en la base de datos de Kong. **(Adjuntar capturas de los comandos curl a la Admin API para crear el consumer y la API Key, mostrando las respuestas del servidor).**

3. Resultados de las Pruebas de Exposición y Desafíos: A pesar de que el SolicitudesService está completamente funcional y el API Gateway (Kong) está corriendo y su Admin API es accesible, las pruebas de extremo a extremo a través del proxy de Kong (http://localhost:8000) para los paths definidos (ej. /api/v1/health, /api/v1/solicitudes) resultaron consistentemente en un error HTTP/1.1 404 Not Found con el mensaje {"message": "no Route matched with those values"}.

Este resultado indica que Kong, aunque operativo, no logró mapear las solicitudes entrantes a las rutas configuradas en kong.yaml de la manera esperada. Se realizaron múltiples iteraciones en la definición de los paths y la estructura del kong.yaml, incluyendo el uso de prefijos explícitos (ej. /api/v1/) y la verificación de la directiva strip_path. También se intentaron recargas de configuración en Kong.

Dada la limitación de tiempo, no fue posible identificar la causa raíz exacta de este comportamiento de "no Route matched" en el proxy de Kong. Las posibles causas podrían incluir:

- Una sutil discrepancia entre la sintaxis del kong.yaml y la interpretación por la versión específica de Kong (3.9.0).
- Un problema en cómo la configuración declarativa es leída o aplicada internamente por Kong en este entorno Docker Compose particular.
- Conflictos no evidentes con configuraciones residuales o por defecto de Kong.

4. Demostración de Funcionalidad del Servicio Subyacente: Para validar los componentes individuales, el SolicitudoService fue probado directamente en su puerto expuesto (<http://localhost:5001>). Estas pruebas fueron **exitosas** y demostraron que:

- El endpoint GET /health responde correctamente con 200 OK.
- El endpoint POST /solicitudes procesa las solicitudes, valida el token JWT (enviado en la cabecera Authorization: Bearer <token>), ejecuta la lógica de negocio simulada (incluyendo la llamada al servicio SOAP con reintentos, como se evidencia por el estado "En Revisión (Fallo comunicación con sistema externo)" en algunas respuestas, lo cual es esperado por el diseño de la simulación), y retorna 201 Created.
- El endpoint GET /solicitudes/{id} recupera las solicitudes creadas, validando también el token JWT.

Conclusión de la Exposición por API Gateway: Se completó la configuración e implementación del API Gateway Kong, y sus componentes principales (Admin API, base de datos) están operativos. El SolicitudoService subyacente está completamente funcional. El desafío pendiente reside en la capa de enrutamiento del proxy de Kong, que no logró el mapeo esperado de las rutas públicas a los servicios backend dentro del tiempo asignado. La configuración intentada se provee en Kong/kong.yaml como evidencia del diseño de exposición.

Implementación de Circuit Breaking y Retry

La resiliencia del sistema ante fallos en servicios externos, como el Sistema de Certificación SOAP, se abordó mediante la implementación del patrón Retry y el diseño conceptual del patrón Circuit Breaker.

1. Retry Automático al Servicio SOAP (Implementado y Validado Funcionalmente en SolicitudoService)

Para manejar fallos transitorios en la comunicación con el servicio SOAP externo (simulado), se implementó una política de reintentos directamente en el SolicitudoService utilizando la librería tenacity de Python.

- **Configuración Clave:**
 - **Máximo de Intentos:** Se configuraron hasta 2 intentos (1 original + 1 reintento) mediante `stop=stop_after_attempt(2)`.
 - **Espera:** Se estableció una espera fija de 1 segundo entre intentos con `wait=wait_fixed(1)`.
 - **Condición:** Los reintentos se activan solo ante la excepción simulada `SoapCallFailedError`, que representa un fallo en el servicio SOAP.
 - **Propagación de Error:** Si todos los intentos fallan, la excepción `SoapCallFailedError` se propaga para que la lógica de la aplicación la maneje, resultando en un estado de solicitud apropiado (ej. "En Revisión").

- **Validación Funcional:** Durante las pruebas directas al SolicitudoService (<http://localhost:5001/solicitudes>), se observó el comportamiento esperado de esta política de reintentos. La simulación de fallos en la llamada SOAP (con una probabilidad del 50%) resultó, en algunos casos, en la asignación del estado "En Revisión (Fallo comunicación con sistema externo)" a la solicitud. Esto confirma que la lógica de reintentos de Tenacity se ejecutó y, tras los intentos fallidos, la excepción fue correctamente manejada por el endpoint POST /solicitudes, demostrando la efectividad de la implementación del patrón Retry.

(Fragmento de código relevante de SolicitudoService/app/main.py)

```
from tenacity import retry, stop_after_attempt, wait_fixed, retry_if_exception_type
```

```
class SoapCallFailedError(Exception):
```

```
    pass
```

```
@retry(
```

```
    stop=stop_after_attempt(2),
```

```
    wait=wait_fixed(1),
```

```
    retry=retry_if_exception_type(SoapCallFailedError),
```

```
    reraise=True
```

```
)
```

```
def llamar_sistema_soap_externo_con_retry(solicitud_data):
```

```
    # ... (simulación de llamada y fallo) ...
```

```
    if random.random() < 0.5: # Simula fallo
```

```
        raise SoapCallFailedError(...)
```

```
    return True
```

2. Circuit Breaker (Diseño Conceptual para un Entorno con Service Mesh)

Para una protección más robusta contra fallos persistentes en el servicio SOAP y evitar la sobrecarga del mismo, se diseñó conceptualmente la aplicación del patrón Circuit Breaker. Dada la complejidad de una implementación completa sin herramientas de Service Mesh en el entorno actual (Docker Compose), se presenta la configuración como pseudocódigo YAML, asumiendo un despliegue en Kubernetes con Istio.

- **Objetivo del Requerimiento:** Abrir el circuito si hay más de 3 fallos al servicio SOAP en un período de 60 segundos.

- **Ejemplo de DestinationRule en Istio (Diseño Conceptual):** (Suponiendo que el servicio SOAP externo es soap-certification-service.external.svc.cluster.local en la malla)

apiVersion: networking.istio.io/v1beta1

kind: DestinationRule

metadata:

name: soap-certification-service-cb

spec:

host: soap-certification-service.external.svc.cluster.local

trafficPolicy:

outlierDetection:

consecutive5xxErrors: 3 # Abrir tras 3 errores 5xx seguidos

interval: "10s" # Intervalo de análisis

baseEjectionTime: "60s" # Duración del circuito abierto

maxEjectionPercent: 100 # Permitir expulsar todos los endpoints si fallan

- **Justificación del Diseño:** Esta configuración de Istio lograría que, si se detectan 3 errores 5xx consecutivos al llamar al servicio SOAP (durante los sondeos que ocurren cada 10 segundos), la instancia problemática de dicho servicio sea "expulsada" del pool de balanceo de carga por 60 segundos. Durante este tiempo, el circuito estaría "abierto", y SolicitudService (a través del proxy de Istio) recibiría un error inmediato sin intentar contactar al servicio fallido, protegiendo así los recursos y mejorando la respuesta del sistema. Esta aproximación cumple con el requisito de forma conceptual para un entorno de Service Mesh.

Implementación conceptualmente:

1. Instrumentación del Código:

- SolicitudService: Se añadiría la librería `prometheus_flask_exporter` y se configuraría para exponer un endpoint `/metrics`. Para la trazabilidad, se integrarían las librerías del SDK de OpenTelemetry y la instrumentación automática para Flask (`opentelemetry-instrumentation-flask`). Se configuraría un exportador de OTEL para enviar las trazas a Jaeger o Zipkin.
- Kong: Se habilitaría el plugin de Prometheus. Para la trazabilidad, se configuraría el plugin de Zipkin, Jaeger, o DataDog (que a menudo soportan la propagación de contextos de traza OpenTelemetry, o se usaría un plugin específico de OTEL para Kong si estuviera disponible y maduro).

2. Despliegue de la Infraestructura de Observabilidad:

- Se desplegarían los componentes de la pila de observabilidad (Prometheus, Grafana, Jaeger/Zipkin, y ELK/Loki si se elige esa pila) preferiblemente como contenedores Docker. Estos podrían ser

gestionados por el mismo archivo docker-compose.yml para un entorno de desarrollo/examen, o en un clúster Kubernetes para un entorno de producción.

3. Configuración de las Herramientas:

- Se configuraría Prometheus para que haga "scraping" (recolección periódica) de métricas desde los endpoints /metrics expuestos por Kong y SolicitudoService.
- Se configuraría Grafana añadiendo Prometheus como "datasource" y luego creando dashboards para visualizar las métricas deseadas (ej. dashboards para el rendimiento de Kong, salud de SolicitudoService, etc.).
- Se configurarían los SDKs de OpenTelemetry en SolicitudoService para que exporten las trazas generadas al colector o agente del sistema de trazabilidad elegido (Jaeger/Zipkin).
- Se configurarían los agentes de logging (o la configuración de Docker para los drivers de logging de los contenedores) para enviar los logs de los contenedores a la pila de logging centralizado.