

Universidad de Las Américas
Facultad de Ingenierías y Ciencias Agropecuarias
Ingeniería De Software
Progreso 1

Nombres: Enrique Merizalde

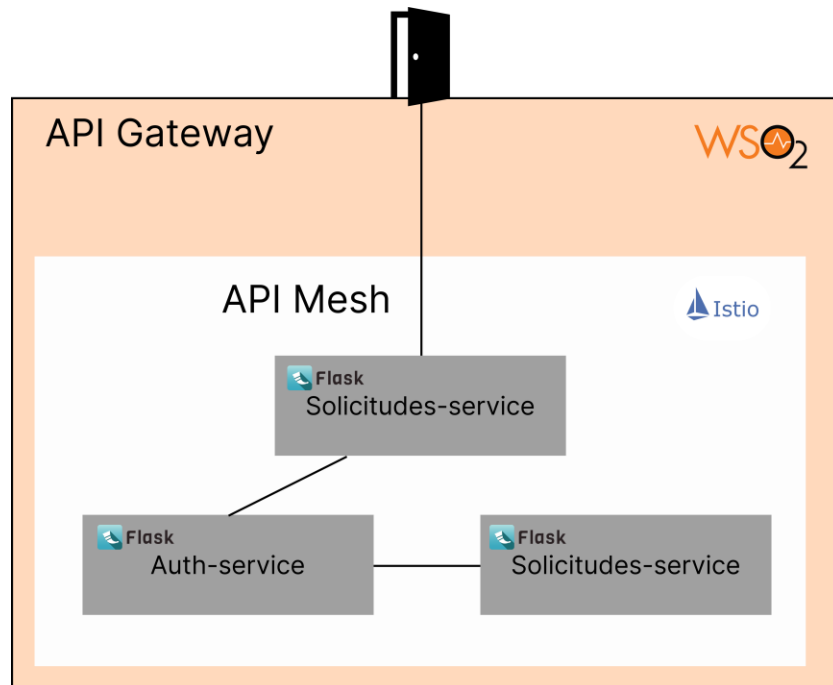
Fecha: 28/05/2025

INFORME PRUEBA DE INTEGRACIÓN

1. Objetivo General

Diseñar e implementar una solución de integración funcional entre tres sistemas independientes para la gestión de solicitudes académicas, integrando servicios REST y SOAP bajo un API Gateway, con patrones de Service Mesh como Circuit Breaking y Retry.

2. Arquitectura del sistema



Se ha diseñado un sistema basado en 3 microservicios que interactúan entre, los cuales estarán dentro de un API Mesh y serán expuestos mediante un API Gateway.

Descripción de Componentes:

- SolicitudService (Flask, REST): recibe solicitudes, valida el token JWT y llama al servicio de certificación.
- AuthService (Flask): emite y valida tokens JWT.
- CertificacionService (Flask, simula SOAP): simula el registro de certificación.
- WSO2 API Manager: actúa como gateway, asegurando y limitando el acceso a la API.
- Istio Service Mesh: implementa las políticas de Circuit Breaking y Retry para evitar controlar la cantidad de reintentos al fallar y que no se realicen peticiones exageradas que puedan romper la aplicación. Esto se ha configurado en un archivo .yaml

Flujo del proceso:

1. El cliente envía una solicitud a /solicitudes pasando un token JWT.
2. WSO2 API Gateway verifica el token y aplica limitación de tasa.
3. El tráfico pasa por el Ingress Gateway de Istio, quien enruta al microservicio solicitud-service.
4. solicitud-service valida el token contactando a auth-service.
5. Luego llama al certificacion-service para simular la operación SOAP.
6. Istio gestiona los retries y circuit breaking en la comunicación interna.
7. La respuesta final es retornada al cliente.

\

3. Implementación de Microservicios

SolicitudService

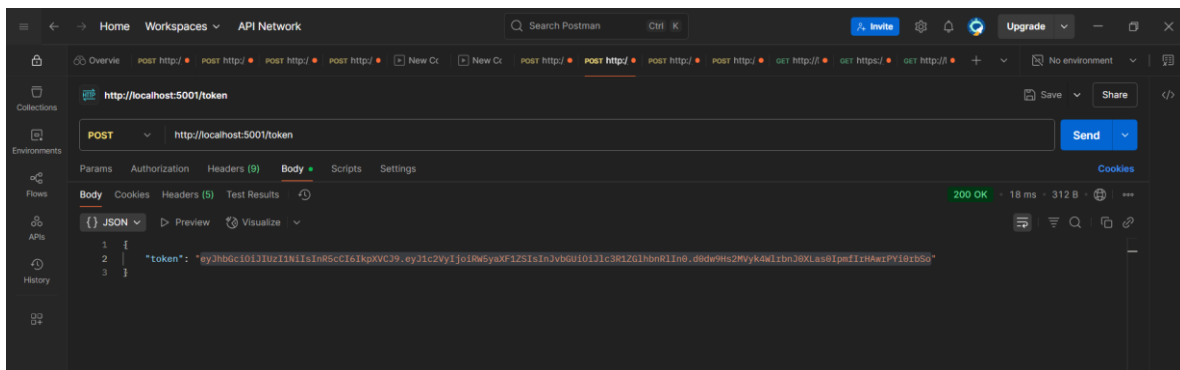
- Endpoint: POST /solicitudes, GET /solicitudes/<id>
- JWT obligatorio en el header.
- Llama al mock SOAP y responde el estado.

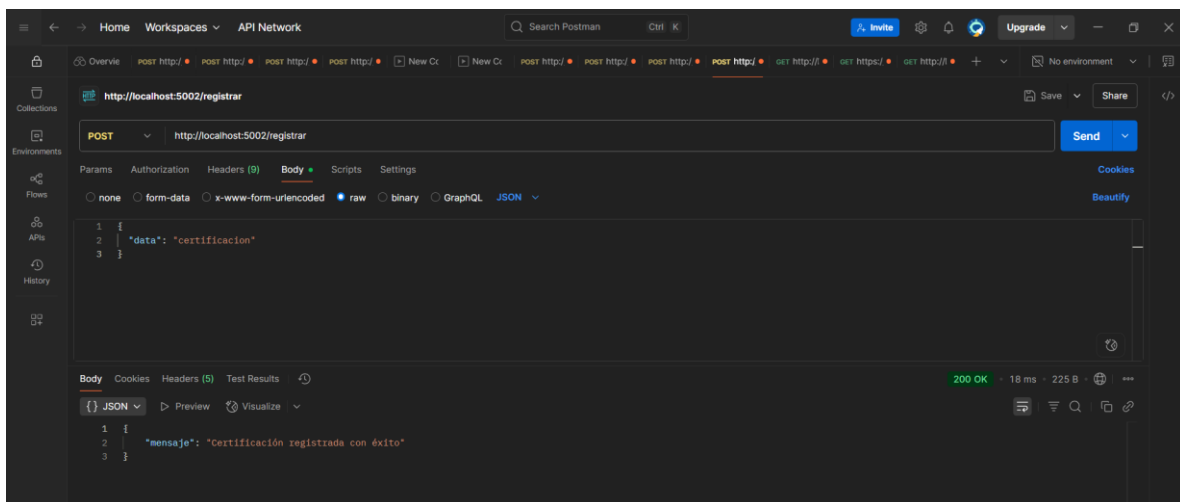
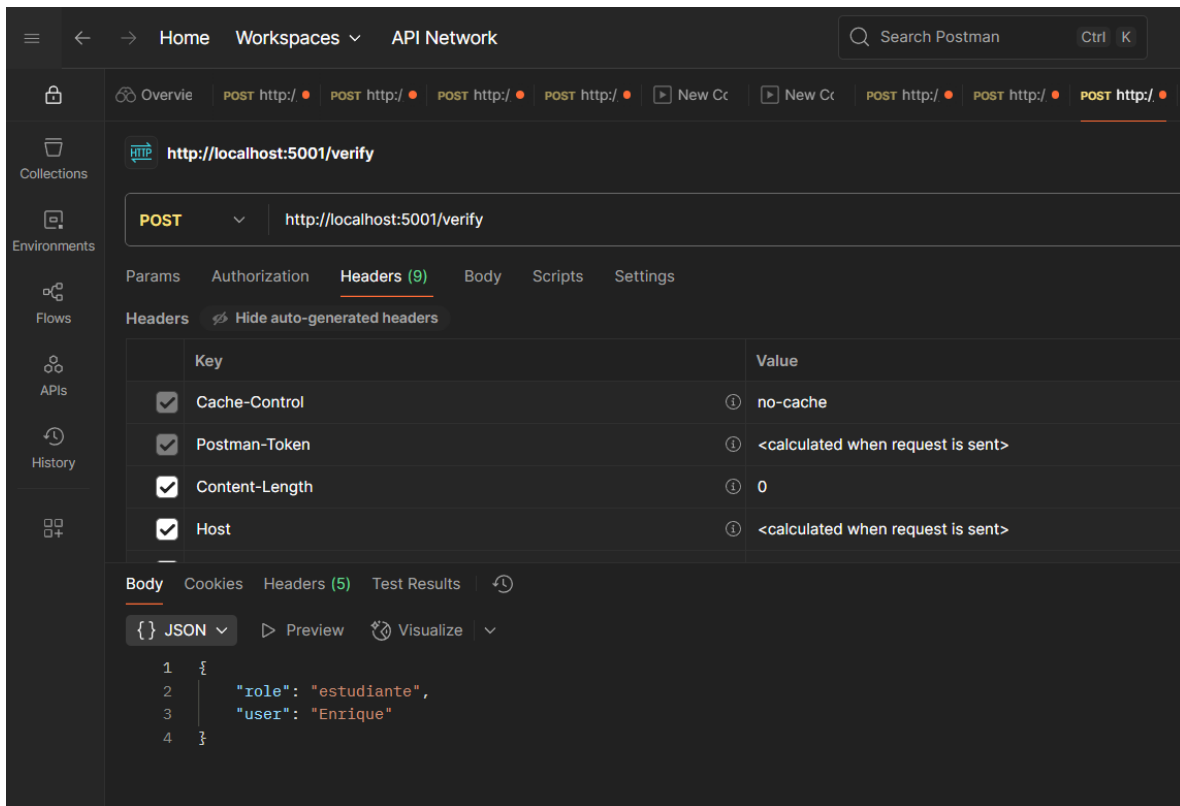
AuthService

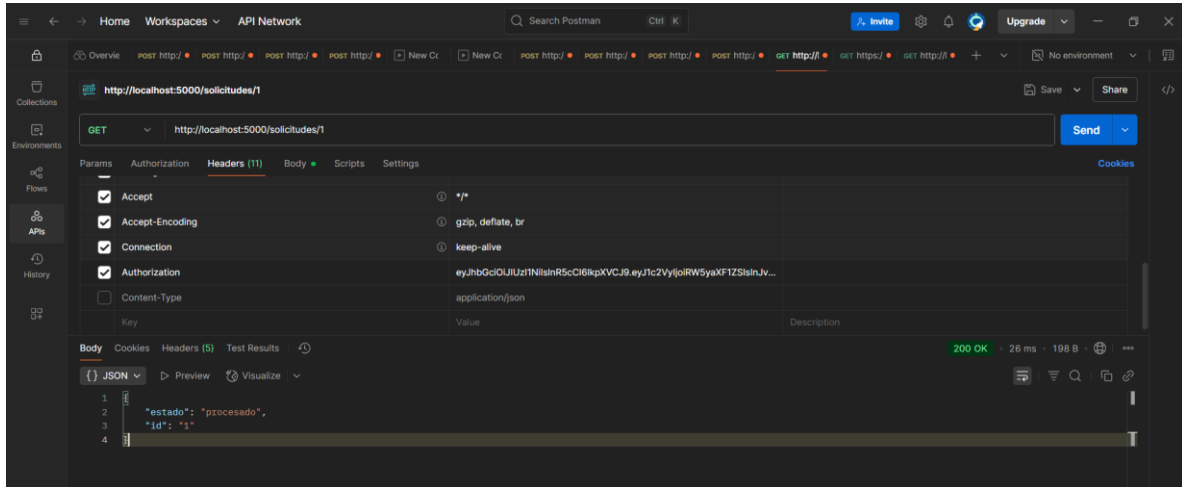
- Endpoint: POST /token para generar JWT.
- Endpoint: POST /verify para validar JWT.

CertificacionService

- Endpoint: POST /registrar
- Devuelve mensaje de éxito o error simulado.







4. Inicialización de ISTIO y Dockerizar servicios

- Paso 1: Iniciar Minikube con perfil personalizado**

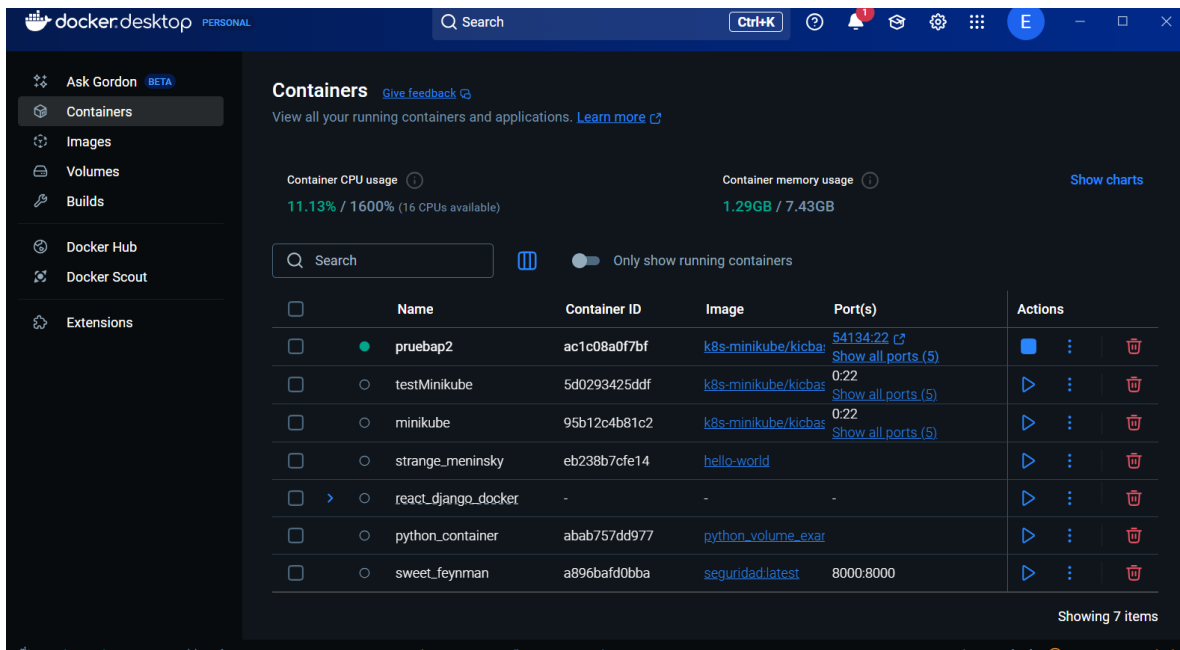
`minikube start -p pruebap2 --memory=6000 --cpus=4 --driver=Docker`

- Paso 2: Verificar y activar el perfil de Minikube**

`minikube profile list`

`minikube profile pruebap2`

`kubectl config current-context # Verifica el contexto actual`



- Paso 3: Instalar Istio con perfil demo**

`istioctl install --set profile=demo -y`

`kubectl label namespace default istio-injection=enabled`

```
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\emeri> istioctl install --set profile=demo -y

      _oo_
     ooo_
    ooo_oo
   ooo_oo_
  ooo_oo_oo
 ooo_oo_oo_
ooo_oo_oo_oo
 ooo_oo_oo_oo
  ooo_oo_oo_oo
   ooo_oo_oo_oo
    ooo_oo_oo_oo
     ooo_oo_oo_oo
      _oo_

✓ Istio core installed 🚩
✓ Istiod installed 🟢
✓ Egress gateways installed 🚢
✓ Ingress gateways installed 🚢
✓ Installation complete
PS C:\Users\emeri> kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

- **Paso 4: Construir imágenes Docker en entorno de Minikube**
& minikube -p pruebap2 docker-env | Invoke-Expression

```
cd path\to\auth-service
docker build -t auth-service .
```

```
cd ../certificacion-service
docker build -t certificacion-service .
```

```
cd ../solicitud-service
docker build -t solicitud-service
```

```
PS C:\INTEGRACION\PruebaP2_EM\solicitud-service> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
auth-service-64d98cc46c-zfhhs       2/2     Running   0           108m
certificacion-service-66767d59f6-m2khc 2/2     Running   0           113m
solicitud-service-85b5c48d8f-4npfp    2/2     Running   0           10m
PS C:\INTEGRACION\PruebaP2_EM\solicitud-service> |
```


- **Paso 5: Crear archivos YAML por cada servicio en carpeta K8/**

Se definieron archivos Deployment y Service para:

- **auth-service**
- **certificacion-service**
- **solicitud-service**

También se configuraron recursos Istio como:

- Gateway
- VirtualService
- DestinationRule

- **Paso 6: Aplicar los YAML**
kubectl apply -f K8/

Explicacion Configuracion Retry y Circuit Breaking con Istio

En el archivo istio-rules.yaml, se ha manejado la resiliencia de los microservicios mediante el uso de 2 recursos que son el Virtualservice y el Destination Rule.

```
retries:  
  attempts: 2  
  perTryTimeout: 2s  
  retryOn: gateway-error,  
  connect-failure,refused-stream,5xx
```

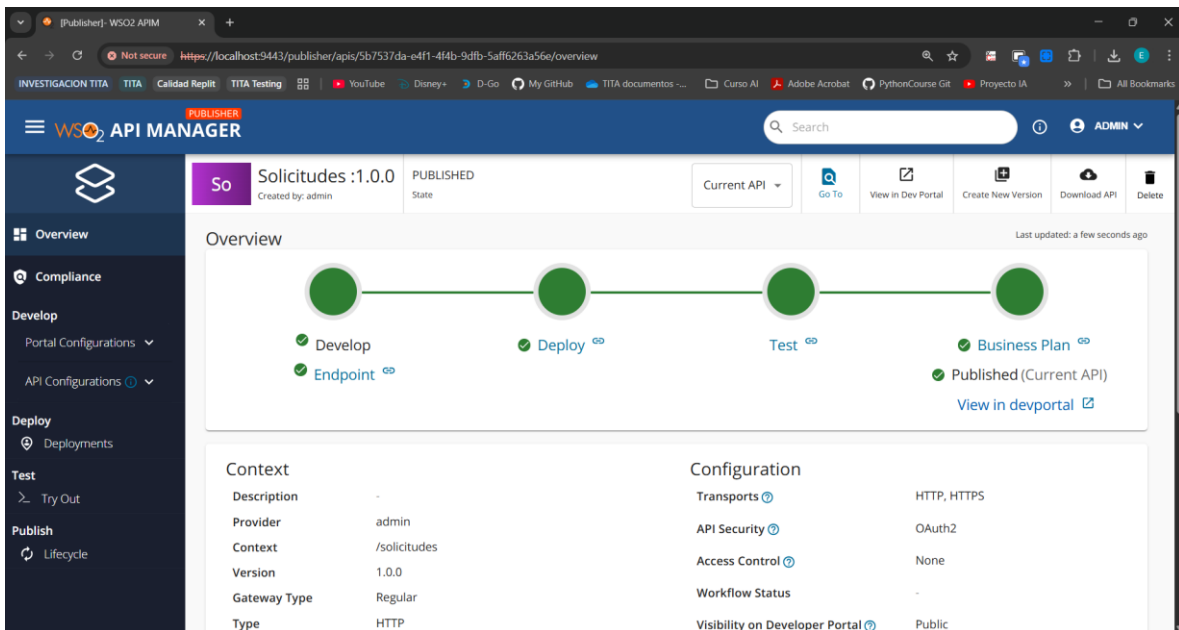
Esto permite reintentar hasta 2 veces si ocurre una falla transitoria del servicio SOAP.

```
outlierDetection:  
  consecutive5xxErrors: 3 # 3 errores  
  5xx seguidos  
  interval: 60s # evaluados cada 60  
  segundos  
  baseEjectionTime: 30s # tiempo de  
  espera tras "sacarlo"  
  maxEjectionPercent: 100
```

Si certification-service falla 3 veces en 60 segundos, Istio lo expulsa temporalmente por 30s.

5. API Gateway con WSO2

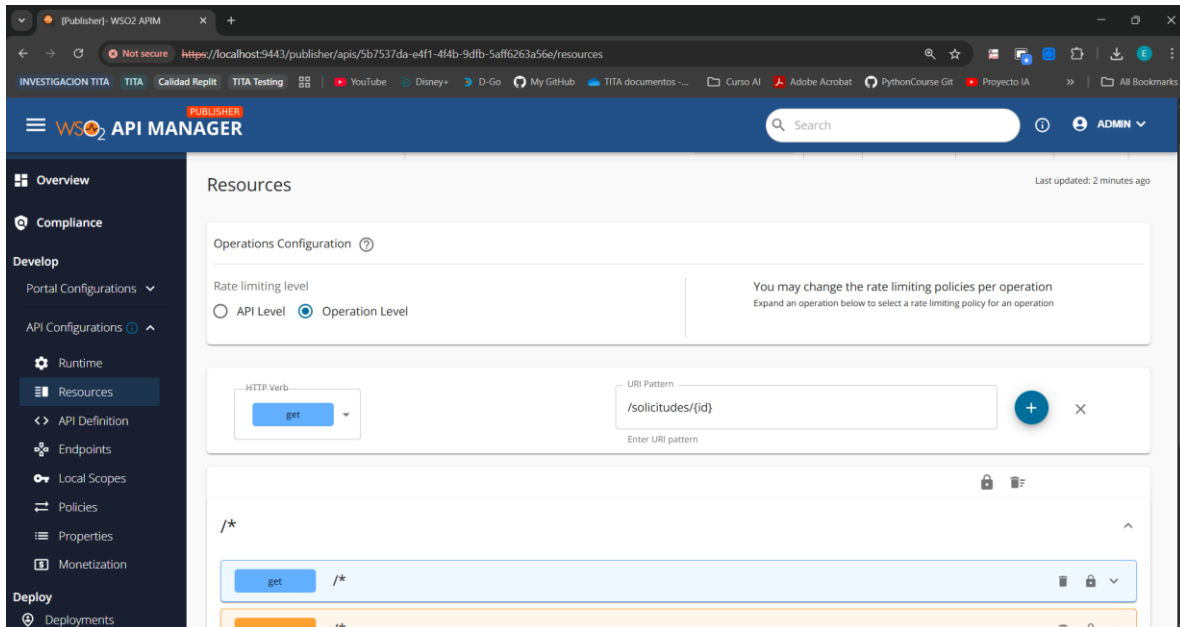
- Se registró el endpoint /solicitudes en WSO2 Publisher.
- Se realizó la configuración como se puede visualizar en las capturas
- Se suscribió al Default Application para poder generar test Keys
- Se generó un resource que sea el get/{id}
- Se probó el Api Gateway publicado a través de Postman como se puede ver en las capturas.



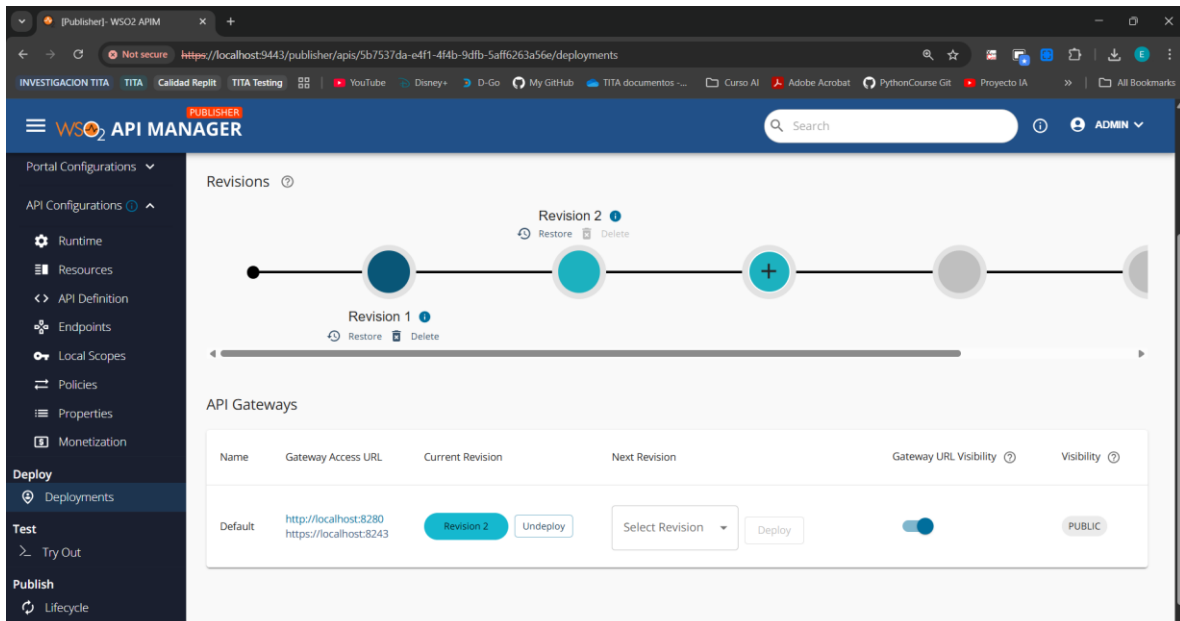
The screenshot displays the WSO2 API Manager Publisher interface. The top navigation bar includes the WSO2 logo, a search bar, and an 'ADMIN' dropdown. The left sidebar contains navigation links for Overview, Compliance, Develop, Deploy, Test, and Publish. The main content area shows the 'Solicitudes :1.0.0' API, created by 'admin', in a 'PUBLISHED' state. The 'Overview' section features a lifecycle diagram with four stages: Develop, Deploy, Test, and Business Plan, each with a green checkmark indicating success. Below this, the 'Context' and 'Configuration' sections provide detailed settings for the API.

Context	
Description	-
Provider	admin
Context	/solicitudes
Version	1.0.0
Gateway Type	Regular
Type	HTTP

Configuration	
Transports	HTTP, HTTPS
API Security	OAuth2
Access Control	None
Workflow Status	-
Visibility on Developer Portal	Public



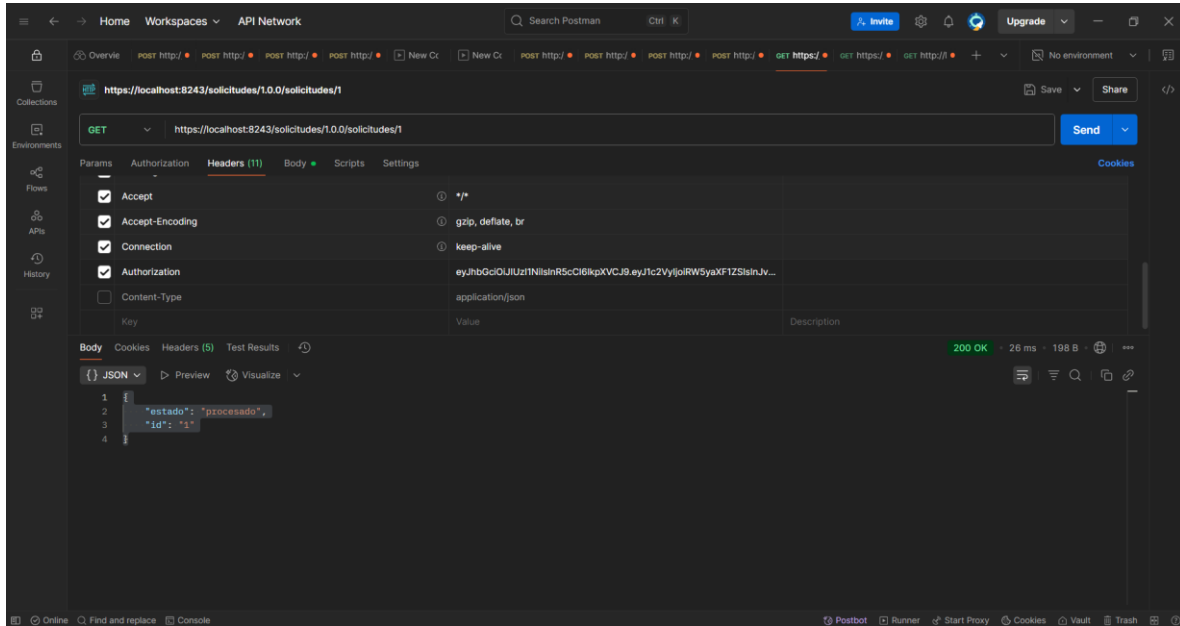
The screenshot shows the WSO2 API Manager Publisher interface. The left sidebar contains navigation options: Overview, Compliance, Develop (with sub-options: Portal Configurations, API Configurations, Runtime, Resources, API Definition, Endpoints, Local Scopes, Policies, Properties, Monetization), and Deploy (with sub-option: Deployments). The main content area is titled 'Resources' and shows the 'Operations Configuration' section. It includes a 'Rate limiting level' section with radio buttons for 'API Level' and 'Operation Level' (selected). Below this is a form to add a new resource, with fields for 'HTTP Verb' (set to 'get') and 'URI Pattern' (set to '/solicitudes/{id}'). A list of existing resources is shown below, with the first resource having a 'get' method and a '*' pattern.



The screenshot shows the WSO2 API Manager Publisher interface, specifically the 'Revisions' and 'API Gateways' sections. The 'Revisions' section displays a timeline with two revisions: 'Revision 1' and 'Revision 2'. 'Revision 2' is the current revision. Below the timeline is a table for 'API Gateways'.

Name	Gateway Access URL	Current Revision	Next Revision	Gateway URL Visibility	Visibility
Default	http://localhost:8280 https://localhost:8243	Revision 2	Undeploy	Select Revision	Deploy

The 'API Gateways' section also includes a 'Gateway URL Visibility' toggle (set to 'ON') and a 'Visibility' dropdown (set to 'PUBLIC').



6. Observabilidad y Trazabilidad

Para verificar la observabilidad y trazabilidad del sistema, se accede a la interfaz de Prometheus desplegada en Minikube para consultar métricas como `istio_requests_total`, `istio_request_duration_seconds` y `istio_requests_errors_total`, las cuales permiten analizar el comportamiento del tráfico, la latencia y los errores por servicio. Con Kiali se visualiza gráficamente el flujo de peticiones entre microservicios, identificando cuellos de botella o rutas fallidas. Finalmente, Jaeger permite rastrear solicitudes individuales, observando cada salto entre servicios y midiendo el tiempo de ejecución en cada componente del sistema. Estas herramientas en conjunto facilitan la detección de errores, el monitoreo en tiempo real y el análisis de performance.

Git: <https://github.com/CHACHO617/PruebaP2Integracion>