

PROYECTO GENOSENTINEL

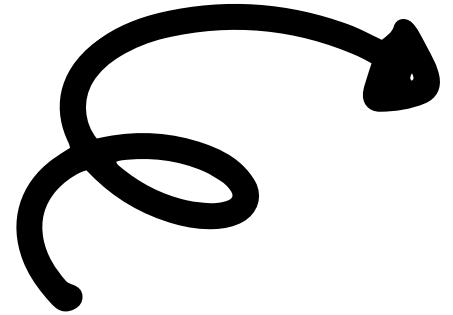
SISTEMA DE GESTIÓN CLÍNICA Y ANÁLISIS GENÓMICO

ESTUDIANTES:SANTIAGO RUBIO-JACOBO ARROYAVE

PROFESOR: SIMON GAVIRIA

ASIGNATURA: PROGRAMACION BACKEND

UNIVERSIDAD AUTONOMA DE MANIZALES



INTRODUCCIÓN

Este proyecto integra los microservicios de Autenticación, Clínica y Genómica, cada uno con funciones independientes pero conectados mediante un flujo de datos seguro y orquestado en Kubernetes.



MICROSERVICIO SPRING BOOT - AUTENTICACIÓN (JAVA)

1. EL CLIENTE ENVÍA CREDENCIALES

→ Usuario y contraseña llegan al endpoint /auth/login.

2. VALIDACIÓN

→ Spring verifica en la base de datos si el usuario existe y si la contraseña coincide.

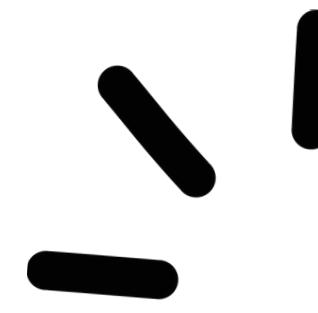
3. GENERACIÓN DE TOKEN JWT

→ El microservicio crea un token que funciona como “pasaporte digital”.

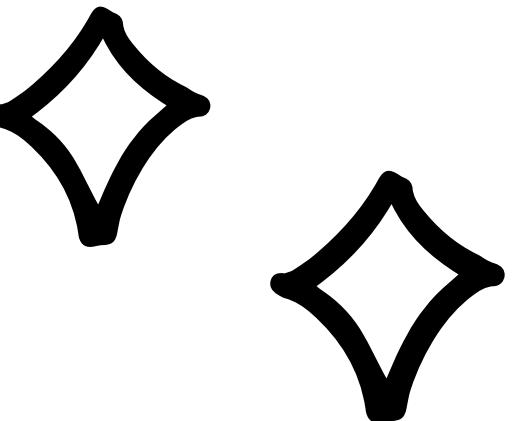
4. RESPUESTA

→ Devuelve el JWT al frontend o al otro microservicio que lo necesite.





MICROSERVICIO NESTJS - GESTIÓN CLÍNICA (NODE.JS)



1. EL CLIENTE O DJANGO ENVÍA REQUEST

→ Incluye el JWT recibido del microservicio de Spring.

2. NEST VERIFICA SI EL TOKEN ES VÁLIDO

→ Realiza la autentificacion y validacion antes del controlador.

3. PROCESA LA OPERACIÓN

Ejemplo:

- Crear paciente
- Consulta un tumor
- Actualiza historia clinica

4. RESPUESTA

→ Devuelve los datos solicitados en formato JSON.





MICROSERVICIO DJANGO - ANALYTICS / REPORTES (PYTHON)

1. DJANGO RECIBE SOLICITUDES DESDE NEST O SPRINGBOOT

→ La API de Django captura la petición y la prepara para el servicio.

2. LA VISTA LLAMA AL SERVICIO DE DOMINIO

- Allí se valida la lógica y se coordinan los pasos necesarios.
- Si hace falta, consulta datos al microservicio de Clínica.

3. ACCEDER A LA BASE DE DATOS MEDIANTE EL ORM

→ Crea, actualiza o consulta registros según el flujo.

4. PROCESA LA INFORMACIÓN

→ Aplica reglas, transforma datos y construye los DTOs de salida.

5. RESPONDE AL CLIENTE

- Retorna un JSON estructurado o el archivo generado (Excel, PDF, etc.).
- Los errores pasan por el handler global para mantener respuestas consistentes.



COMUNICACIÓN ENTRE LOS MICROSERVICIOS

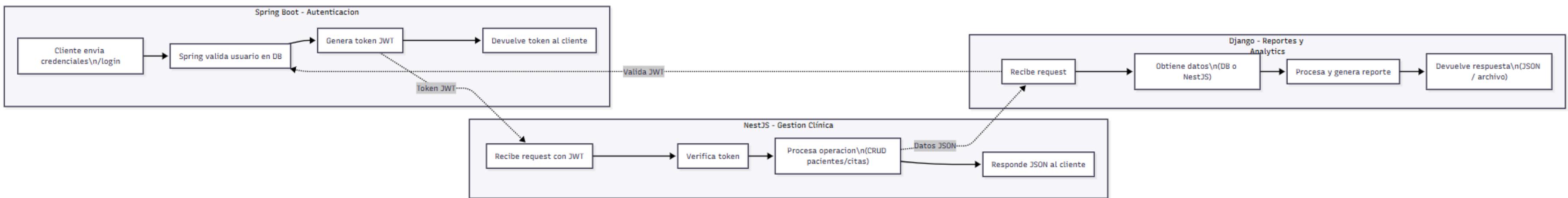
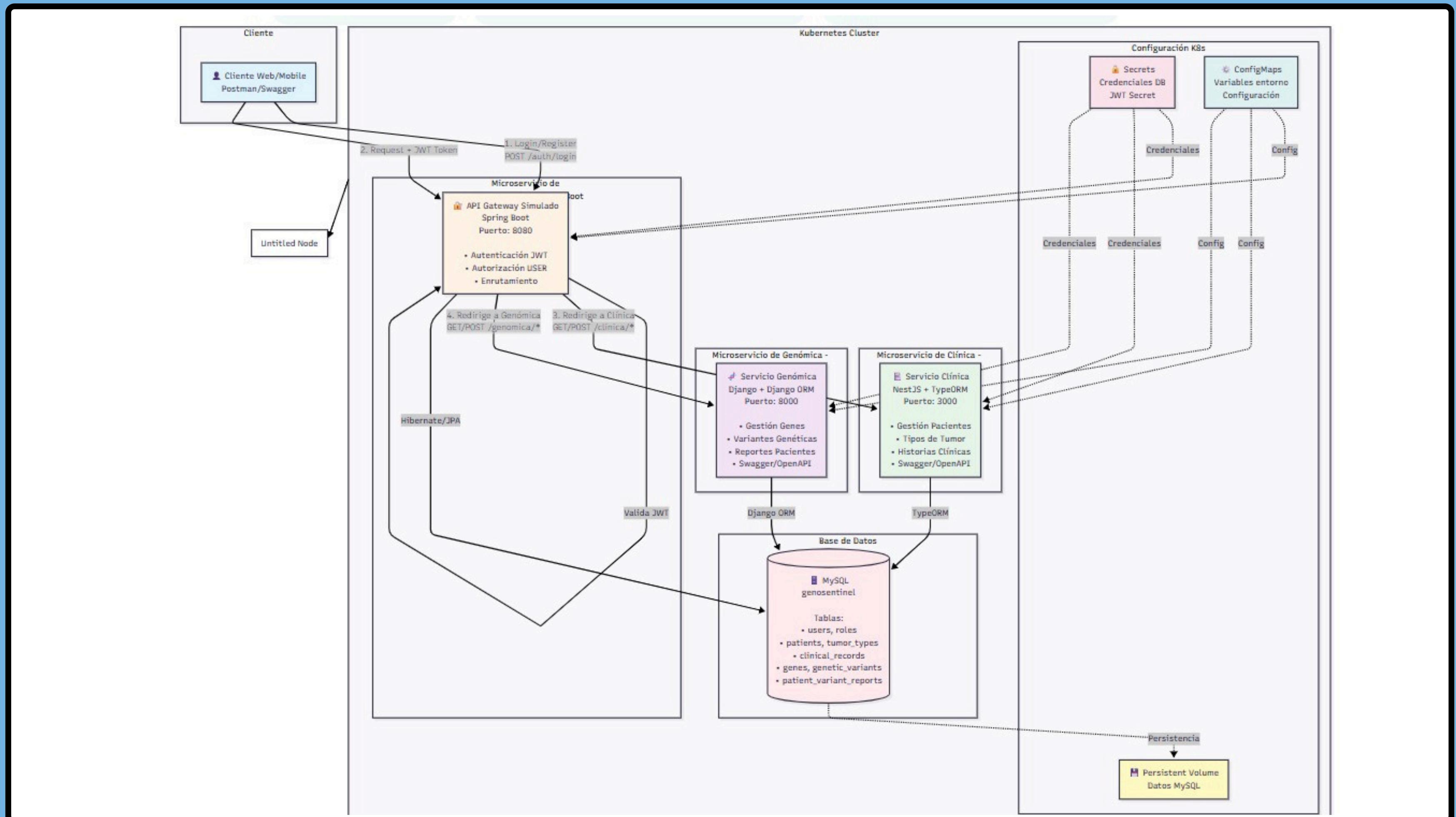
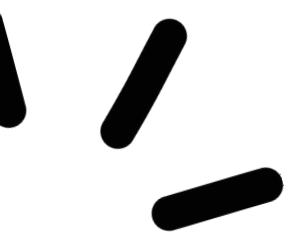


DIAGRAMA DE ARQUITECTURA



COMANDOS USADOS PARA LA CREACION DE KUBERNETES



1. Abrir Docker desktop, esperar a que corra.

2. **Instalar minikube:**

<https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe>

3. **Verificar que esté instalado:** C:\Users\arroy>minikube version
minikube version: v1.37.0 commit:
65318f4cff9c12cc87ec9eb8f4cdd57b25047f3

4. **Iniciar Minikube usando Docker como backend:** minikube start --driver=docker

5. **Navegar al directorio del microservicio y construir la imagen** cd nombre microservicio
docker build -t nombre-imagen:latest .
cd .. (devolverse a la raíz del proyecto, repo en github)

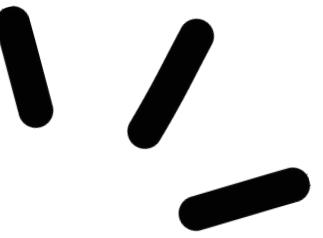
6. **Verificar que la imagen se creó** docker images

7. **Aplicar configuraciones de Kubernetes - MySQL primero**
kubectl apply -f k8s/mysql/



kubernetes

COMANDOS USADOS PARA LA CREACION DE KUBERNETES



8. Aplicar configuraciones del microservicio deseado:

```
kubectl apply -f k8s/nombre-dentrode-k8s/
```

9. Verificar que los pods estén corriendo:

```
kubectl get pods
```

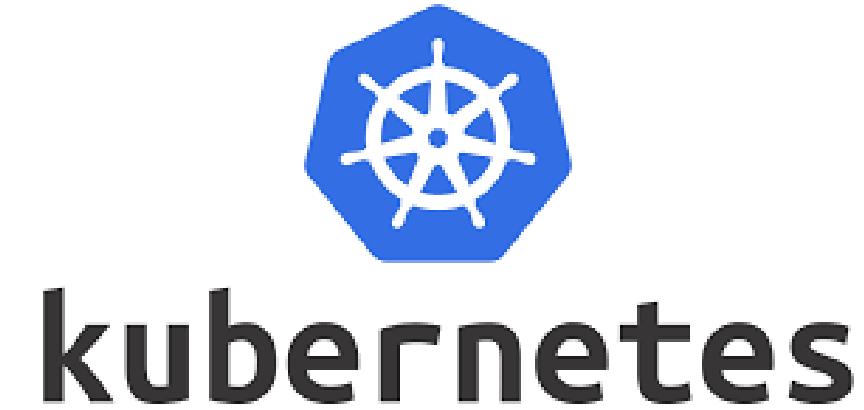
10. Obtener la URL del servicio:

```
minikube service nombre-servicio --url
```

11. Ver logs de un pod específico : (reemplaza <nombre-pod> con el nombre real)

```
kubectl logs <nombre-pod>
```

12. Sirve para reiniciar los pods una vez se hagan cambios en el código y se hayan levantado las imágenes de nuevo: **kubectl rollout restart deployment nombre-deployment**



MUCHAS GRACIAS

www.unsitiogenial.es