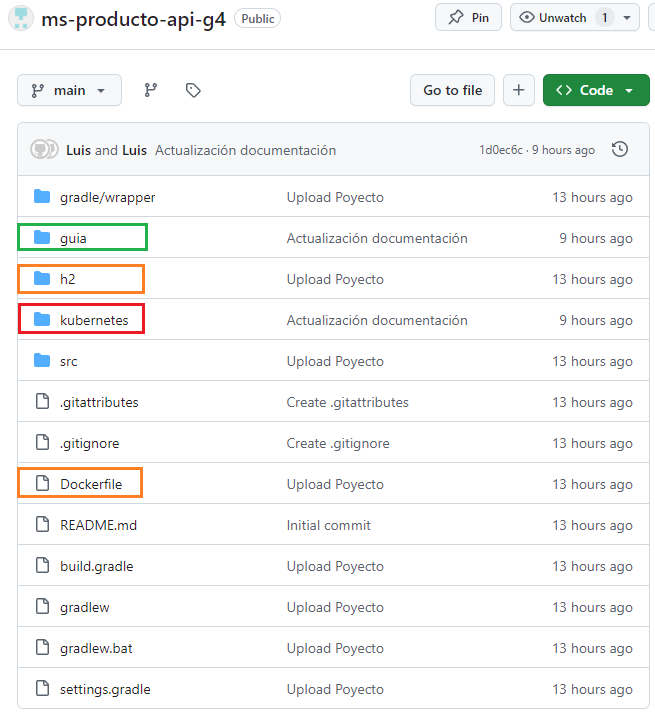
**GUÍA DE DESPLIEGUE DE UNA APLICACIÓN SPRING BOOT EN KUBERNETES CON DOCKER Y KILLERCODA**

Exponer una aplicación Spring Boot en Kubernetes utilizando NodePort, incluyendo el acceso a múltiples réplicas. Para el despliegue usaremos la plataforma Killercoda. Se configurará un Ingress para exponer la aplicación.

1. **Estructura del Repositorio del proyecto**

<https://github.com/Anducias/ms-producto-api-g4.git>

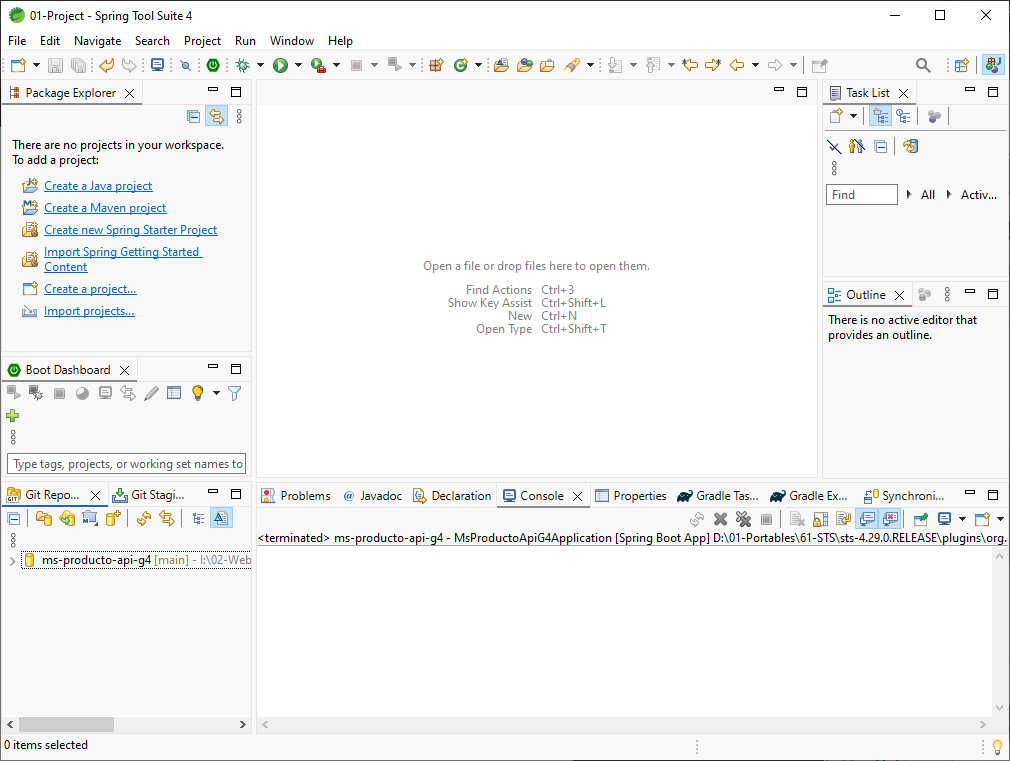


Adicional a las fuentes del proyecto se encuentran también indicadas los objetos relacionados a la generación de las imágenes vía Docker (enmarcados de color naranja), los objetos relacionados a las acciones desde kubernetes (enmarcados de color rojo) y la documentación (guía y acceso a imágenes) de todo el proyecto (enmarcado de color verde)

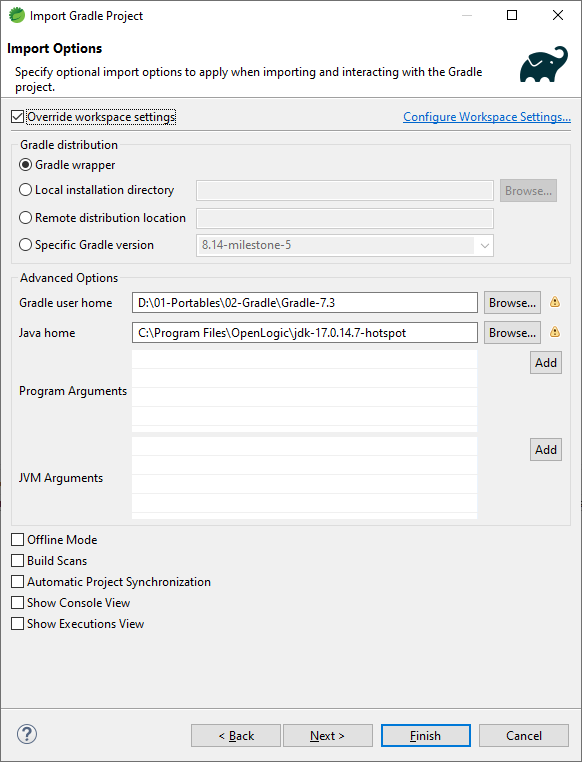
1. **Generación y Prueba de Microservicio de forma local**

Clonamos el repositorio de GitHub indicado en el punto 1

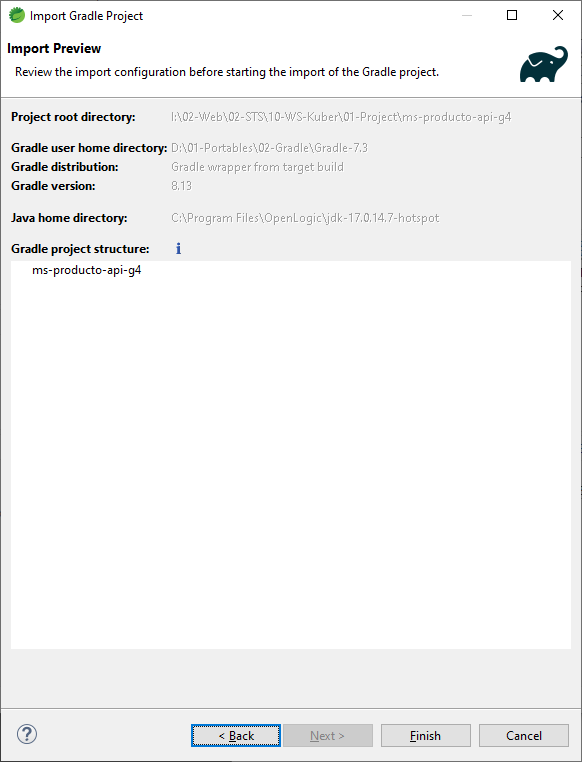
git clone https://github.com/Anducias/ms-producto-api-g4.git



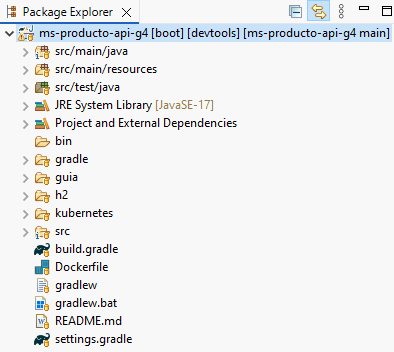
Importamos el proyecto Gradle



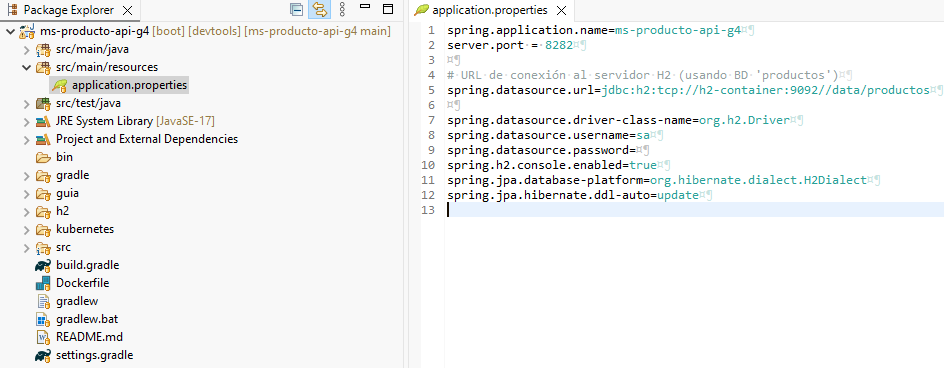
Verificamos que se importa sin problemas

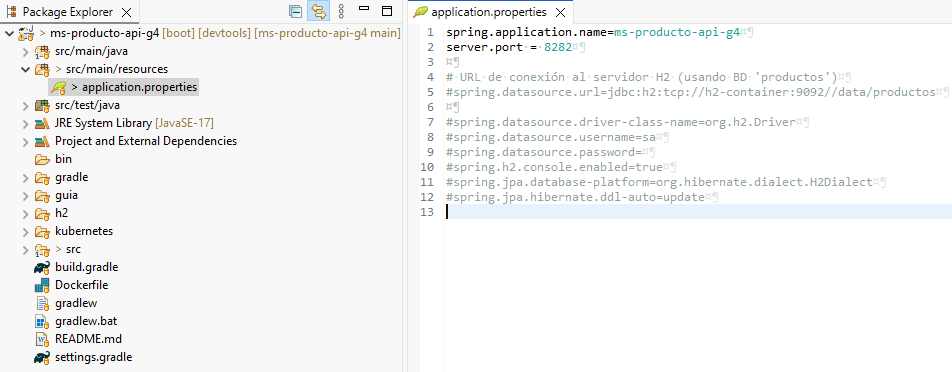


Podremos apreciar la estructura del proyecto

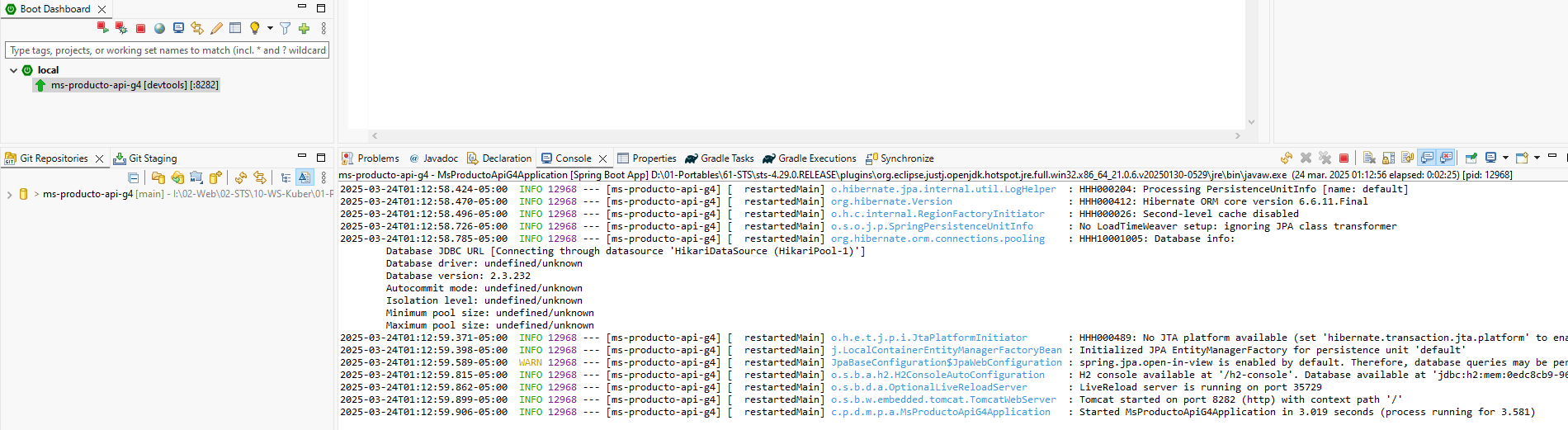


Para las pruebas de manera local comentaremos en el archivo **‘application.properties’** la configuración con la BF H2.



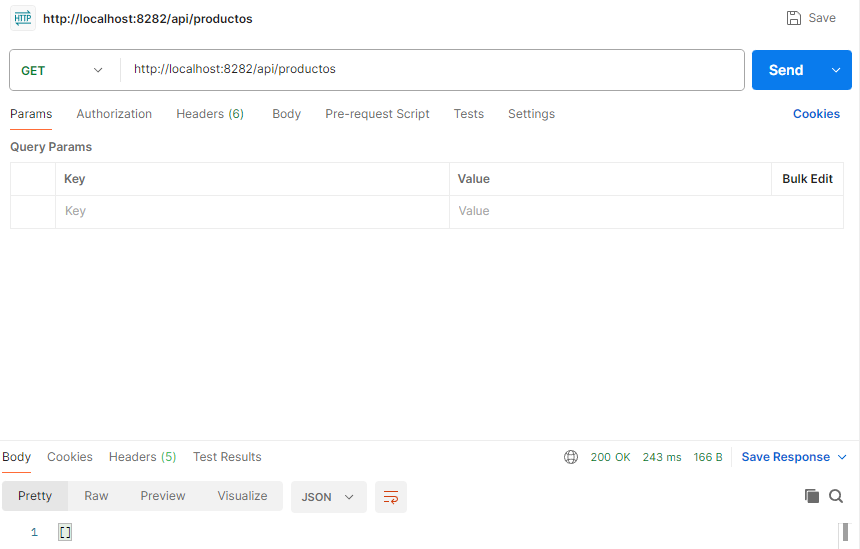


Procederemos a ejecutar el proyecto con SpringBoot

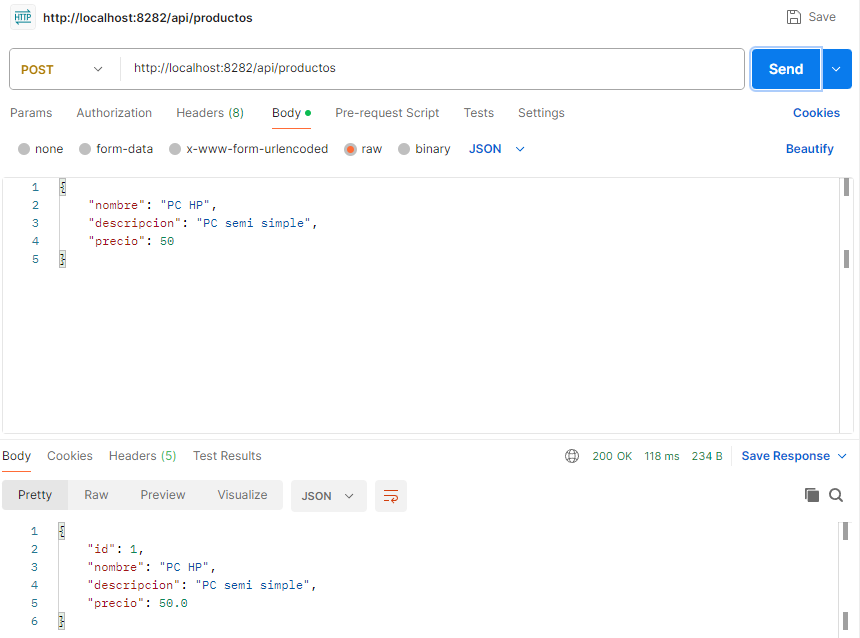


Realizamos las pruebas CRUD de manera local con el puerto indicado en el archivo de configuración

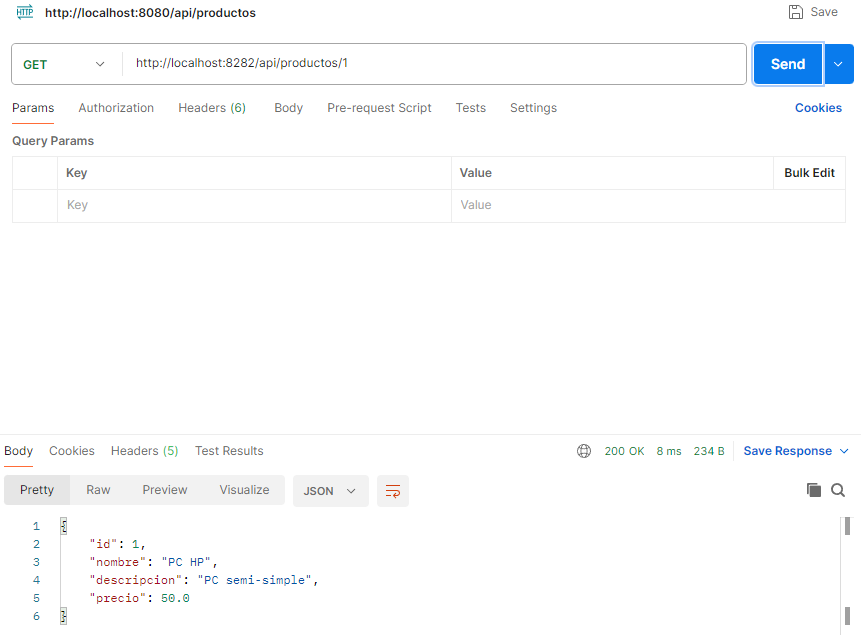
*Listar Productos*



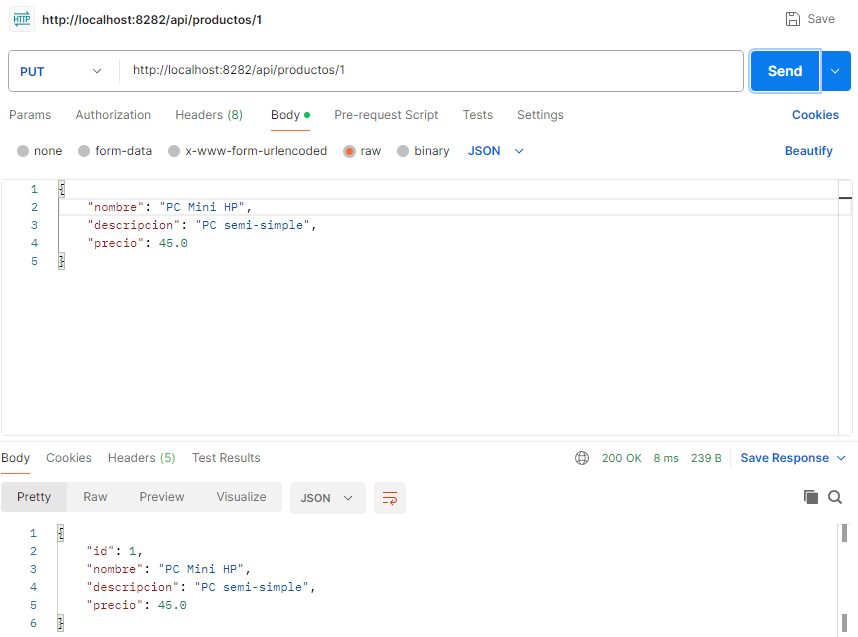
*Registrar Producto*



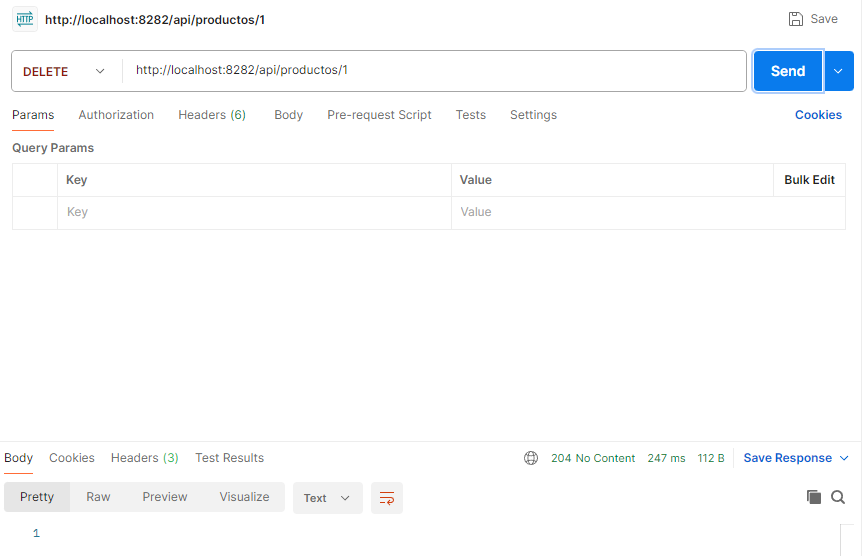
*Obtener Producto*



*Actualizar Producto*

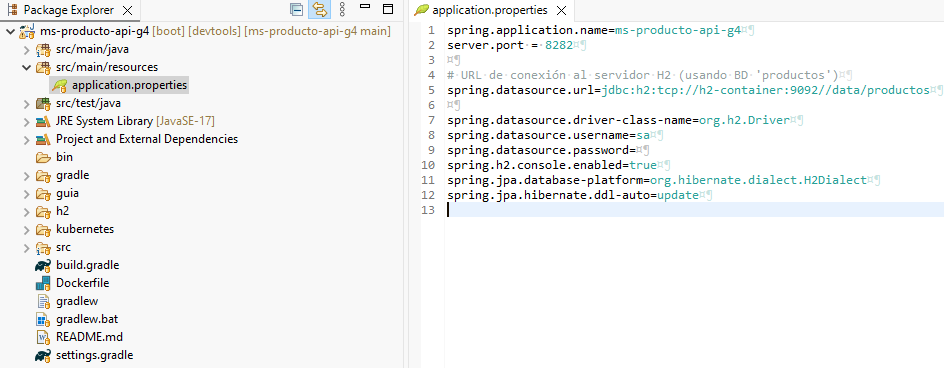


*Eliminar Producto*



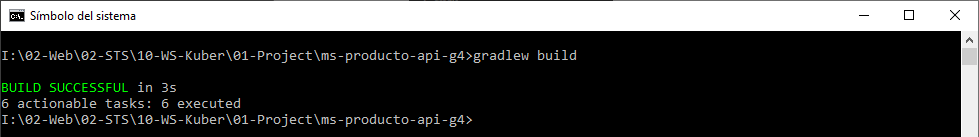
1. **Generando el JAR del proyecto y de imágenes para almacenar en Docker Registry**

Para generar el jar revertiremos los comentarios en la etapa previa (ya que el microservicio se conectará a la BD H2)

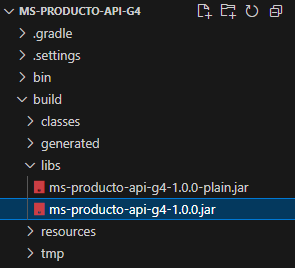


Procederemos a generar el jar

gradlew build



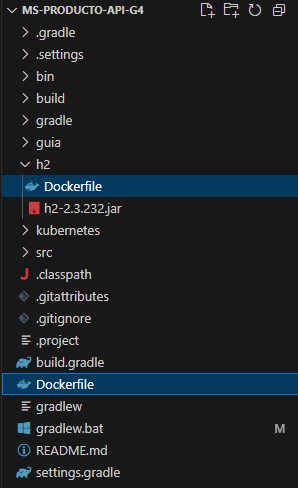
Verificamos efectivamente que se generó el jar sin problemas



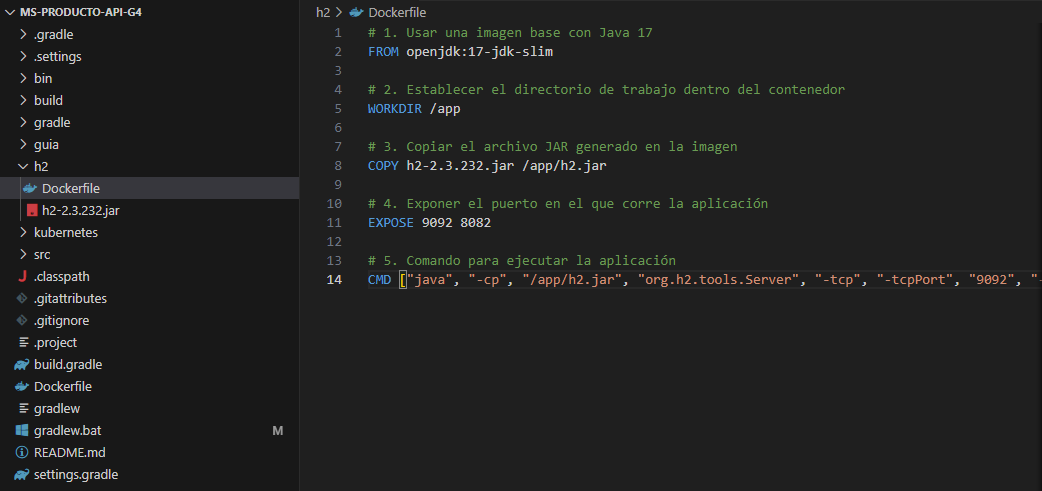
Nos conectamos a Docker (iniciamos Docker Desktop)

docker login

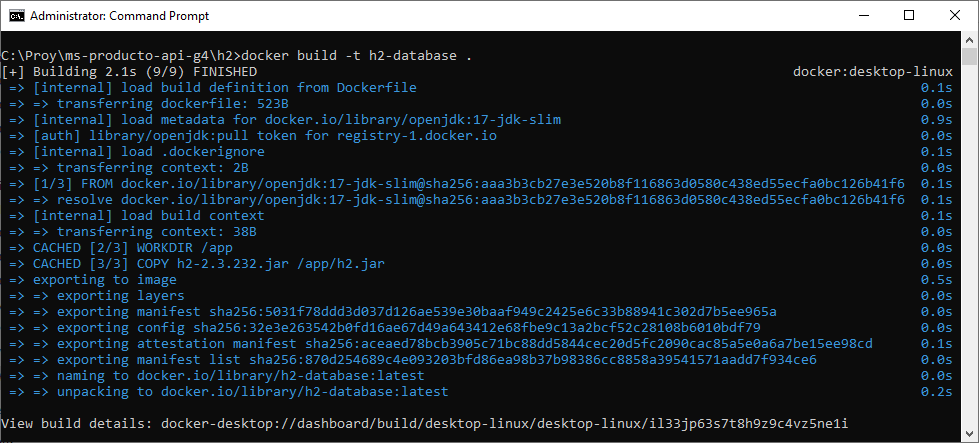
Para el proyecto generaremos 2 imágenes (uno relacionado con la BD H2 y otra al propio microservicio que se conectará a la BD H2)



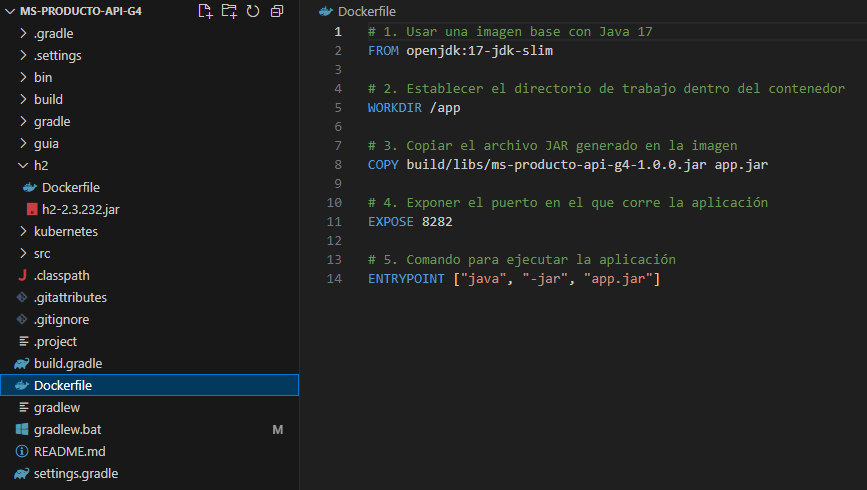
Primero generaremos la imagen de la BD H2, la cual se encuentra en la carpeta h2 del proyecto, para ello nos ubicaremos dentro de esa carpeta y ejecutaremos el comando respectivo.



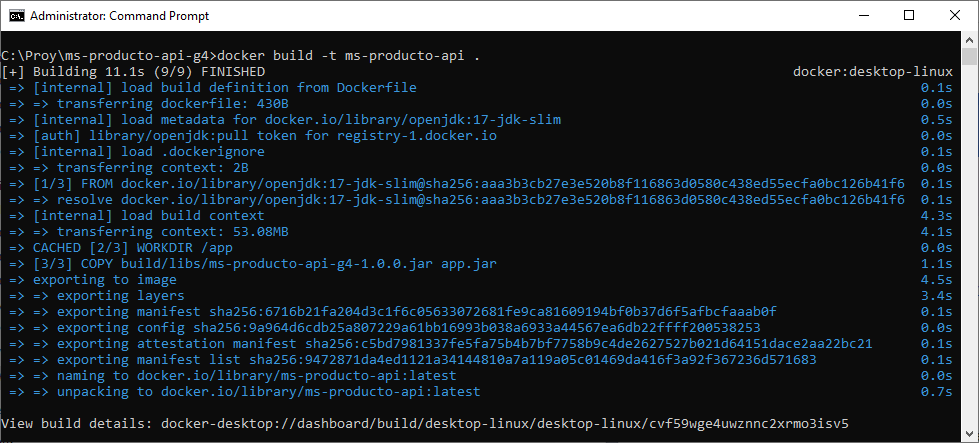
docker build -t h2-database .



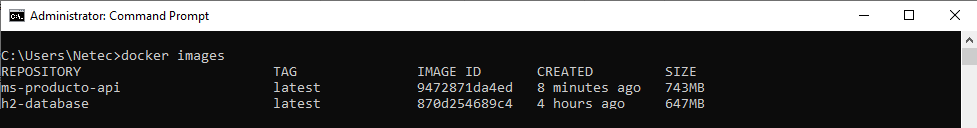
Luego generaremos la imagen del microservicio, la cual se encuentra en la carpeta raíz del proyecto, para ello regresaremos a la carpeta raíz y ejecutaremos el comando respectivo.



docker build -t ms-producto-api .



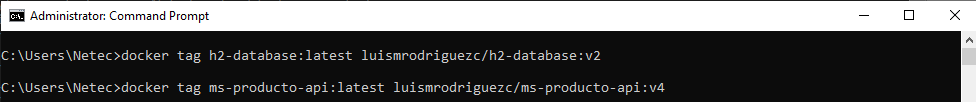
Verificamos las imágenes generadas



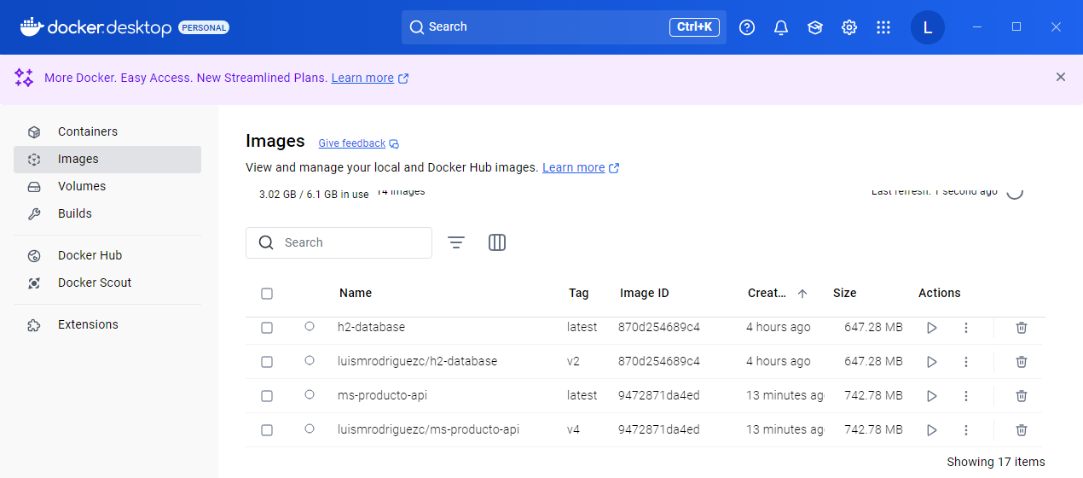
Realizamos la identificación única de las imágenes en Docker Registry

docker tag h2-database:latest luismrodriguezc/h2-database:v2

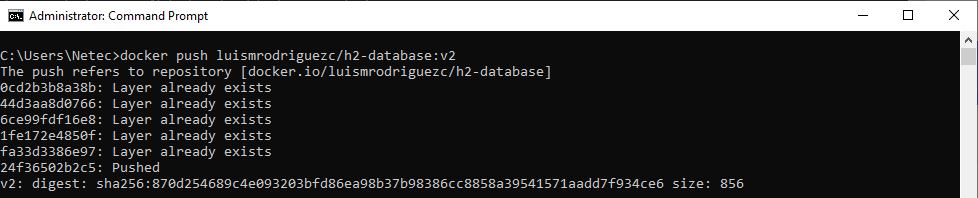
docker tag ms-producto-api:latest luismrodriguezc/ms-producto-api:v4

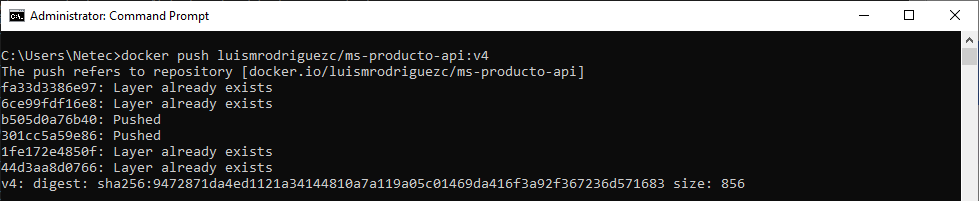


Verificamos ello en Docker Desktop

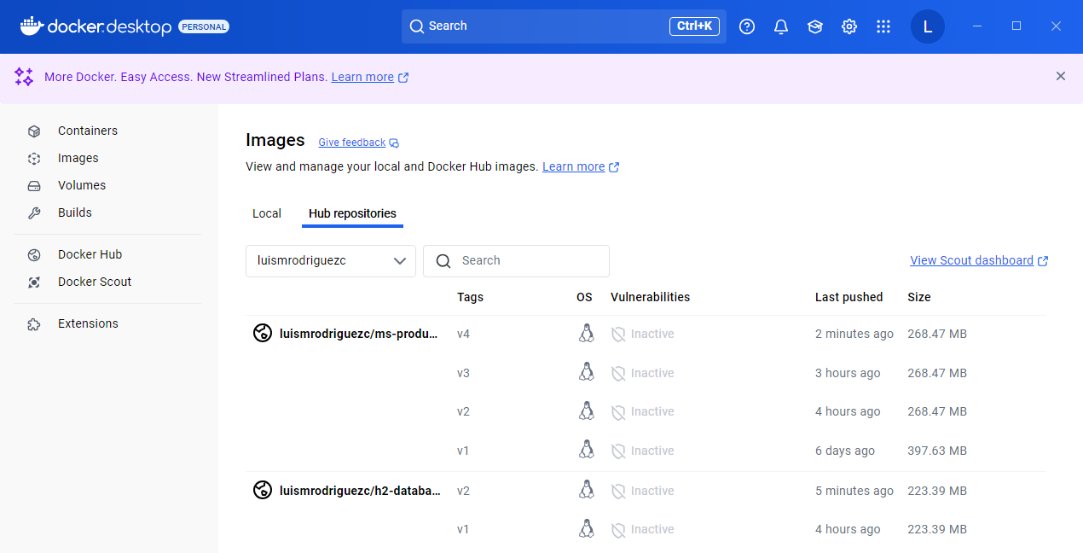


Posteriormente realizaremos el push respectivo para que las sean de acceso público



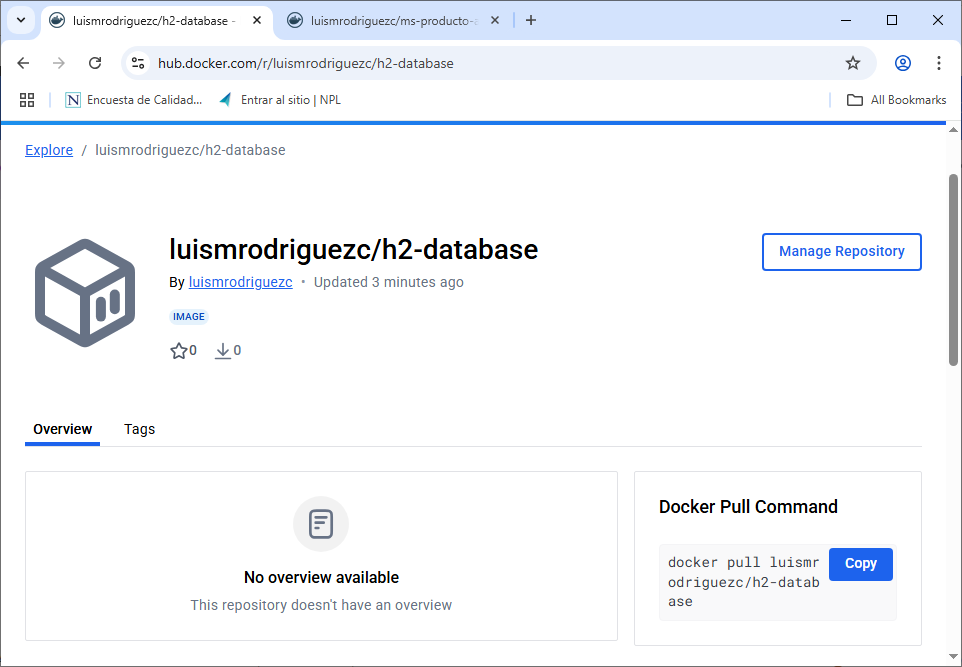


Verificamos ello en Docker Desktop

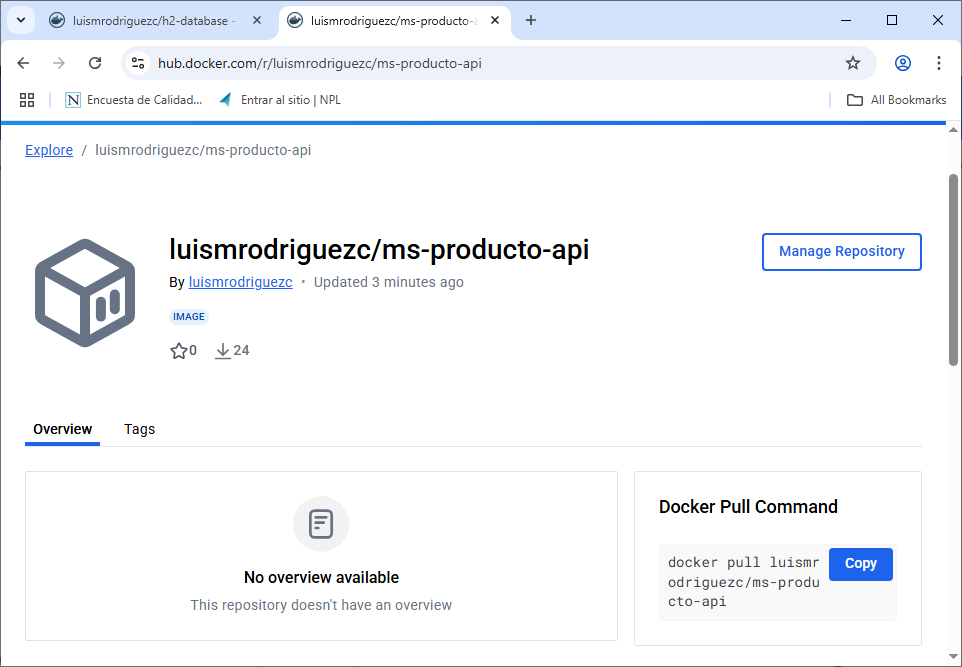


Una vez publicada procederemos a acceder a sus detalles para poder obtener el acceso libre de las imágenes desde DockerHub.

<https://hub.docker.com/r/luismrodriguezc/h2-database>



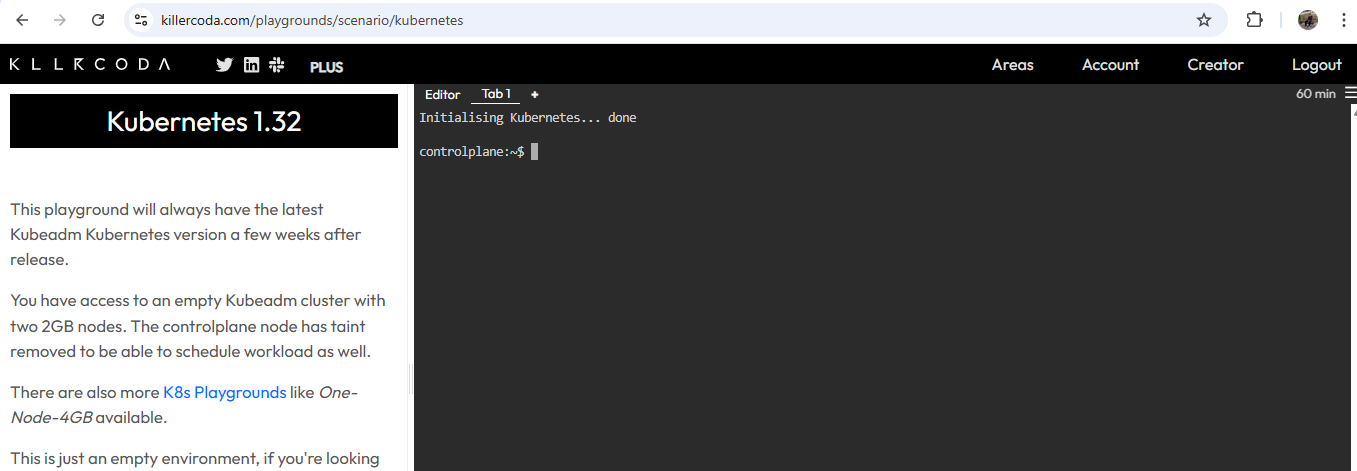
<https://hub.docker.com/r/luismrodriguezc/ms-producto-api>



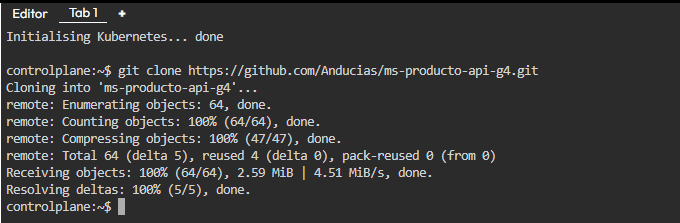
1. **Desplegando el Proyecto desde Killercoda**

Accederemos a Killercoda y clonaremos el repositorio de GitHub:

<https://killercoda.com/playgrounds/scenario/kubernetes>



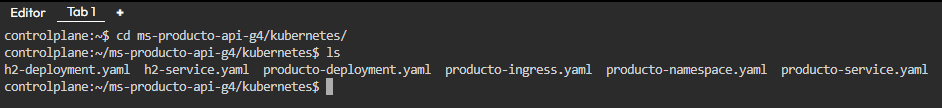
git clone https://github.com/Anducias/ms-producto-api-g4.git



Accedemos a la carpeta kubernetes y vemos su contenido

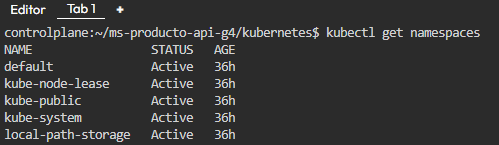
cd ms-producto-api-g4/kubernetes/

ls



De la información visualizada, procederemos a crear el namespace, pods y services, pero antes veremos si existen objetos previamente

kubectl get namespaces



Verificamos que el namespace que vamos a crear no existe (lo cual es lógico), por lo que procederemos a generar tanto el namespace, pods y services

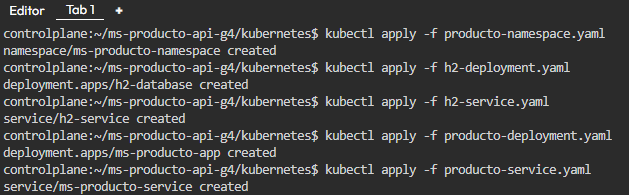
kubectl apply -f producto-namespace.yaml

kubectl apply -f h2-deployment.yaml

kubectl apply -f h2-service.yaml

kubectl apply -f producto-deployment.yaml

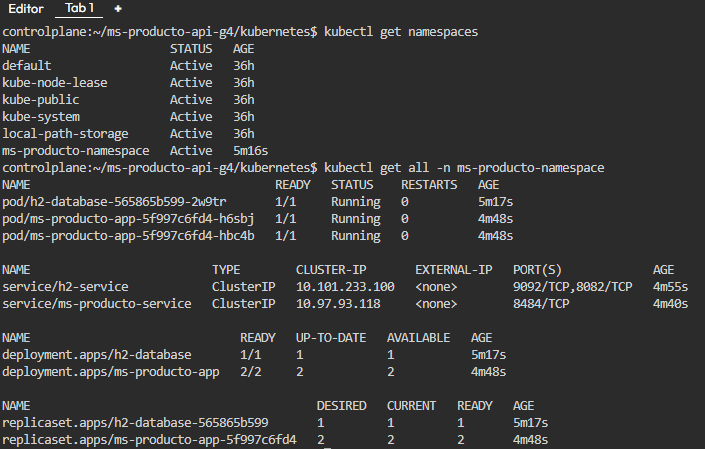
kubectl apply -f producto-service.yaml



Verificaremos los objetos creados

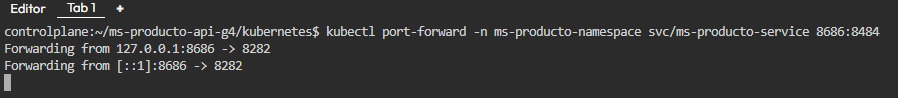
kubectl get namespaces

kubectl get all -n ms-producto-namespace



Procederemos a realizar la validación del correcto funcionamiento del service

kubectl port-forward svc/ms-producto-service 8686:8484 -n ms-producto-namespace



Abriremos otra pestaña y ejecutaremos los comandos respectivos (reemplazar IP\_SERVICIO y PORT por la información devuelta en el paso anterior).

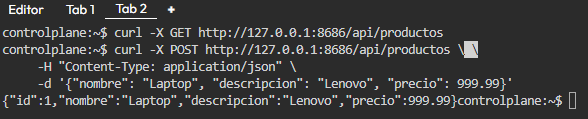
curl -X GET http://<IP\_SERVICIO>:<PORT>/api/productos

curl -X POST http://<IP\_SERVICIO>:<PORT>/api/productos \

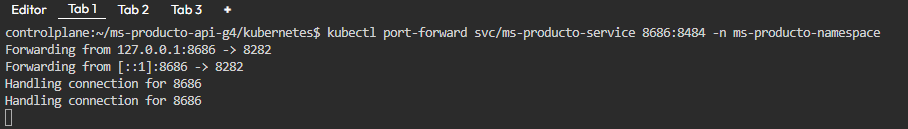
-H "Content-Type: application/json" \

-d '{"nombre": "Laptop", "descripcion": "Lenovo", "precio": 999.99}'

Donde los resultados deben ser en el primer caso un valor vacío y en el segundo la ejecución de la inserción

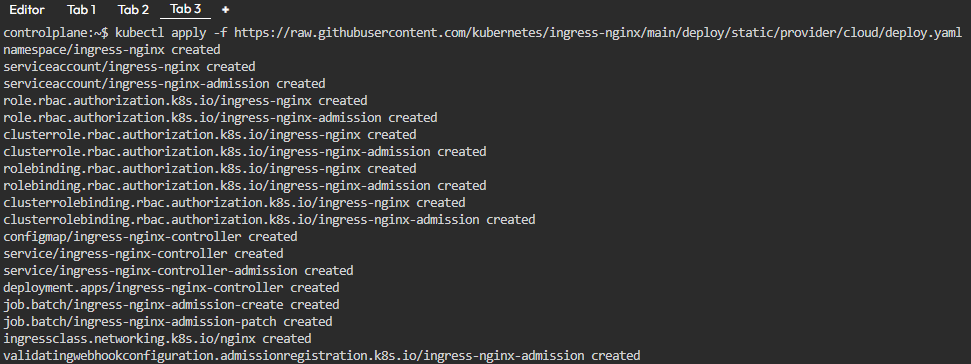


Si regresamos a la pestaña anterior, observaremos las respuestas relacionadas a las acciones CRUD



Ahora abriremos otra pestaña y procederemos a instalar el NGINX Ingress Controller

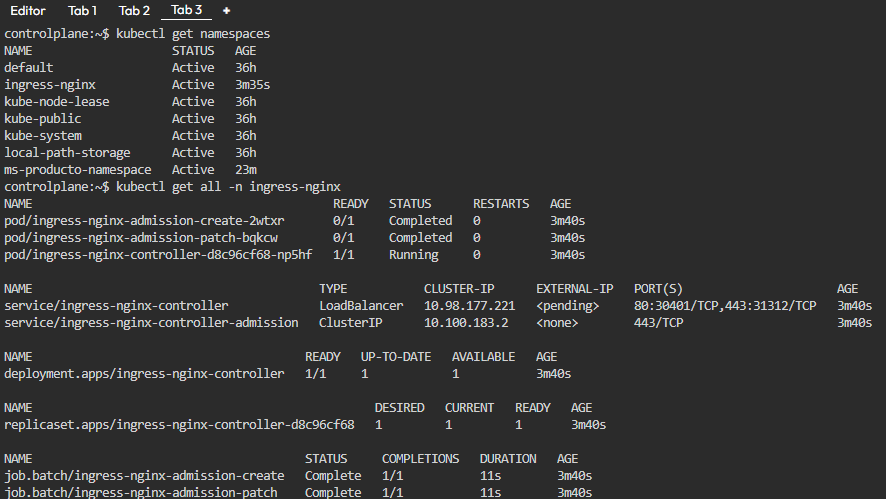
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml



Verificaremos que los pods y services estén activos y en ejecución

kubectl get namespaces

kubectl get all -n ingress-nginx



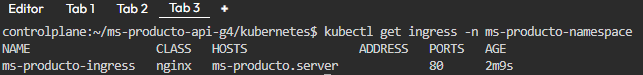
Luego procederemos a instalar el último archivo del proyecto, es decir, crearemos el ingress del proyecto

kubectl apply -f producto-ingress.yaml



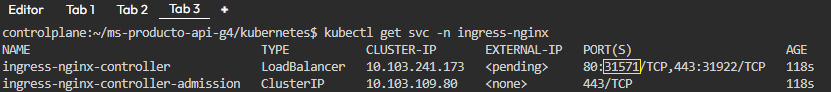
Verificamos si se ha generado correctamente

kubectl get ingress -n ms-producto-namespace



Ahora para continuar con las pruebas de las operaciones CRUD, necesitaremos el puerto que expone el ingress, por lo cual lo consultaremos

kubectl get svc -n ingress-nginx



Abriremos otra pestaña y ejecutaremos los comandos respectivos para el CRUD (reemplazar PORT por la información devuelta en el paso anterior).

Listar Productos

*curl -H "Host: ms-producto.server" http://localhost:<*PORT*>/api/productos*

Agregar Producto

*curl -X POST -H "Host: ms-producto.server" -H "Content-Type: application/json" \*

*-d '{"nombre": "Laptop", "descripcion": "Lenovo Yoga", "precio": 1000}' \*

*http://localhost:<*PORT*>/api/productos*

Obtener Producto

*curl -H "Host: ms-producto.server" http://localhost:<*PORT*>/api/productos/2*

Actualizar Producto

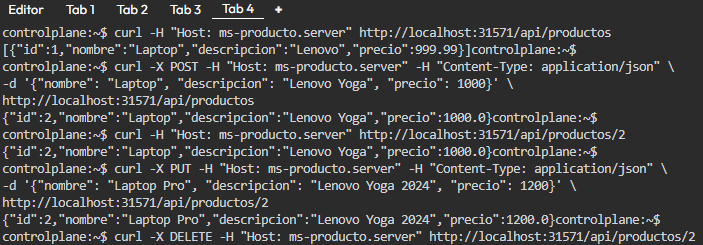
*curl -X PUT -H "Host: ms-producto.server" -H "Content-Type: application/json" \*

*-d '{"nombre": "Laptop Pro", "descripcion": "Lenovo Yoga 2024", "precio": 1200}' \*

*http://localhost:<*PORT*>/api/productos/2*

Eliminar Prodcuto

curl -X DELETE -H "Host: ms-producto.server" http://localhost*:<*PORT*>/*api/productos/2



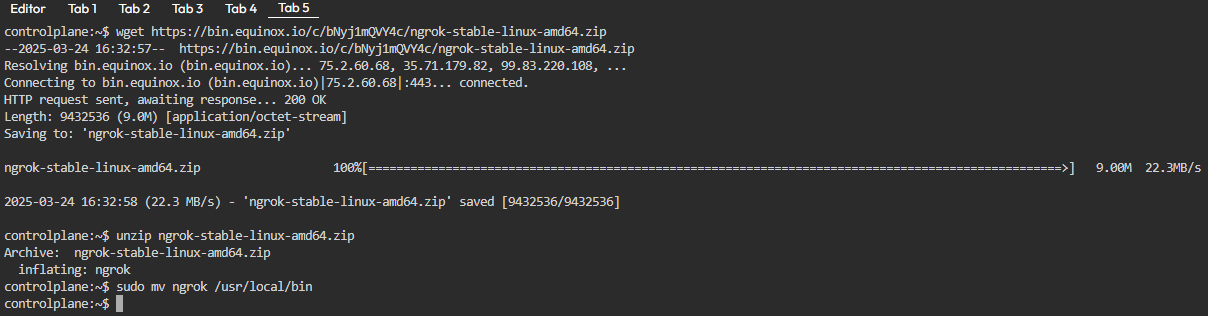
1. **Instalación y configuración de NGROK (Opcional)**

Descargaremos e instalaremos ngrok a fin de poder acceder externamente desde Postman al servicio desplegado en Killercoda

wget https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-stable-linux-amd64.zip

unzip ngrok-stable-linux-amd64.zip

sudo mv ngrok /usr/local/bin



Agregamos el token de autenticación

ngrok config add-authtoken 2u5Y731jQWjLgVE7GpTjCOGYSX1\_7GfHZzeHZYcrhFCR3i7ay



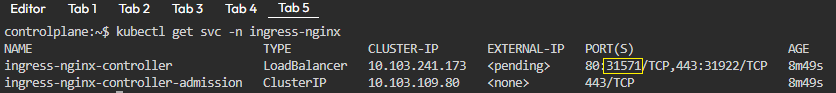
Verificamos la instalación

ngrok --version



Detectamos el puerto de ingress controller para exponer las solicitudes externas

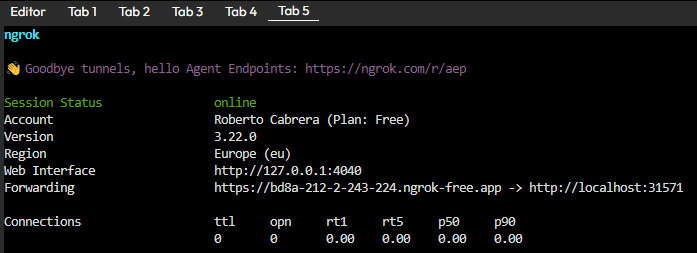
kubectl get svc -n ingress-nginx



Exponemos el ingress externamente con ngrok con el puerto identificado (reemplazar PORT por la información devuelta en el paso anterior).

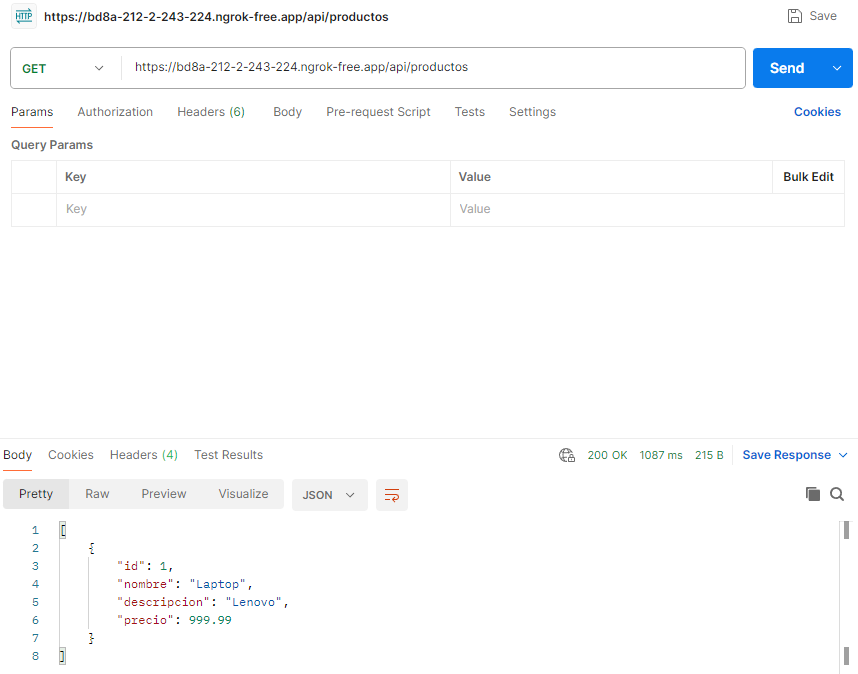
ngrok http <PORT> --host-header=ms-producto.server

Nos mostrará la URL que usaremos externamente para comunicarnos vía Postman

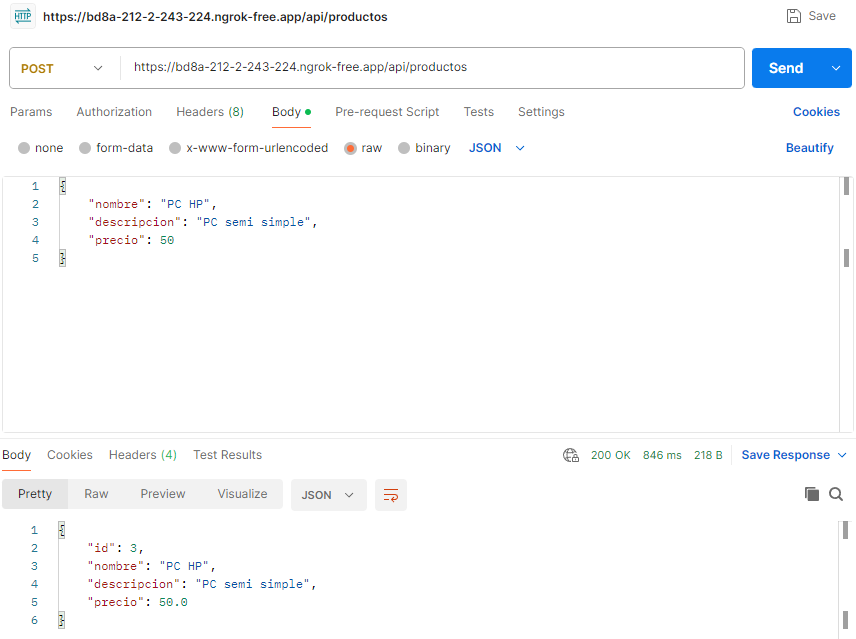


Copiamos la URL expuesta externamente y realizamos las pruebas correspondientes (esto está conectado con los datos registrados previamente, tanto en la prueba de service y de ingress)

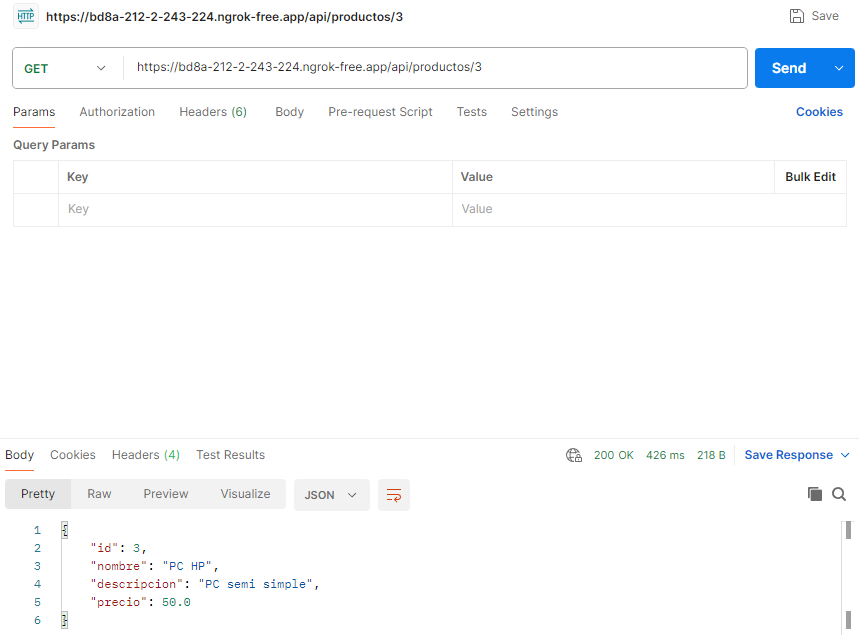
*Listar Productos*



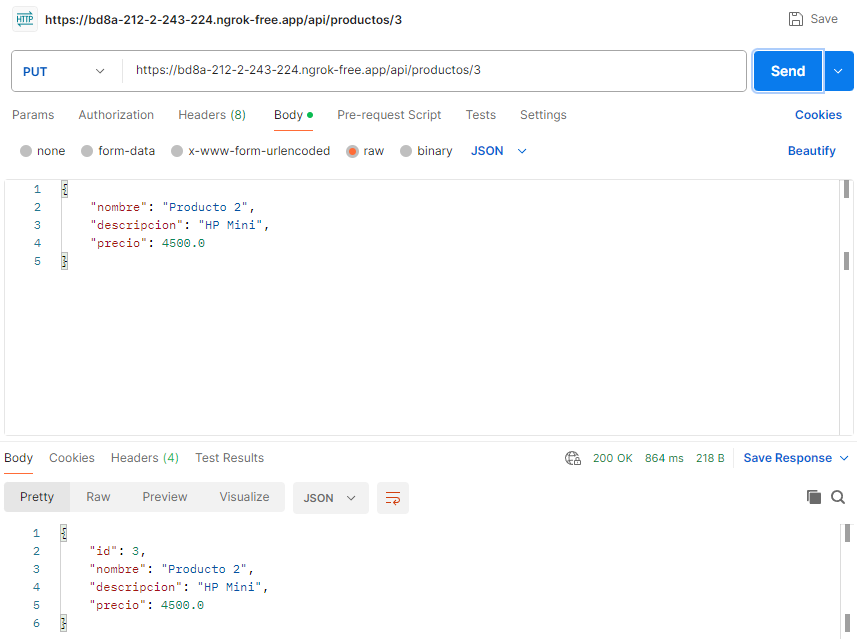
*Registrar Producto*



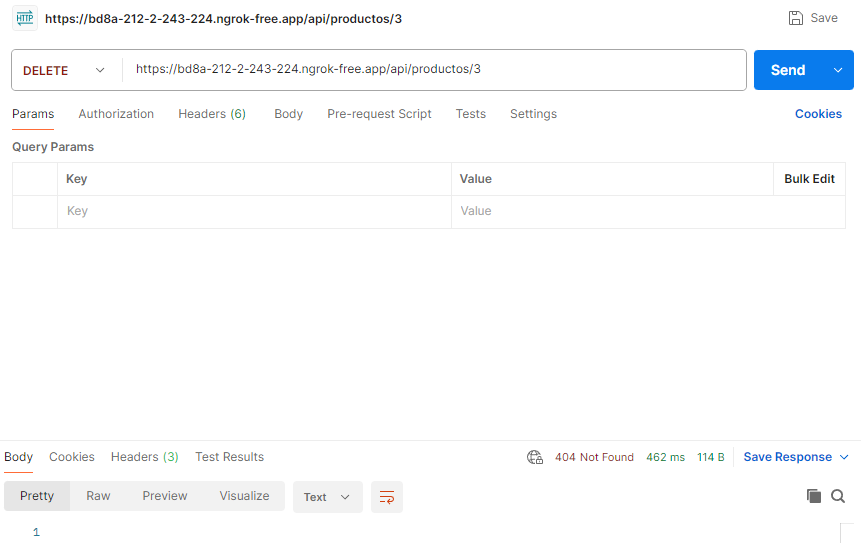
*Obtener Producto*



*Actualizar Producto*



*Eliminar Producto*



Revisamos las respuestas desde ngrok

