**Anexo 1: Evidencias y Pruebas del proyecto**

1. **Dockerfile y Registro de Imagen**

Se tiene el servicio desarrollado con spring-boot “pfinal-product-service”:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Se genera el ejecutable a través de gradle:

Texto

Descripción generada automáticamente

Se verificar el jar generado:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

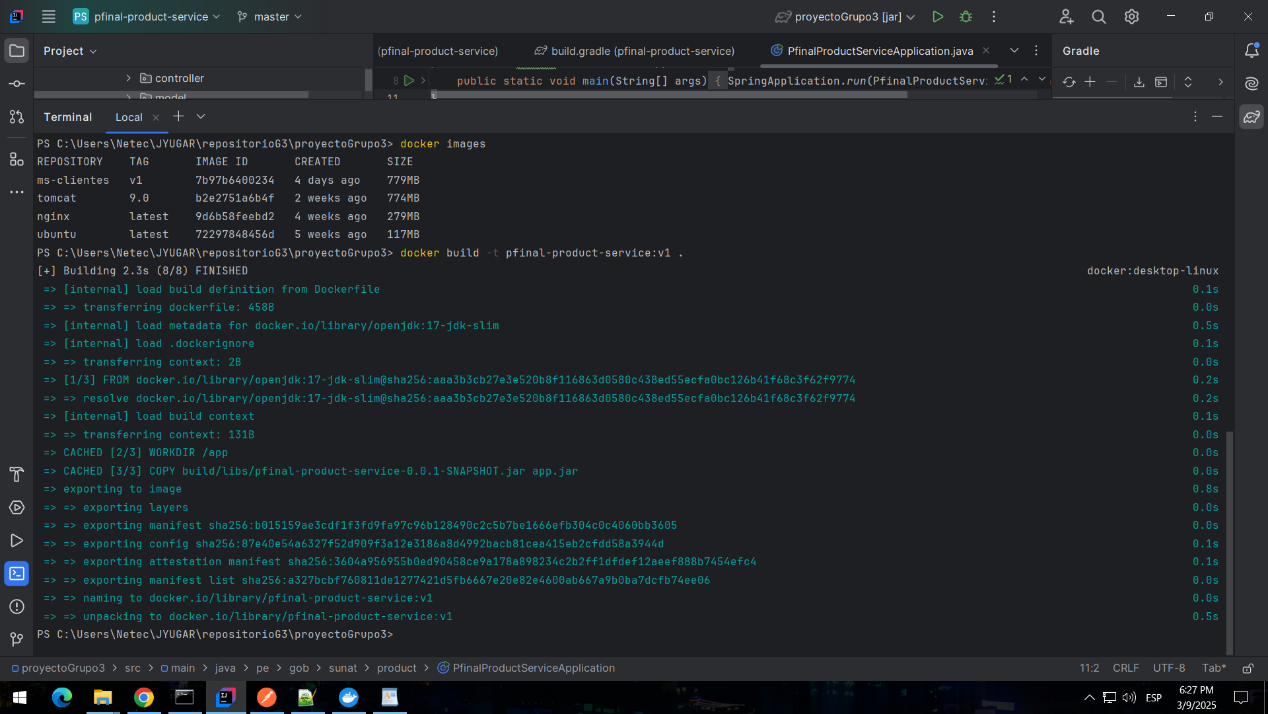
Descripción generada automáticamente

Se crea el archivo Dockerfile indicando la version de java, el programa jar generado y el puerto por donde se expondrá la aplicacion:

Captura de pantalla de computadora

Descripción generada automáticamente

Se ejecuta el comando docker para la creación de una imagen a partir del jar.



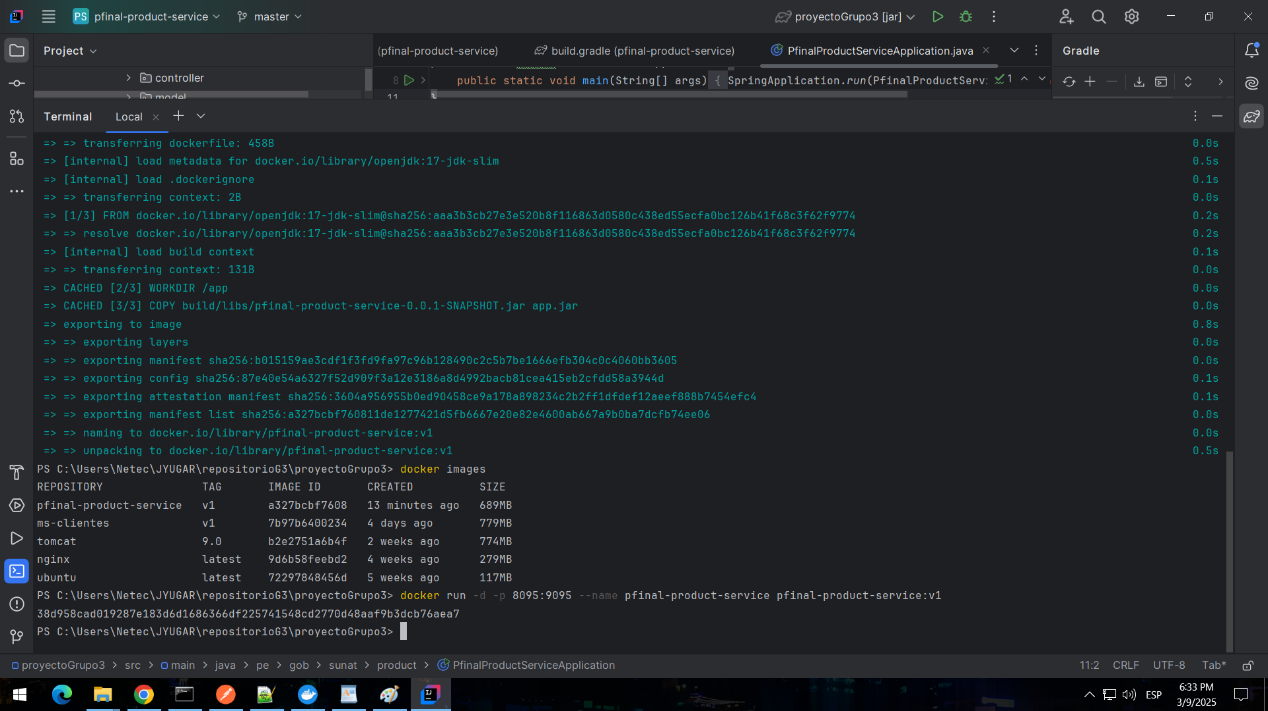
Se verifica la imagen creada:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Desplegar un contenedor basado en la imagen

docker run -d -p 8095:9095 --name pfinal-product-service pfinal-product-service:v1

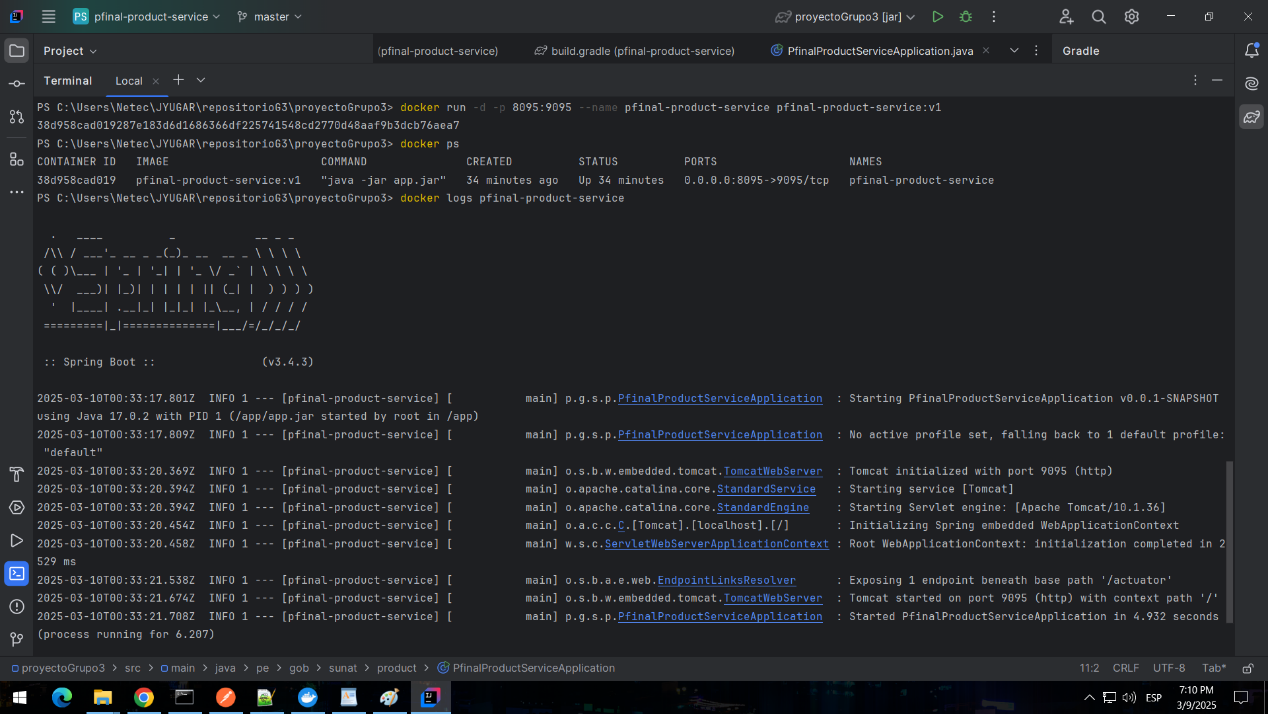


Verificar el estado del contenedor:

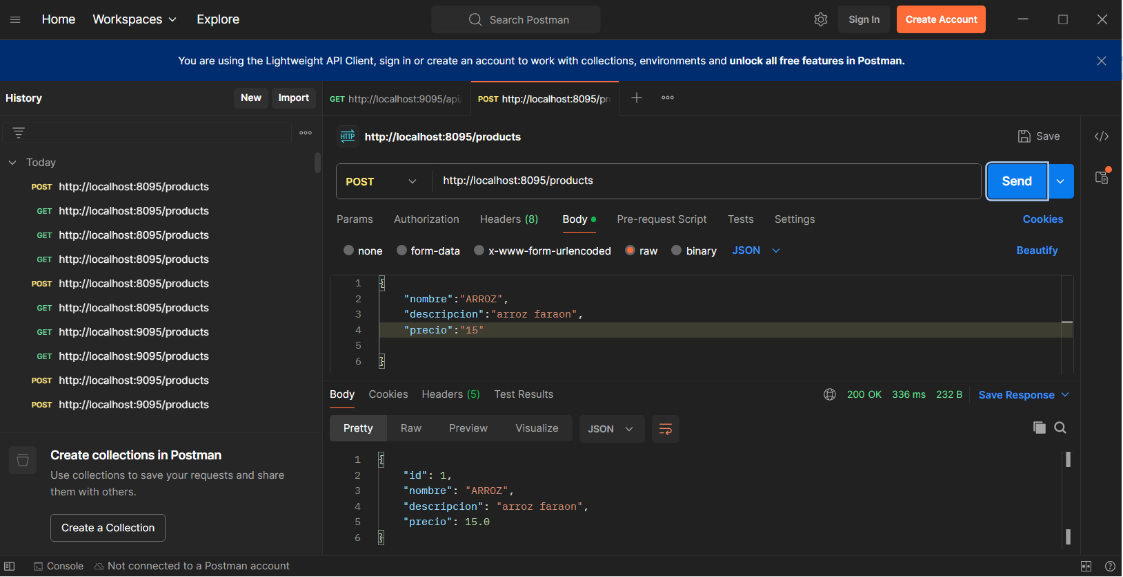
Pantalla de computadora

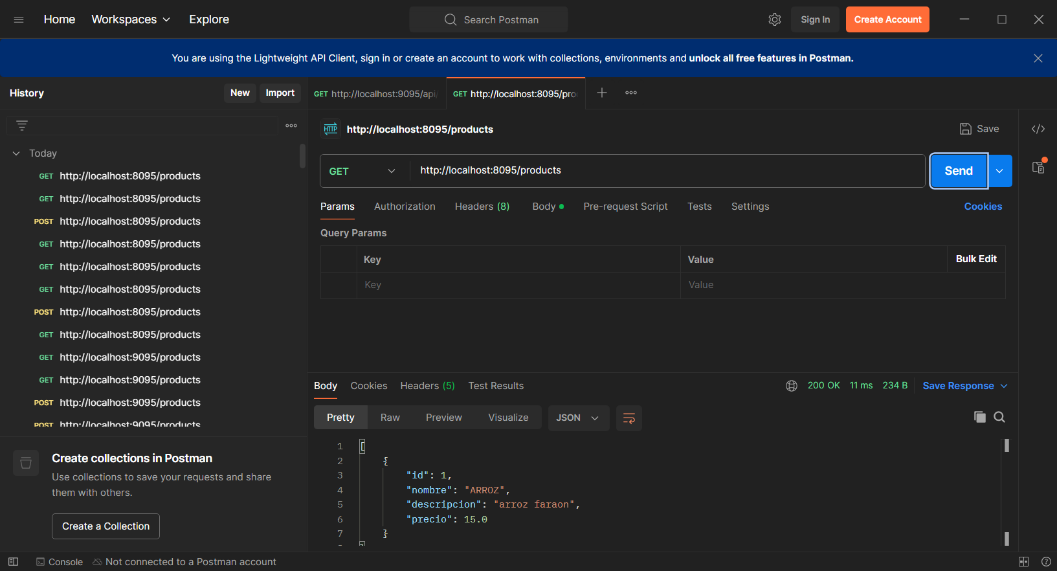
Descripción generada automáticamente

Verificar que el contenedor esté en ejecución a través de su log:



Probamos el servicio en postman pero a través del puerto del Contenedor:





**Subiendo imagen a docker hub**

Selecciono la imagen a subir:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Tagueo de imagen:

Texto

Descripción generada automáticamente

Push de imagen:

Texto

Descripción generada automáticamente

Se visualiza la imagen subida al dockerhub

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Sitio web

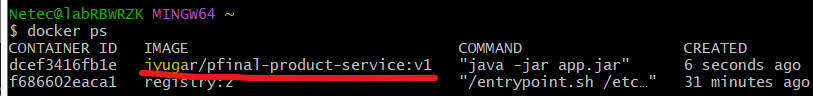
Descripción generada automáticamente

Se usa la imagen de Docker Hub para generar un 2do Contenedor:

Texto

Descripción generada automáticamente

Se verifica el contenedor generado:



1. **Despliegue en Kubernetes**

**Se hace uso de killercoda.com**

Creación de namespace: *kubectl create namespace pfinal*

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Edición de archivos Yaml

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Cargar los archivos al clúster:

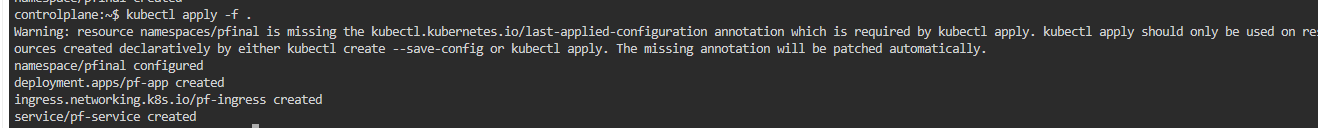
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Aplicamos archivos.



Instalamos ingress-nginx-controller

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

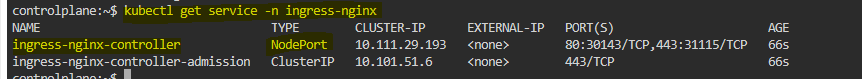
Texto

Descripción generada automáticamente

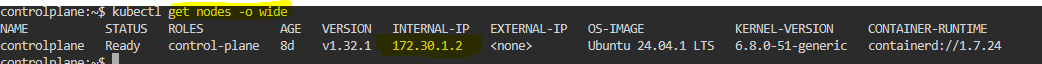
Editar cambiar: *kubectl edit service ingress-nginx-controller -n ingress-nginx*

Texto

Descripción generada automáticamente



Puerto 80 30143



IP 172.30.1.2

Matriculamos en el HOSTS

echo "172.30.1.2 empresa.com" | sudo tee -a /etc/hosts



curl <http://empresa.com:30143/products>

Texto

Descripción generada automáticamente

Después Configuramos un servidor Ngninx y cambiamos la configuración del Ingress con el dominio que otorga Killercoda.

Texto

Descripción generada automáticamente

Nginx

Texto

Descripción generada automáticamente

Host

Texto

Descripción generada automáticamente

**4. pruebas de aplicación**

*POST*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*GET*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*POST*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*GET*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente