Reto #5
Juliana Casas Ramirez
https://hub.docker.com/r/julianacasas28/spring-petclinic
https://github.com/juliana1212/Reto5_spring-petclinic.git

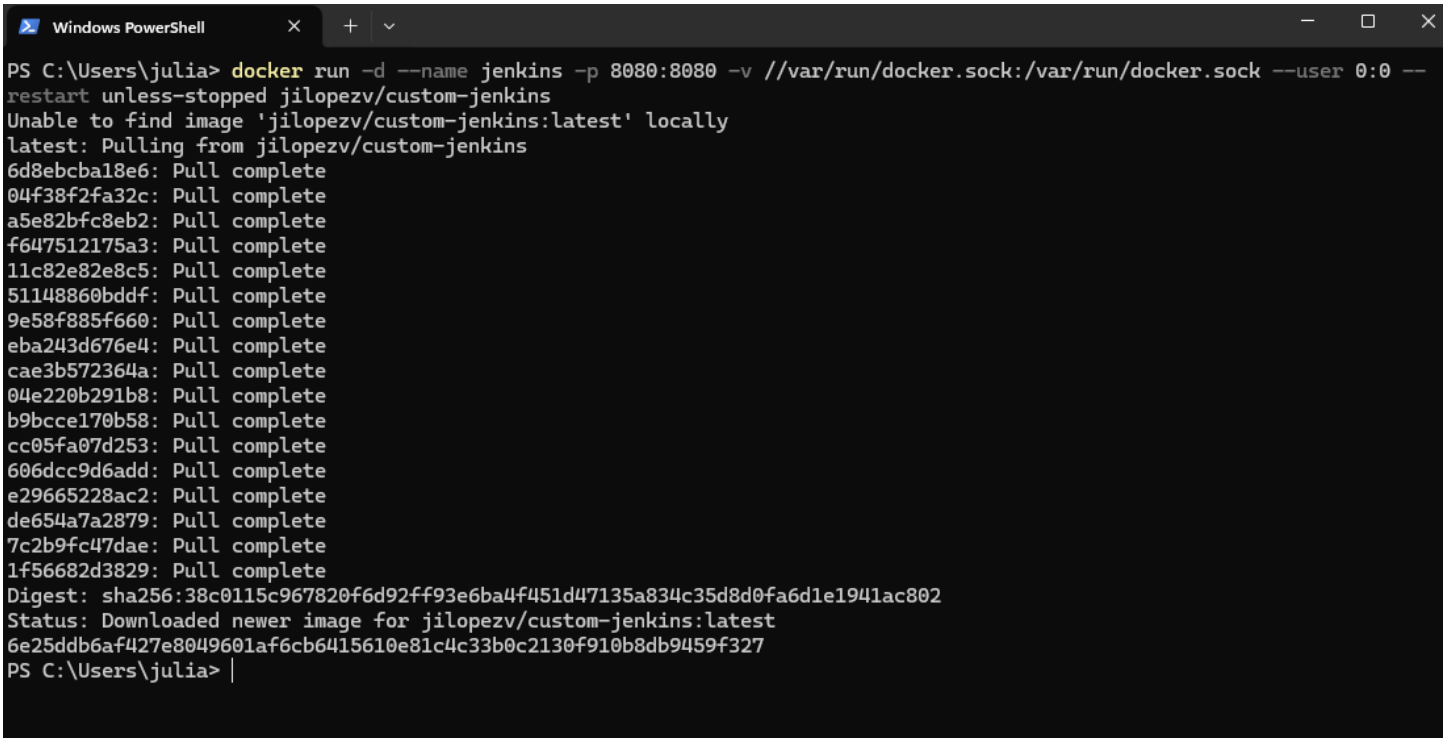
Reto #5

Jenkins + Docker + GitHub

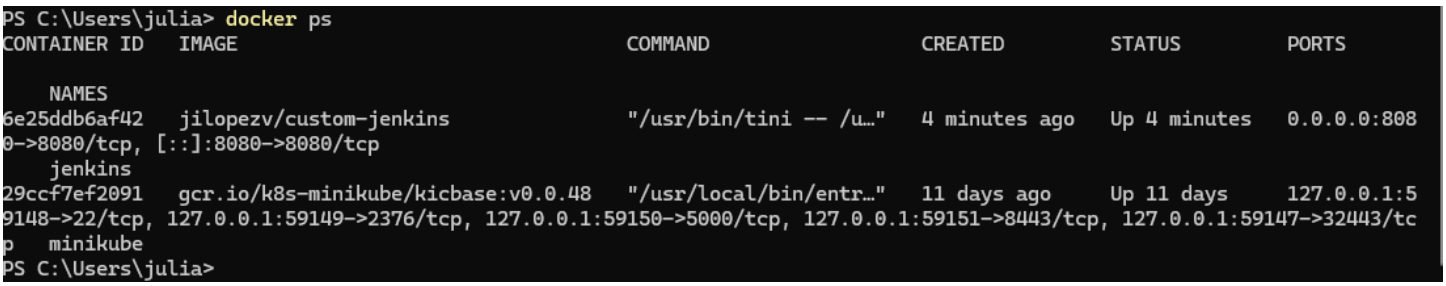
Evidencias: https://hub.docker.com/r/julianacasas28/spring-petclinic
https://github.com/juliana1212/Reto5_spring-petclinic.git

Paso 1: Funcionamiento de Jenkins localmente

En esta evidencia se muestra Jenkins ejecutándose de forma local en el puerto 8080, desplegado desde un contenedor Docker.
Se puede observar la interfaz principal del servidor con los proyectos configurados y el pipeline listo para ejecutarse.
Esto demuestra que Jenkins está correctamente instalado y accesible en el entorno local, funcionando de manera estable dentro del contenedor Docker.

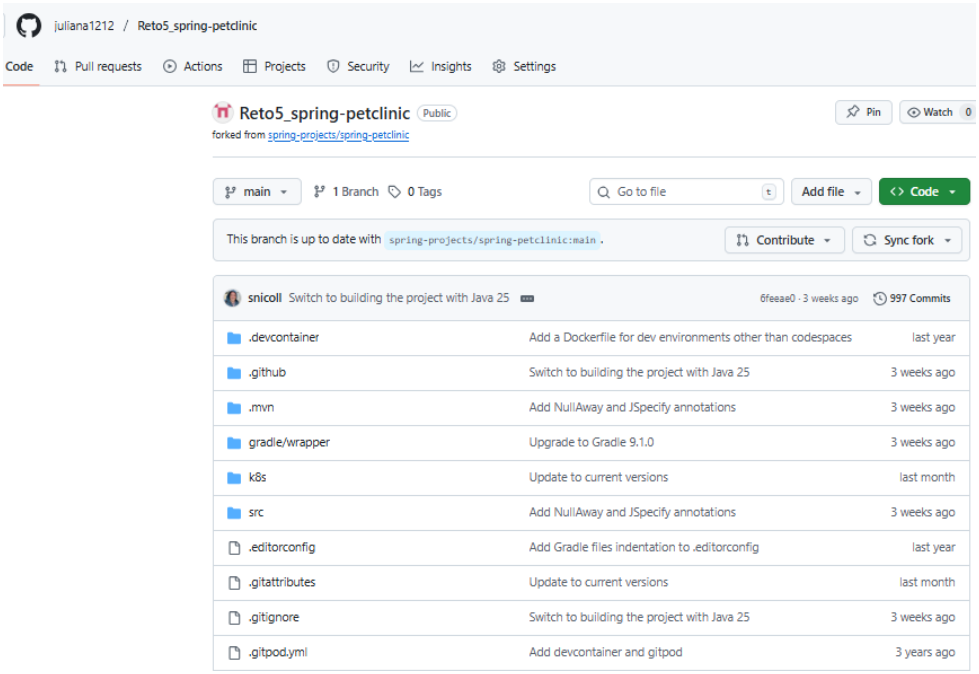


Comprobación de que el contenedor de Jenkins está corriendo correctamente en Docker, exponiendo el puerto 8080 para acceder al servicio localmente



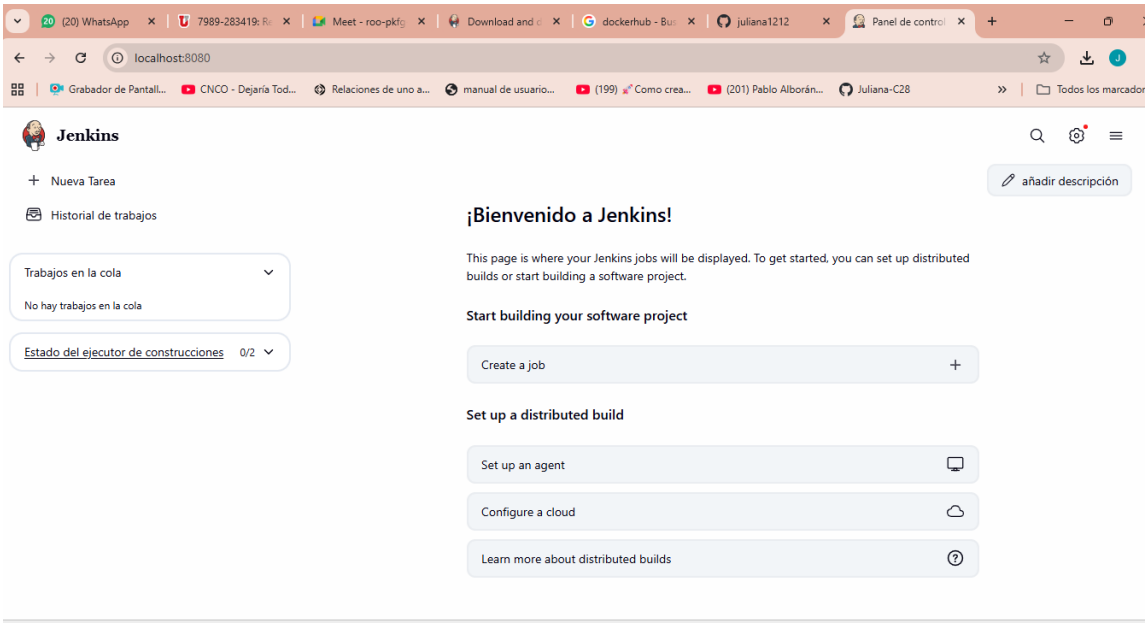
Paso 2:

Aquí se ve el repositorio forkeado en mi cuenta de GitHub y los cambios aplicados del tutorial (Jenkinsfile y Dockerfile actualizados). Este repositorio es la fuente que usa Jenkins para construir el proyecto.



Paso 3:

En esta imagen se muestra la página principal de Jenkins accediendo desde el navegador mediante la dirección http://localhost:8080.
Esto confirma que el contenedor de Jenkins se ejecutó correctamente y que el servicio está funcionando de manera local.



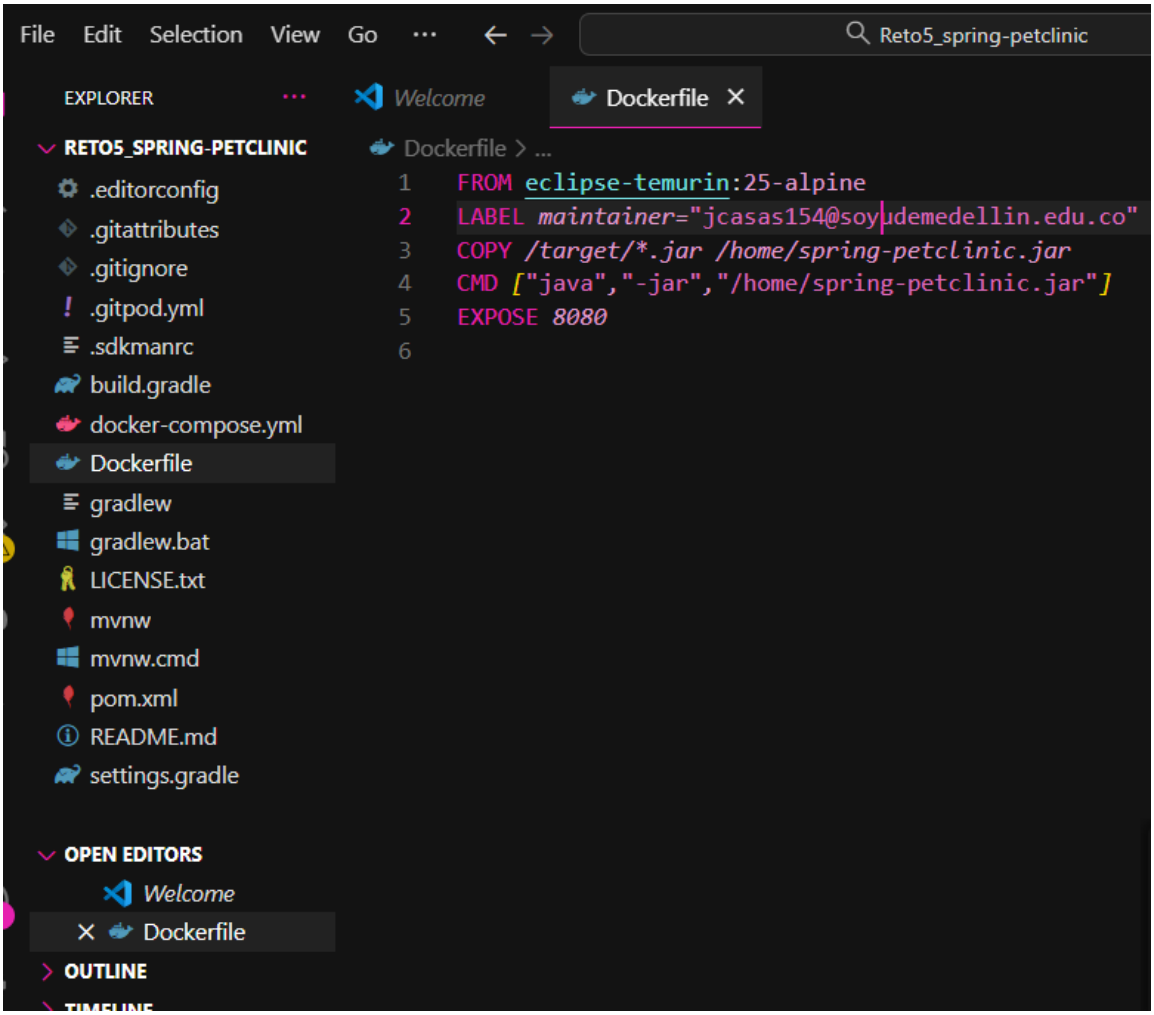
Paso 4:

En la siguiente imagen se observa el comando git clone ejecutado desde PowerShell, con el cual se clona el repositorio del proyecto **Spring PetClinic** desde GitHub.
Esto permitió obtener el código fuente completo para trabajar localmente y realizar las configuraciones necesarias.
El proceso finaliza exitosamente mostrando el mensaje *"done"*, lo que indica que todos los archivos fueron descargados correctamente

```
PS C:\Users\julia\Downloads> git clone https://github.com/juliana1212/Reto5_spring-petclinic.git
Cloning into 'Reto5_spring-petclinic'...
remote: Enumerating objects: 10931, done.
remote: Total 10931 (delta 0), reused 0 (delta 0), pack-reused 10931 (from 1)
Receiving objects: 100% (10931/10931), 7.92 MiB | 210.00 KiB/s, done.

Resolving deltas: 100% (4114/4114), done.
PS C:\Users\julia\Downloads>
```

Configuración del Dockerfile para construir la imagen base de la aplicación con Java 25



Paso 5:

En esta evidencia se muestra el uso de los comandos git add ., git commit y git push para registrar y enviar los cambios realizados en el proyecto hacia el repositorio remoto en GitHub.
El mensaje de confirmación "se creo archivo dockerfile" indica que se añadió el nuevo archivo Dockerfile al control de versiones, permitiendo que quede almacenado y visible dentro del repositorio del proyecto.

```
PS C:\Users\julia\Downloads\Reto5_spring-petclinic> git add .
PS C:\Users\julia\Downloads\Reto5_spring-petclinic> git commit -m "se creo archivo dockerfile"
[main b811ffb] se creo archivo dockerfile
 1 file changed, 5 insertions(+)
 create mode 100644 Dockerfile
PS C:\Users\julia\Downloads\Reto5_spring-petclinic> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 435 bytes | 145.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/juliana1212/Reto5_spring-petclinic.git
   6feeae0..b811ffb  main -> main
PS C:\Users\julia\Downloads\Reto5_spring-petclinic>
```

Luego, dentro del proyecto clonado, se creó el archivo Dockerfile, que define cómo se construirá la imagen Docker de la aplicación.

Este archivo indica que la imagen se basará en Java 25 (Temurin), copiará el archivo .jar generado por Maven y expondrá el puerto 8080 para acceder a la aplicación.

Jenkinsfile

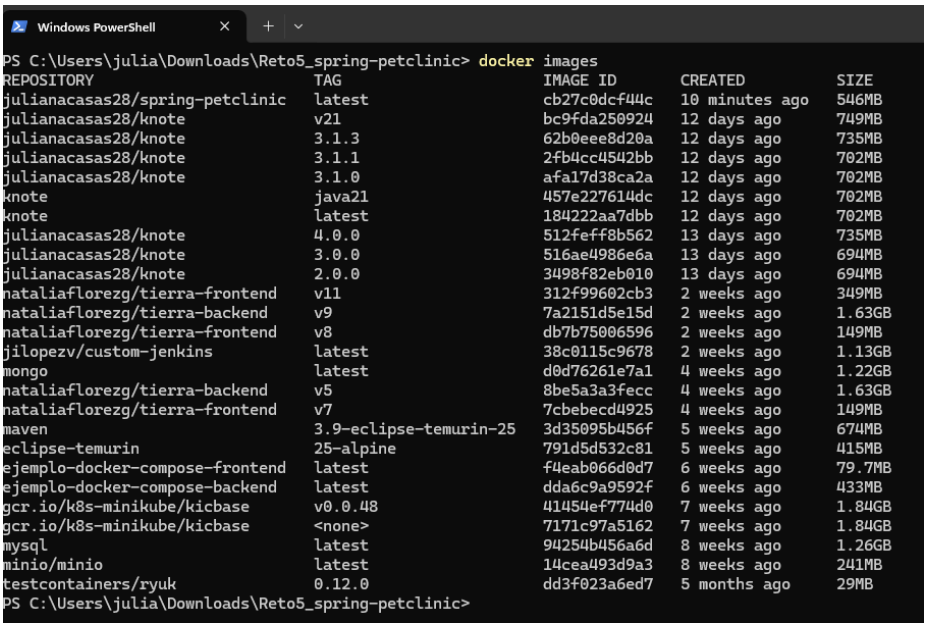
```
pipeline {
  stages {

    stage('Maven Install') {
      agent {
        docker {
          image 'maven:3.9-eclipse-temurin-25'
          reuseNode true
        }
      }
      steps {
        sh 'mvn clean install'
      }
    }

    stage('Docker Build') {
      agent any
      steps {
        sh 'docker build -t julianacasas28/spring-petclinic:latest .'
      }
    }

  }
}
```

Se realizó el commit y push del nuevo archivo Dockerfile al repositorio en GitHub, registrando los cambios realizados para mantener la trazabilidad del proyecto. Esto asegura que el archivo quede guardado en la nube y disponible para el pipeline en Jenkins.

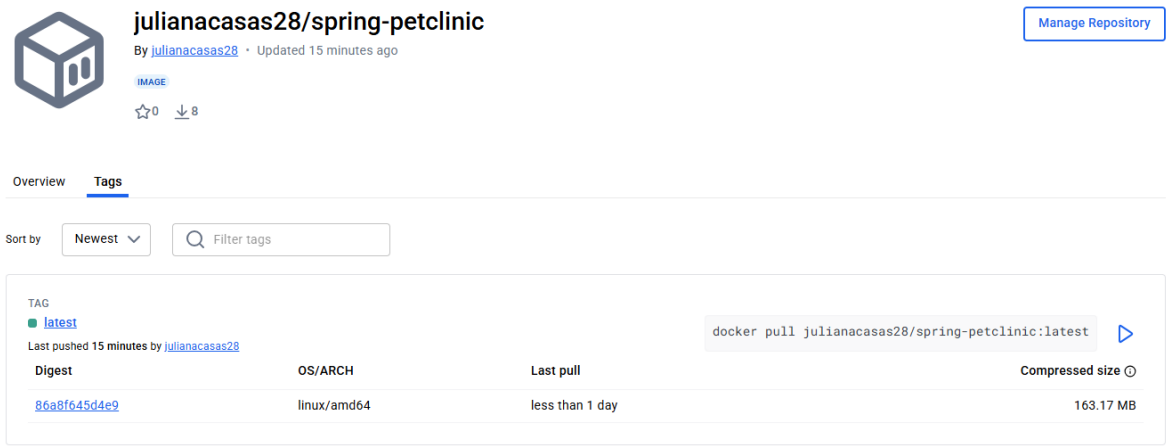


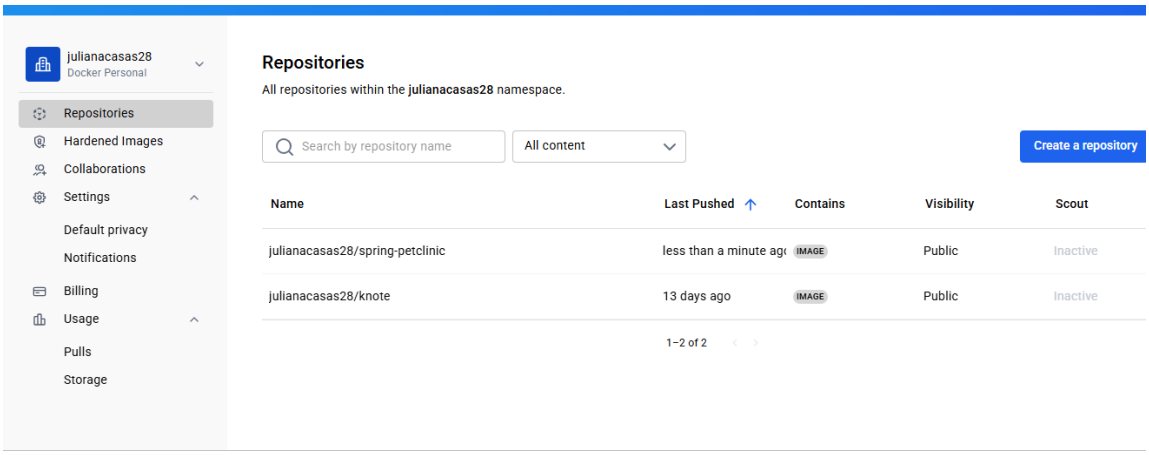
Paso 6:

En estas capturas se evidencia que el pipeline de Jenkins finalizó correctamente y logró subir la imagen de la aplicación Spring PetClinic al repositorio de Docker Hub.
Se puede observar que en el perfil del usuario julianacasas28 aparece el repositorio público llamado spring-petclinic, con la etiqueta latest (última versión de la imagen).

Esto confirma que el proceso de construcción y publicación funcionó correctamente:

1. Jenkins construyó el contenedor Docker desde el proyecto clonado y compilado.
2. Se realizó el *push* automático hacia Docker Hub usando las credenciales configuradas.
3. En Docker Hub se puede ver el registro “julianacasas28/spring-petclinic:latest”, con su digest y el tamaño comprimido de 163.17 MB.
4. El estado muestra “last pushed less than a minute ago”, lo que demuestra que fue cargado recientemente y de forma satisfactoria.





Paso 7: Aplicación corriendo localmente

En esta imagen se puede ver la aplicación Spring PetClinic funcionando correctamente en el navegador. Esto demuestra que el pipeline configurado en Jenkins logró construir la imagen Docker y que luego se pudo ejecutar de forma local con el comando docker run -p 8080:8080 julianacasas28/spring-petclinic:latest.

La interfaz muestra la página principal de la aplicación con el mensaje de bienvenida y la imagen de las mascotas, lo que confirma que el contenedor está funcionando sin errores.
Con este paso se valida que el reto fue completado exitosamente:

- Jenkins construyó la imagen de manera automática.
- La imagen fue subida correctamente a Docker Hub (usuario: *julianacasas28*).
- Finalmente, la aplicación se ejecutó localmente a través del contenedor Docker, comprobando el correcto funcionamiento del pipeline completo.