

## Reto #5

Juliana Casas Ramirez

<https://hub.docker.com/r/julianacasas28/spring-petclinic>

[https://github.com/juliana1212/Reto5\\_spring-petclinic.git](https://github.com/juliana1212/Reto5_spring-petclinic.git)

## Reto #5

### Jenkins + Docker + GitHub

Evidencias: <https://hub.docker.com/r/julianacasas28/spring-petclinic>

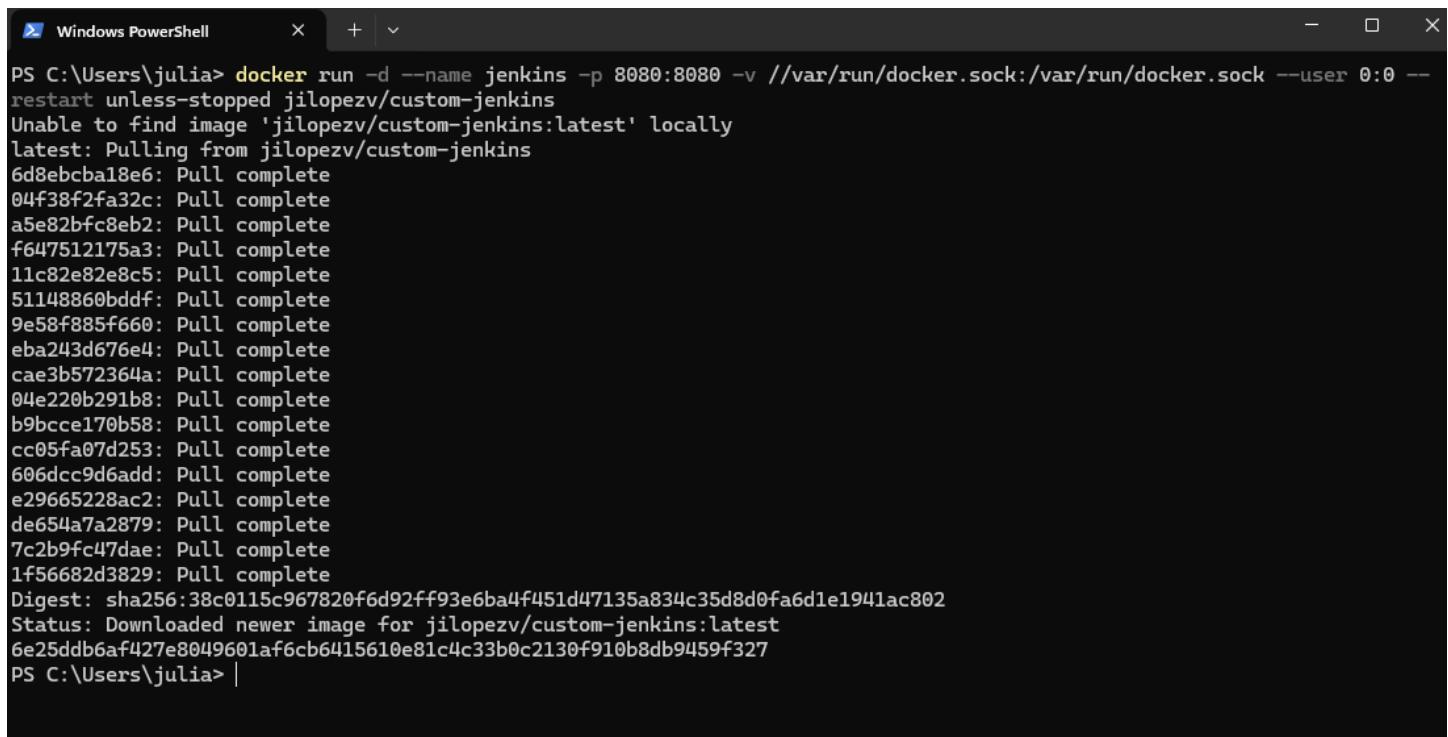
[https://github.com/juliana1212/Reto5\\_spring-petclinic.git](https://github.com/juliana1212/Reto5_spring-petclinic.git)

#### 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.

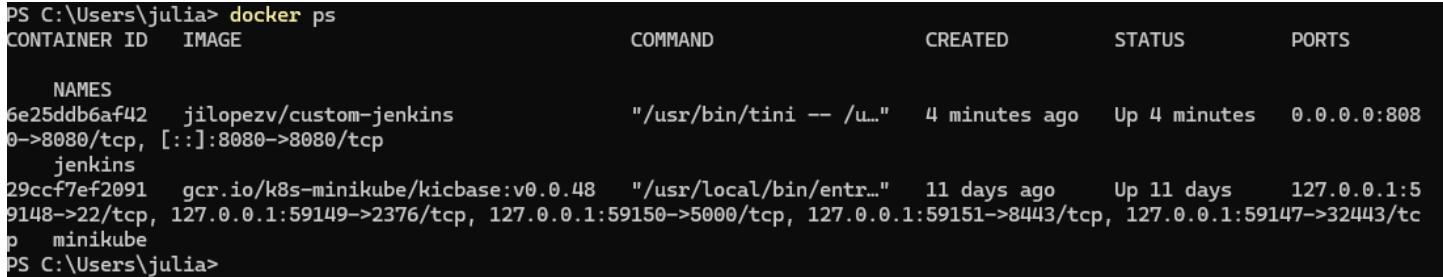
Esto demuestra que Jenkins está correctamente instalado y accesible en el entorno local, funcionando de manera estable dentro del contenedor Docker.



```
Windows PowerShell

PS C:\Users\julia> docker run -d --name jenkins -p 8080:8080 -v //var/run/docker.sock:/var/run/docker.sock --user 0:0 --restart unless-stopped jilopezv/custom-jenkins
Unable to find image 'jilopezv/custom-jenkins:latest' locally
latest: Pulling from jilopezv/custom-jenkins
6d8ebcba18e6: Pull complete
04f38f2fa32c: Pull complete
a5e82bfc8eb2: Pull complete
f647512175a3: Pull complete
11c82e82e8c5: Pull complete
51148860bddf: Pull complete
9e58f885f660: Pull complete
eba243d676e4: Pull complete
cae3b572364a: Pull complete
04e220b291b8: Pull complete
b9bcce170b58: Pull complete
cc05fa07d253: Pull complete
606dcc9d6add: Pull complete
e29665228ac2: Pull complete
de654a7a2879: Pull complete
7c2b9fc47dae: Pull complete
1f56682d3829: Pull complete
Digest: sha256:38c0115c967820f6d92ff93e6ba4f451d47135a834c35d8d0fa6d1e1941ac802
Status: Downloaded newer image for jilopezv/custom-jenkins:latest
6e25ddb6af427e8049601af6cb6415610e81c4c33b0c2130f910b8db9459f327
PS C:\Users\julia> |
```

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



COLUMN	COMMAND	CREATED	STATUS	PORTS
NAMES				
6e25ddb6af42_jilopezv/custom-jenkins	" /usr/bin/tini -- /u..."	4 minutes ago	Up 4 minutes	0.0.0.0:8080->8080/tcp
jenkins				
29cff7ef2091_gcr.io/k8s-minikube/kicbase:v0.0.48	" /usr/local/bin/entr..."	11 days ago	Up 11 days	127.0.0.1:5
9148->22/tcp, 127.0.0.1:59149->2376/tcp, 127.0.0.1:59150->5000/tcp, 127.0.0.1:59151->8443/tcp, 127.0.0.1:59147->32443/tcp				
p_minikube				
PS C:\Users\julia>				

## 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.

The screenshot shows a GitHub repository page for 'Reto5\_spring-petclinic'. The repository is public and was forked from 'spring-projects/spring-petclinic'. The main branch is 'main' with 1 branch and 0 tags. The repository has 997 commits from user 'snicoll', all of which are up-to-date with the original repository. The commits are listed in reverse chronological order, with dates ranging from 3 years ago to last year. The commits include changes to '.devcontainer', '.github', '.mvn', 'gradle/wrapper', 'k8s', 'src', '.editorconfig', '.gitattributes', '.gitignore', and '.gitpod.yml' files.

## 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.

The screenshot shows the Jenkins dashboard at <http://localhost:8080>. The dashboard features a '¡Bienvenido a Jenkins!' (Welcome to Jenkins!) message. It includes sections for 'Trabajos en la cola' (Jobs in the queue) and 'Estado del ejecutor de construcciones' (Build executor status). A prominent 'Create a job' button is available. Below it, there are links for 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'.

#### 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

The screenshot shows a code editor interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, ..., back, forward, search bar containing "Reto5\_spring-petclinic".
- Explorer:** Shows the project structure under "RETO5\_SPRING-PETCLINIC": .editorconfig, .gitattributes, .gitignore, .gitpod.yml, .sdkmanrc, build.gradle, docker-compose.yml, Dockerfile (highlighted), gradlew, gradlew.bat, LICENSE.txt, mvnw, mvnw.cmd, pom.xml, README.md, settings.gradle.
- Open Editors:** Shows the "Dockerfile" editor tab (highlighted).
- Dockerfile Content:**

```
1 FROM eclipse-temurin:25-alpine
2 LABEL maintainer="jcasa154@soyidemedellin.edu.co"
3 COPY /target/*.jar /home/spring-petclinic.jar
4 CMD ["java","-jar","/home/spring-petclinic.jar"]
5 EXPOSE 8080
6
```

## 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
1 pipeline {
2     stages {
3         stage('Maven Install') {
4             agent {
5                 docker {
6                     image 'maven:3.9-eclipse-temurin-25'
7                     reuseNode true
8                 }
9             }
10            steps {
11                sh 'mvn clean install'
12            }
13        }
14    }
15
16    stage('Docker Build') {
17        agent any
18        steps {
19            sh 'docker build -t julianacasas28/spring-petclinic:latest .'
20        }
21    }
22
23
24 }
```

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.

```
Windows PowerShell x + v
PS C:\Users\julia\Downloads\Reto5_spring-petclinic> docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
julianacasas28/spring-petclinic  latest   cb27c0dcf44c  10 minutes ago  546MB
julianacasas28/knote           v21      bc9fda250924  12 days ago   749MB
julianacasas28/knote           3.1.3    62b0eee8d20a  12 days ago   735MB
julianacasas28/knote           3.1.1    2fb4cc4542bb  12 days ago   702MB
julianacasas28/knote           3.1.0    afa17d38ca2a  12 days ago   702MB
knote                 java21   457e227614dc  12 days ago   702MB
knote                 latest    184222aa7dbb  12 days ago   702MB
julianacasas28/knote           4.0.0    512feff8b562  13 days ago   735MB
julianacasas28/knote           3.0.0    516ae4986e6a  13 days ago   694MB
julianacasas28/knote           2.0.0    3498f82eb010  13 days ago   694MB
nataliaflorezg/tierra-frontend v11      312f99602cb3  2 weeks ago   349MB
nataliaflorezg/tierra-backend  v9       7a2151d5e15d  2 weeks ago   1.63GB
nataliaflorezg/tierra-frontend v8       db7b75006596  2 weeks ago   149MB
jilopezv/custom-jenkins        latest   38c0115c9678  2 weeks ago   1.13GB
mongo                  latest   d0d76261e7a1  4 weeks ago   1.22GB
nataliaflorezg/tierra-backend v5       8be5a3a3fecc  4 weeks ago   1.63GB
nataliaflorezg/tierra-frontend v7       7cbebedc4925  4 weeks ago   149MB
maven                  3.9-eclipse-temurin-25 3d35995b456f  5 weeks ago   674MB
eclipse-temurin            25-alpine 791d5d532c81  5 weeks ago   415MB
ejemplo-docker-compose-frontend latest   f4eab066d0d7  6 weeks ago   79.7MB
ejemplo-docker-compose-backend latest   dda6c9a9592f  6 weeks ago   433MB
gcr.io/k8s-minikube/kicbase   v0.0.48  41454ef774d0  7 weeks ago   1.84GB
gcr.io/k8s-minikube/kicbase  <none>   7171c97a5162  7 weeks ago   1.84GB
mysql                  latest   94254b456a6d  8 weeks ago   1.26GB
minio/minio              latest   14cea493d9a3  8 weeks ago   241MB
testcontainers/ryuk           0.12.0   dd3f023a6ed7  5 months ago  29MB
PS C:\Users\julia\Downloads\Reto5_spring-petclinic>
```

## 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.

The screenshot shows the Docker Hub repository page for `julianacasas28/spring-petclinic`. The repository was created by `julianacasas28` and updated 15 minutes ago. The `latest` tag is selected. The page displays the following details:

Digest	OS/ARCH	Last pull	Compressed size
<a href="#">86a8f645d4e9</a>	linux/amd64	less than 1 day	163.17 MB

At the top right, there is a [Manage Repository](#) button. Below the table, there is a link to run the Docker image: `docker pull julianacasas28/spring-petclinic:latest`.

## 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.

