



I.E.S. GABRIEL GARCÍA MÁRQUEZ

TRABAJO FIN DE CICLO

**INTEGRACIÓN Y DESPLIEGUE CONTINUO CON
Buildah, Jenkins, K3s y Git**

CICLO FORMATIVO DE GRADO SUPERIOR DE ASIR

01 de junio de 2025

NOMBRE DEL ALUMNO: Daniel Pérez Bayán
TUTORA DEL TFC: Arancha Hormigo Nieves

ÍNDICE

ÍNDICE	2
1. JUSTIFICACIÓN DEL PROYECTO.	3
2. OBJETIVOS.	4
2.1 OBJETIVO GENERAL.	4
2.2 OBJETIVOS ESPECÍFICOS.	4
3. INTRODUCCIÓN.	5
4. METODOLOGÍA Y DESARROLLO DEL PROYECTO	9
4.1 ENTORNO DE VIRTUALIZACIÓN: VirtualBox.	9
4.2 PREPARACIÓN DEL ENTORNO DE DESARROLLO.	10
4.2.1 INSTALAR LAS HERRAMIENTAS NECESARIAS.	19
4.2.2 INSTALAR K3s (Kubernetes ligero).	19
4.2.3 CONEXIÓN CON EL REPOSITORIO REMOTO EN GitHub.	20
4.2.4 ENLAZAR EL PROYECTO LOCAL EN UBUNTU.	21
4.2.5 CREACIÓN DE IMAGEN CON Buildah Y DockerHub.	25
4.2.6 Kubectl y K3s.	26
4.2.7 APLICACIÓN DE INTEGRACIÓN CONTINUA Y DESPLIEGUE CONTINUO (CI/CD)	30
4.2.8 INSTALACIÓN Y CONFIGURACIÓN DE Jenkins EN K3s MEDIANTE Helm.	30
CREDECIALES A CONFIGURAR	35
4.2.9 Ngrok y webhook.	50
4.3 PREPARACIÓN DEL ENTORNO DE PRODUCCIÓN.	58
4.3.1 INSTALACIÓN DE HERRAMIENTAS NECESARIAS.	59
4.3.2 CONFIGURACIÓN DE CLAVE SSH.	60
4.3.3 ESTRUCTURA DEL PROYECTO	61
4.3.4 K3s Y ACCESO AL CLÚSTER	61
5. RESULTADOS Y DISCUSIÓN.	62
5.1 RESULTADO FINAL Y PRUEBAS REALIZADAS.	62
PRUEBA DEL PIPELINE DE JENKINS.	62
VERIFICACIÓN VISUAL DE WORDPRESS.	65
TABLA COMPARATIVA DEL PROCESO DE SINCRONIZACIÓN.	66
6. CONCLUSIONES.	68
7. COSTE DESGLOSADO DEL PROYECTO.	68
8. REFERENCIAS BIBLIOGRÁFICAS.	69
9. ANEXOS.	71

1. JUSTIFICACIÓN DEL PROYECTO.

La razón por la cual he escogido este proyecto es porque en el ámbito laboral se presenta una necesidad de automatizar procesos de despliegue y gestión de aplicaciones. Dado que en un mundo donde la agilidad, la eficiencia y la escalabilidad son fundamentales, implementar un entorno de integración y despliegue continuo (CI/CD) se convierte en una práctica primordial para equipos de desarrollo y administración de sistemas.

El motivo principal por el que me he decantado por desarrollar este trabajo es la oportunidad de explorar tecnologías actuales y altamente demandadas como Jenkins, K3s, Git y Buildah.

Además, este proyecto me permite aplicar de forma práctica los conocimientos adquiridos durante el ciclo formativo.

Otra razón por la cual he elegido este trabajo es la oportunidad de construir un entorno de producción realista. Esto me permitirá enfrentarme a retos técnicos reales, mejorar mis habilidades y fortalecer mi perfil profesional.

En resumen, este proyecto combina mi interés personal por el mundo automatización de despliegues y la necesidad profesional de adquirir experiencia en tecnologías de integración y despliegue continuo, con el objetivo de estar mejor preparado para el día de mañana.

2. OBJETIVOS.

2.1 OBJETIVO GENERAL.

El objetivo principal de este proyecto es implementar y automatizar un entorno de integración y despliegue continuo para una aplicación WordPress, desplegada en un clúster K3s, utilizando Jenkins, Buildah, Git y otras herramientas complementarias.

Con este trabajo se pretende optimizar los procesos de construcción, prueba y despliegue de aplicaciones, asegurando un trabajo ágil, eficiente y reproducible.

2.2 OBJETIVOS ESPECÍFICOS.

Para alcanzar el objetivo general, se plantean los siguientes objetivos específicos:

- Configurar un clúster Kubernetes ligero (K3s) en un entorno de producción.
- Desarrollar y gestionar contenedores personalizados para WordPress utilizando Buildah como herramienta de creación de imágenes.
- Implantar un sistema de integración y despliegue continuo mediante Jenkins, definiendo un pipeline.
- Gestionar el código fuente y los archivos de configuraciones en un repositorio GitHub, para facilitar el seguimiento de cambios.
- Automatizar el despliegue de actualizaciones de la aplicación WordPress en el clúster a partir de cambios en el repositorio.
- Aplicar herramientas como Helm y Kubectl para gestionar el clúster y los despliegues de forma más eficiente.

3. INTRODUCCIÓN.

En el desarrollo de este proyecto se utilizan una serie de tecnologías y herramientas.

Las principales que se van a emplear son:

- **K3s**: Es una versión ligera de Kubernetes creada para ser más fácil de instalar, más rápida y menos pesada que la versión completa de Kubernetes.



K3S

- **WordPress**: WordPress es un CMS de código abierto, utilizado principalmente para crear y gestionar sitios web. Es uno de los CMS más populares del mundo debido a su facilidad de uso, flexibilidad y gran comunidad de usuarios y desarrolladores.



- **Buildah:** Es una herramienta que permite crear, modificar y gestionar imágenes de contenedores sin requerir de un demonio como Docker. Gracias a esto, se consigue más flexibilidad y seguridad que no dependen de un servicio adicional activo.



- **Docker Hub:** Es una plataforma en la nube que sirve para almacenar, compartir y distribuir imágenes que se crean con Buildah.



- **Helm:** Es una herramienta de gestión de paquetes para Kubernetes. Sirve para automatizar la instalación, actualización y configuración de aplicaciones dentro de un clúster Kubernetes.



- **Kubectl:** Es la herramienta de línea de comandos que se utiliza para interactuar con clústeres de Kubernetes.



- **Jenkins:** Es una herramienta de automatización de código abierto que se utilizará para implementar procesos de integración continua y despliegue continuo. Permite realizar tareas repetitivas de forma automática.



Jenkins

- **Git y GitHub:** Son sistemas de control de versiones que permitirán gestionar el código fuente y automatizar los despliegues.



- **MySQL:** Sistema de gestión de bases de datos para soportar la parte de datos de la aplicación WordPress.



Estas tecnologías tienen una relación directa con distintas asignaturas del ciclo formativo de **Administración de Sistemas Informáticos en Red (ASIR)**:

- **Implantación de Aplicaciones Web:** Relacionada con el despliegue de WordPress y su configuración en entornos de producción.
- **Administración de Sistemas Operativos:** Involucra la configuración de servidores Linux (Ubuntu) como base para los despliegues.
- **Administración de Sistemas Gestores de Bases de Datos:** Conecta con la instalación, configuración y administración de MySQL.
- **Servicios de Red e Internet:** Aplicable en la configuración de servicios de red y exposición de aplicaciones a internet.
- **Seguridad y Alta Disponibilidad:** En relación con las buenas prácticas de seguridad en clústeres Kubernetes y la estabilidad de los servicios.
- **Empresa e Iniciativa Emprendedora:** Aporta la visión de cómo estas tecnologías se utilizan hoy en día en el ámbito empresarial y en proyectos de infraestructura moderna.
- **Formación en Centros de Trabajo (FCT):** La integración de todas estas tecnologías simula entornos reales de trabajo en empresas IT.

4. METODOLOGÍA Y DESARROLLO DEL PROYECTO

En esta sección se va a describir los pasos necesarios para la realización del proyecto, incluyendo configuración de las máquinas, comandos, herramientas externas y procesos.

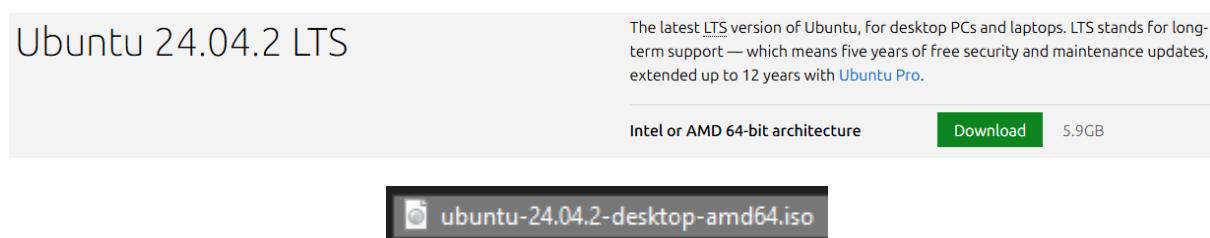
4.1 ENTORNO DE VIRTUALIZACIÓN: VirtualBox.

Para crear los entornos de desarrollo y producción, se va a utilizar Oracle VirtualBox (Versión de 2025), un software de virtualización de código abierto que permite ejecutar múltiples sistemas operativos como máquinas virtuales sobre un mismo equipo físico.

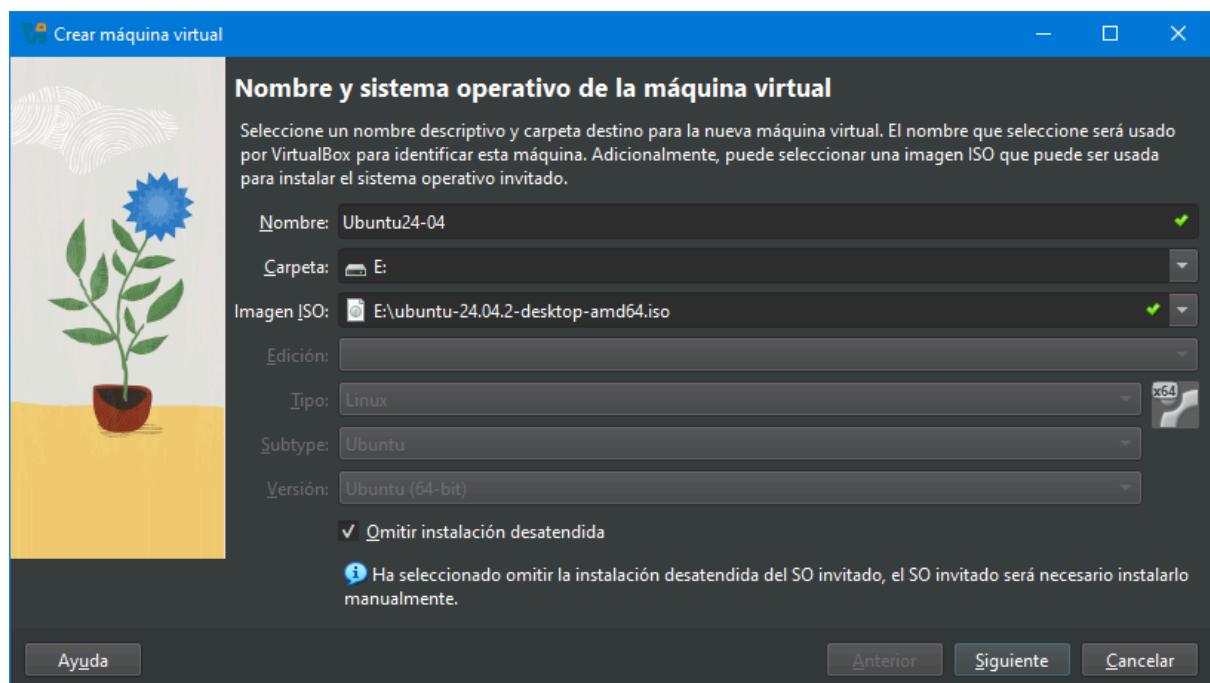


4.2 PREPARACIÓN DEL ENTORNO DE DESARROLLO.

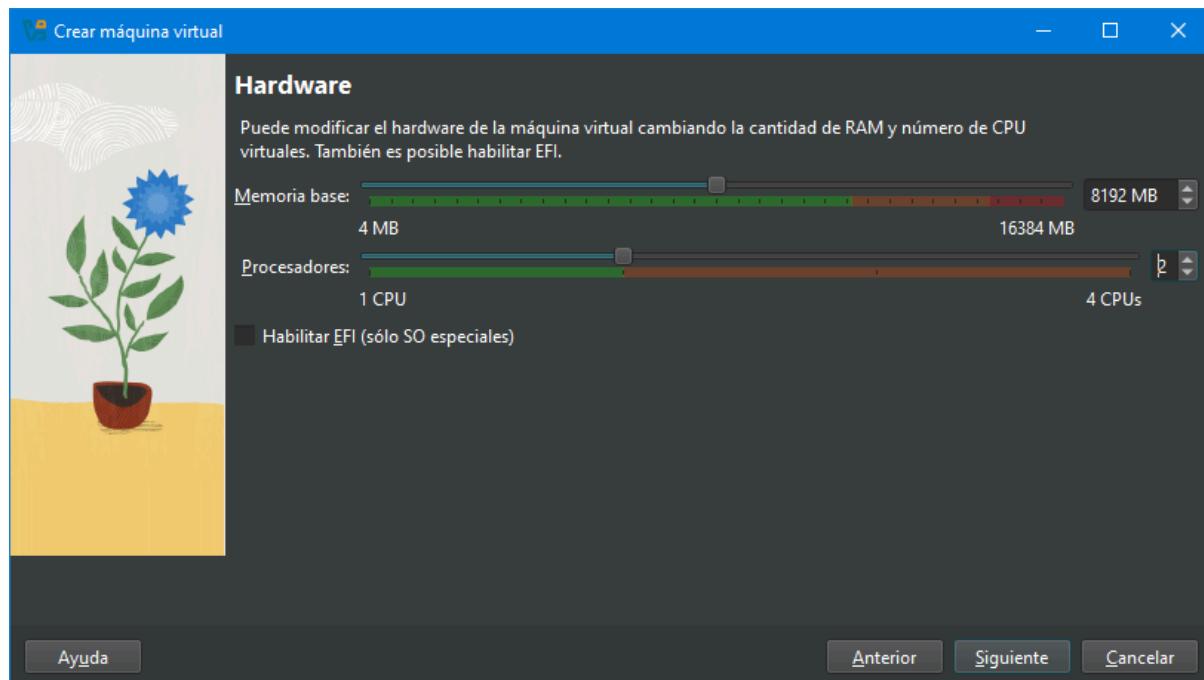
Para preparar el escenario del entorno de desarrollo se ha empleado una máquina virtual con el sistema operativo Ubuntu 24.04 LTS. Para empezar se tendrá que descargar la iso en la página oficial de Ubuntu, donde aparecerá la iso con la que se trabajará:



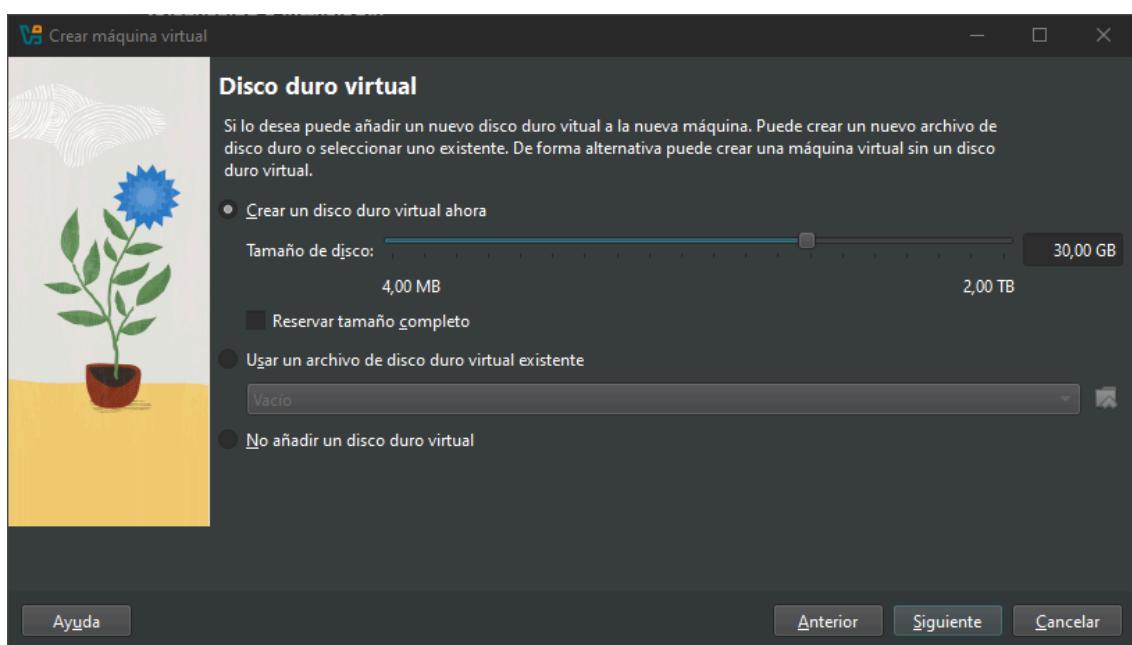
Una vez descargada la iso, se entra en VirtualBox y se crea la máquina virtual, en ella , se selecciona un nombre descriptivo de la máquina (Ubuntu24-04) y se añade la carpeta de destino de la nueva máquina virtual.



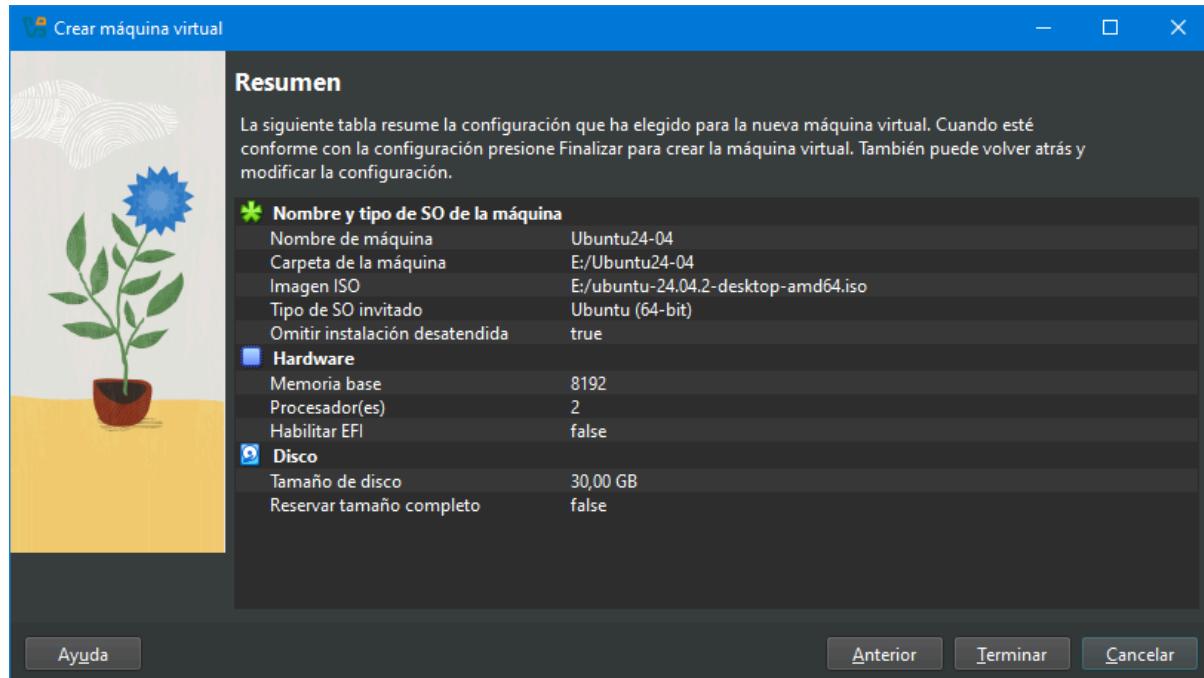
Después, pide modificar el hardware de la máquina virtual, en cuanto a la cantidad de memoria RAM dejaremos establecidos 8 GB de memoria base y en cuanto a el número de CPU virtuales se dejan 2 procesadores ya que se mejorará considerablemente el rendimiento.



Hay que crear un disco virtual, con un tamaño de disco de 30GB.

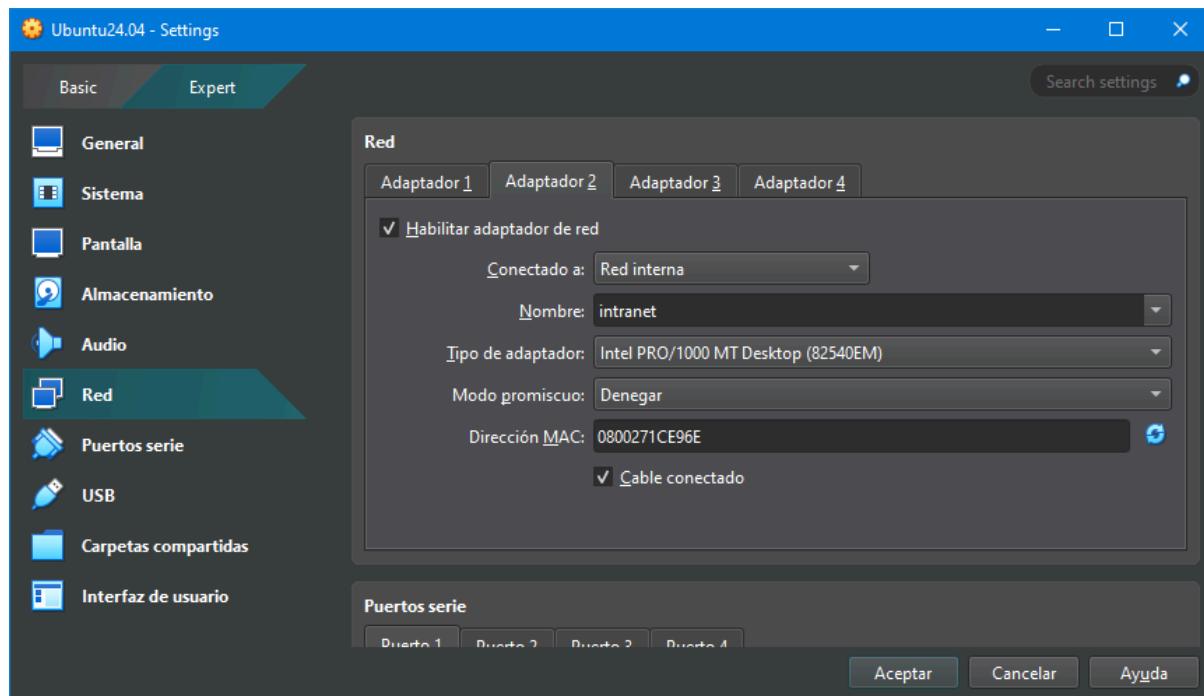
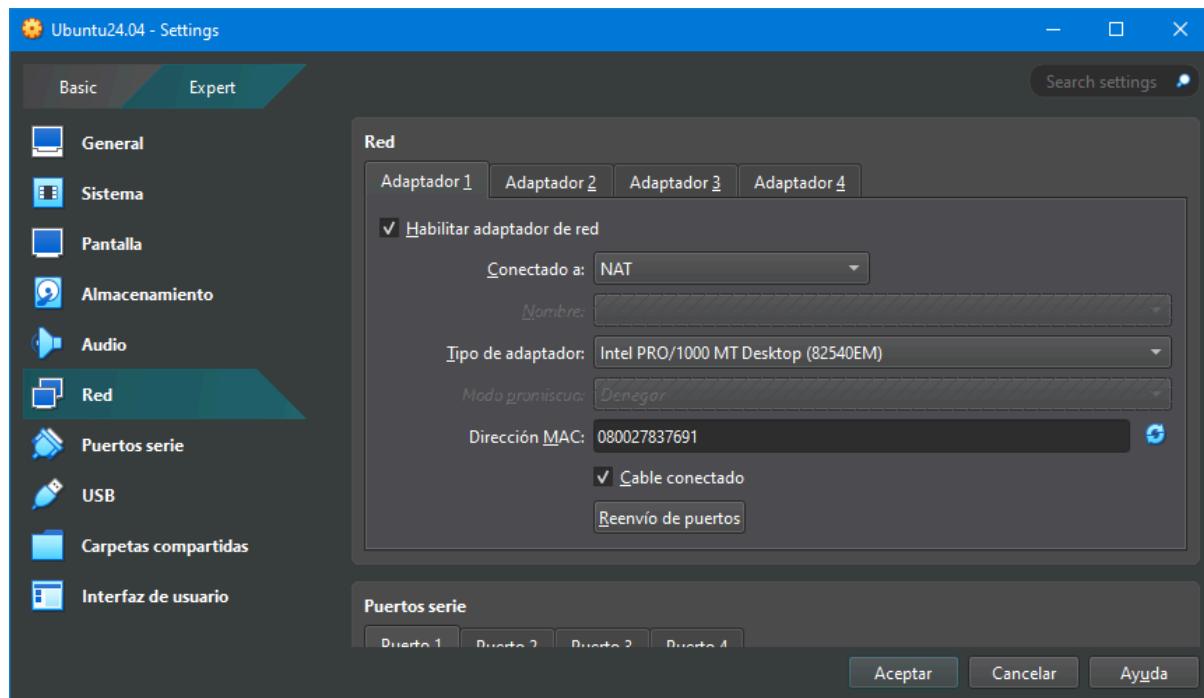


Por último, aparece un pequeño resumen con la información acerca de la máquina virtual.



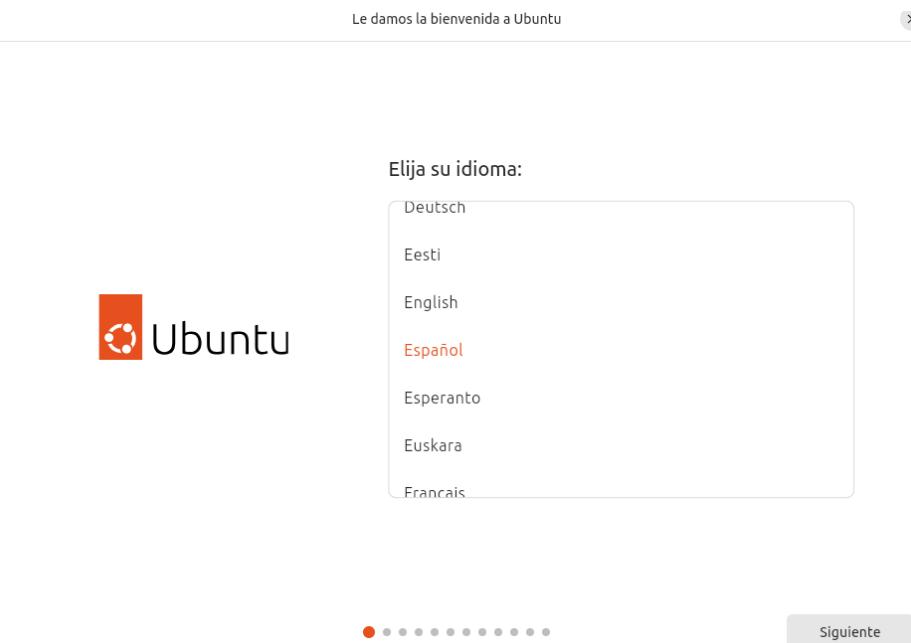
En cuanto a la configuración de red de la máquina tendremos 2 adaptadores:

Adaptador	Tipo	Uso
Adaptador 1	<u>NAT</u>	Acceso a Internet (apt, git)
Adaptador 2	<u>Red Interna (intranet)</u>	Comunicación con la VM de producción



Se arranca la máquina Ubuntu y ahora es momento de configurarla de una manera adecuada.

Se elige el idioma, al igual que la disposición del teclado.



Ahora se realizará la instalación de Ubuntu:



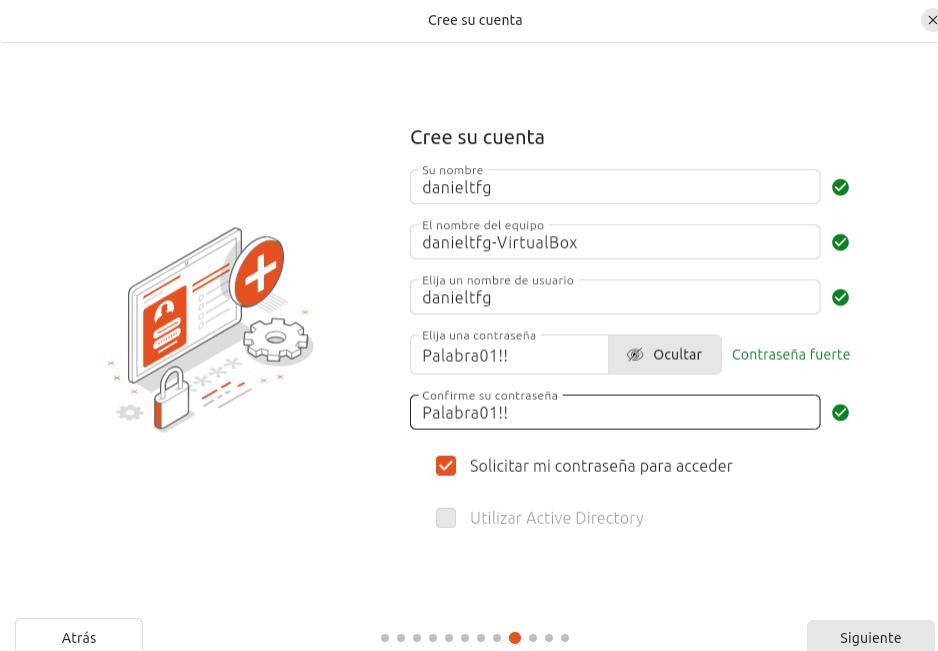
Se elige una selección predeterminada, el navegador web y las utilidades básicas.



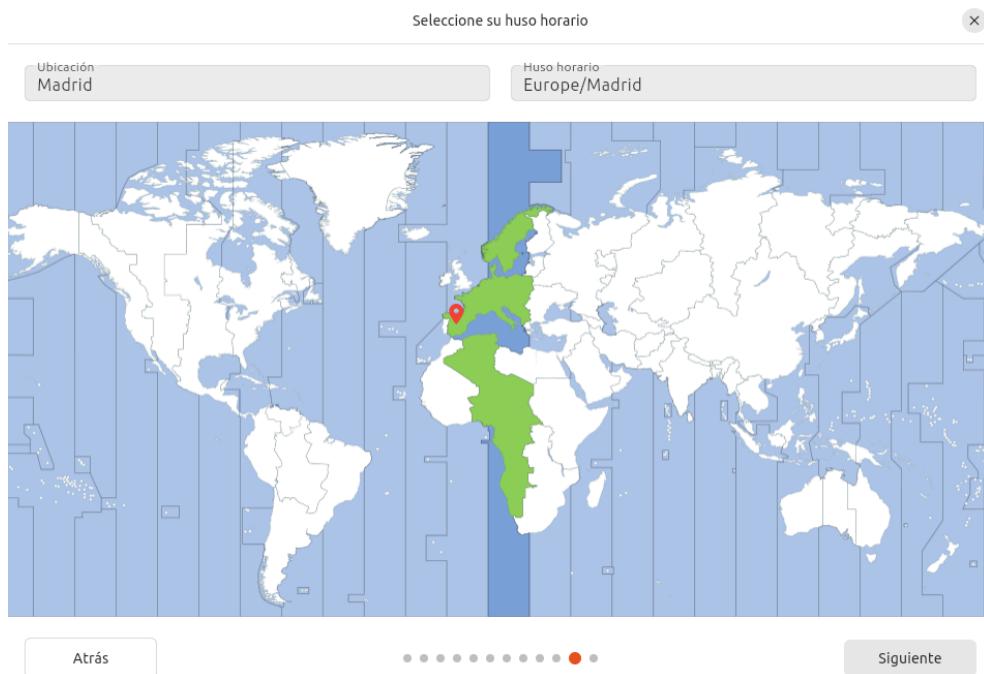
Se selecciona Borrar el disco e instalar Ubuntu, no hay que preocuparse ya que no borra nuestro disco sino el espacio de memoria reservado.



Se crea la cuenta de Ubuntu personalizada, con una contraseña segura.



Se selecciona la zona horaria adecuada.



De esta forma, ya estaría la instalación y la configuración realizada. Se reinicia la máquina para que empiece a funcionar de la manera adecuada.



Ubuntu 24.04.2 LTS está instalado y listo para usarse

Reinicie para completar la instalación o continuar las pruebas.
Los cambios que realice no se guardarán.

[Continuar probando](#)

[Reiniciar ahora](#)

Una vez reiniciado se instalará Guest Additions para la redimensión de pantalla automática.

Arriba en el menú de **VirtualBox → Dispositivos → Insertar imagen de CD de las Guest Additions.**

Las Guest Additions son un paquete especial de controladores y herramientas que se instalan dentro de la máquina virtual, gracias a esto Ubuntu funciona casi como un sistema real en el ordenador, no parece que se esté dentro de VirtualBox.

Por último se actualiza el Sistema Operativo.

Para empezar, hay que asegurarse de que el sistema está al día

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt update
```

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt upgrade -y
```

De esta forma se actualizan los repositorios y luego se instalan todas las actualizaciones disponibles.

Por último, se configuran las IPs de las máquinas manualmente, ya que sino, las VMs no tendrán dirección IP y no podrán comunicarse entre sí. En primer lugar, se hará en la máquina de desarrollo.

```
GNU nano 7.2          /etc/netplan/01-network-manager-all.yaml *
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    enp0s8:
      dhcp4: no
      addresses:
        - 192.168.100.1/24
      nameservers:
        addresses: [8.8.8.8]
```

4.2.1 INSTALAR LAS HERRAMIENTAS NECESARIAS.

Se procede a la descarga e instalación de paquetes y herramientas necesarias.

Instalación de herramientas, curl, git, buildah y mariadb-client

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt install curl git buildah mariadb-client -y
```

4.2.2 INSTALAR K3s (Kubernetes ligero).

Con este comando instalamos K3s:

```
danieltfg@danieltfg-VirtualBox:~$ curl -sfL https://get.k3s.io | sh -
```

Una vez terminado, se configura el KUBECONFIG:

```
danieltfg@danieltfg-VirtualBox:~$ export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
```

Con este comando kubectl funcionará correctamente.

/etc/rancher/k3s/k3s.yaml es el archivo que guarda la conexión con K3s, sirve para decirle a kubectl que archivo usar.

Ahora se mete en .bashrc para que se configure automáticamente cada vez que se abra una terminal:

```
danieltfg@danieltfg-VirtualBox:~$ echo 'export KUBECONFIG=/etc/rancher/k3s/k3s.yaml' >> ~/.bashrc
```

```
danieltfg@danieltfg-VirtualBox:~$ source ~/.bashrc
```

Ahora se crea la carpeta donde se trabajará.

```
danieltfg@danieltfg-VirtualBox:~$ mkdir ~/TFG-WordPress-K3s
```

```
danieltfg@danieltfg-VirtualBox:~$ cd ~/TFG-WordPress-K3s
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$
```

Dentro de la carpeta, se sincronizará con GitHub para el control del código fuente de la aplicación y dónde se ubicarán los ficheros de configuración del clúster, el fichero Dockerfile y el CMS.

4.2.3 CONEXIÓN CON EL REPOSITORIO REMOTO EN GitHub.

1. Se accede a GitHub a través de la URL: <https://github.com>, utilizando la cuenta de usuario dperezbayan.
2. Se crea un "Nuevo repositorio".
3. Configuración de los parámetros:
 - a. Nombre del repositorio: TFG-WordPress-K3s.
 - b. Descripción: Repositorio para el TFG de despliegue CI/CD con WordPress, K3s y Jenkins.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk ().*

Owner *	Repository name *
 dperezbayan	/ <input type="text" value="TFG-WordPress-K3s"/> <input checked="" type="checkbox"/> TFG-WordPress-K3s is available.
Description (optional)	
Repositorio para el TFG de despliegue CI/CD con WordPress, K3s y Jenkins	

<https://github.com/dperezbayan/TFG-WordPress-K3s>

4.2.4 ENLAZAR EL PROYECTO LOCAL EN UBUNTU.

Una vez creado el repositorio en GitHub, el siguiente paso es enlazar el repositorio local, donde se encontraban los archivos del proyecto, con el repositorio remoto en GitHub. Esto se realiza mediante los siguientes pasos en la terminal:

1. Se accede a la carpeta del proyecto.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$
```

2. Se inicializa un repositorio Git en el directorio local.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git init
```

3. Se vincula el repositorio local con el repositorio recién creado en GitHub.

```
~/TFG-WordPress-K3s$ git remote add origin https://github.com/dperezbayan/TFG-WordPress-K3s.git
```

4. Para establecer la rama principal como main:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git branch -M main
```

5. Se añaden todos los archivos del proyecto al repositorio local:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git add .
```

6. Se hace el primer commit para registrar los archivos en el repositorio local:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git commit -m "Primer commit del TFG"  
En la rama main  
Confirmación inicial
```

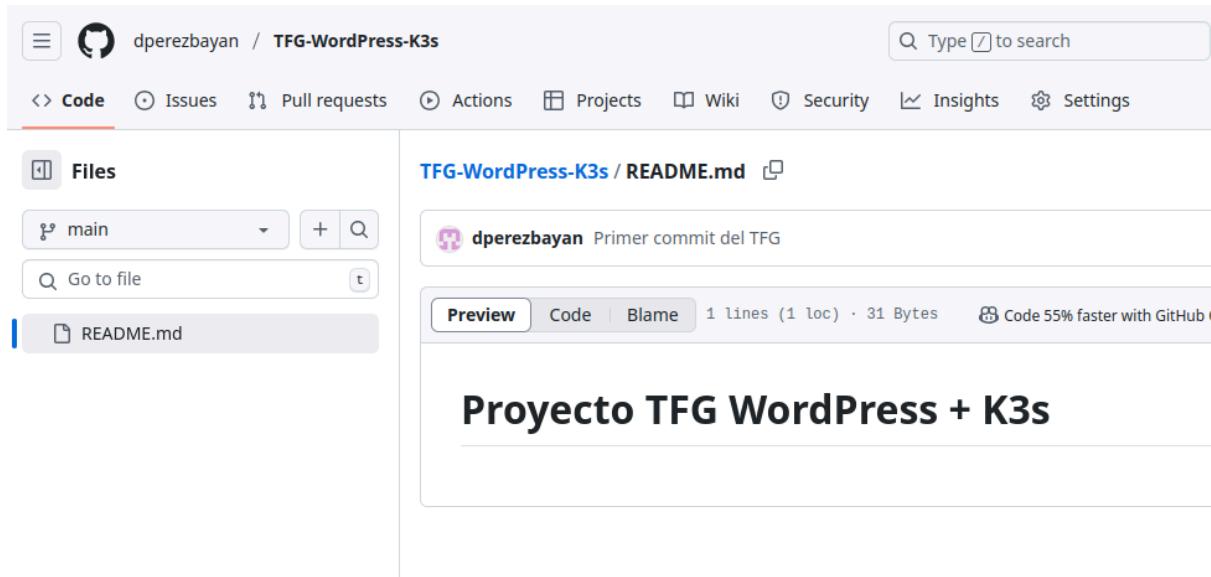
7. Por último, se suben los archivos al repositorio remoto en GitHub:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git push -u origin main
Username for 'https://github.com': dperezbayan
Password for 'https://dperezbayan@github.com':
Enumerando objetos: 3, listo.
Contando objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 265 bytes | 265.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/dperezbayan/TFG-WordPress-K3s.git
 * [new branch]      main -> main
rama 'main' configurada para rastrear 'origin/main'.
```

En cuanto a la password que hay que poner, se genera un token desde GitHub (el cual se guarda en .txt) y se introduce, de esta manera no habrá ningún problema para subir los archivos.



Para verificar que los archivos se han subido correctamente, se accede a la página del repositorio en GitHub: <https://github.com/dperezbayan>. Al ingresar en el repositorio TFG-WordPress-K3s, se puede comprobar que todos los archivos del proyecto que estaban en la máquina de desarrollo ya están presentes en el repositorio remoto de GitHub.



Ahora, se crea el fichero Dockerfile, es un archivo que contiene una serie de instrucciones que Docker utiliza para construir una imagen personalizada. Esta imagen incluye todos los componentes necesarios para ejecutar WordPress en un entorno preparado para su despliegue en K3s.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ touch Dockerfile  
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ sudo nano Dockerfile
```

```
1  # Selección de imagen base
2  FROM debian:12
3
4  # Variables de entorno para conectar con la base de datos
5  ENV WORDPRESS_DB_USER wordpress
6  ENV WORDPRESS_DB_PASSWORD wordpress
7  ENV WORDPRESS_DB_NAME wordpress
8  ENV WORDPRESS_DB_HOST mysql-service
9
10 # Instalación de dependencias necesarias para WordPress
11 RUN apt-get update && apt-get install -y \
12     apache2 \
13     mariadb-client \
14     php \
15     php-mysql \
16     curl \
17     && rm -rf /var/lib/apt/lists/*
18
19 # Descargar y mover WordPress al directorio web
20 RUN curl -o wordpress.tar.gz https://wordpress.org/latest.tar.gz && \
21     tar -xzf wordpress.tar.gz && \
22     mv wordpress/* /var/www/html && \
23     rm -rf wordpress wordpress.tar.gz /var/www/html/index.html
24
25 # Permisos de los archivos para que los use Apache
26 RUN chown -R www-data:www-data /var/www/html
27
28 # Exponer los puertos que usará el contenedor
29 EXPOSE 80 443
30
31 # Comando para arrancar el servidor Apache en primer plano
32 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Se añade el archivo al repositorio local.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git add Dockerfile
```

Se hace un commit con un mensaje descriptivo.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git commit -m "Añadido Dockerfile con la configuración de WordPress"
```

Se suben los cambios al repositorio en GitHub.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ git push origin main
Username for 'https://github.com': dperezbayan
Password for 'https://dperezbayan@github.com':
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 846 bytes | 846.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/dperezbayan/TFG-WordPress-K3s.git
  af797e9..5b28edb main -> main
```

4.2.5 CREACIÓN DE IMAGEN CON Buildah Y DockerHub.

Después de configurar el fichero Dockerfile, con la herramienta Buildah se crea la imagen.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ buildah bud -t wpimagen .
```

Al finalizar, se verifica que la imagen se ha generado correctamente:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ buildah images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
localhost/wpimagen latest   5831e36f0293  5 minutes ago  436 MB
docker.io/library/debian  12      b2ab84c007fe  3 weeks ago   121 MB
```

Para que esta imagen se pueda usar en un clúster K3s, es necesario publicarla en DockerHub, una plataforma que permite almacenar imágenes de contenedor. Antes de subir la imagen, es necesario autenticar la sesión con la cuenta personal de DockerHub. Para ello, primero hay que crearse una cuenta en DockerHub:



Welcome back, dperezbayan!

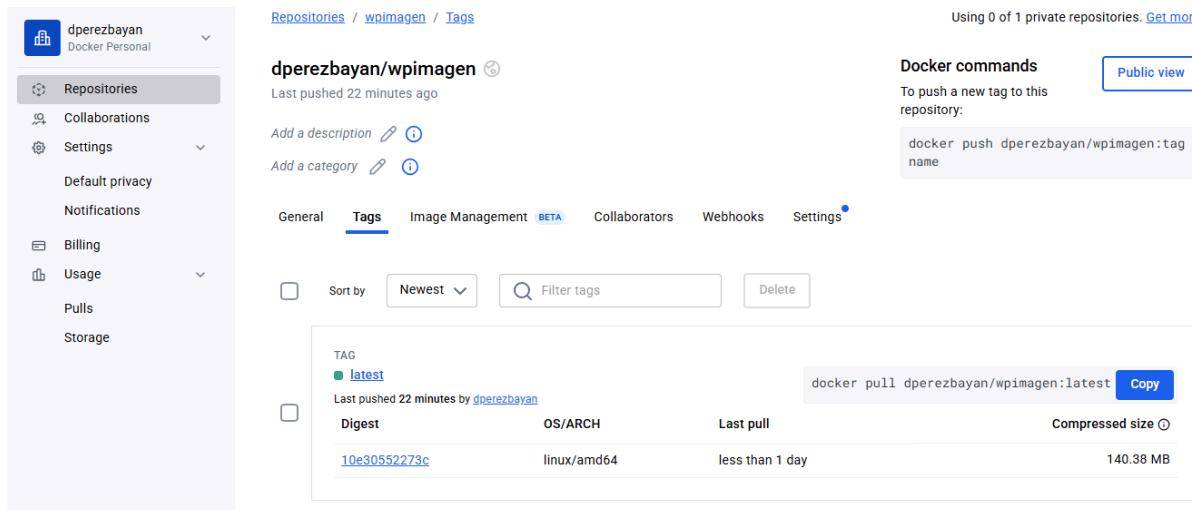
Una vez se tiene la cuenta creada y verificada, hay que loguearse desde la terminal.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ buildah login -u dperezbayan docker.io
Password:
Login Succeeded!
```

Una vez se haya iniciado sesión correctamente, se podrá subir la imagen a DockerHub:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ buildah push wpimagen docker.io/dperezbayan/wpimagen
Getting image source signatures
Copying blob 250fb8a403c done  |
Copying blob cf05a52c0235 skipped: already exists
Copying config 5831e36f02 done  |
Writing manifest to image destination
```

Para verificar que la imagen está creada, se inicia sesión en DockerHub y en repositorios se encuentra la imagen creada anteriormente.



The screenshot shows the DockerHub interface for the repository `dperezbayan/wpimagen`. On the left, there's a sidebar with options like Repositories, Collaborations, Settings, Notifications, Billing, Usage, Pulls, and Storage. The main area shows the repository details: it was last pushed 22 minutes ago, and there are fields to add a description and category. Below this, there are tabs for General, Tags, Image Management (Beta), Collaborators, Webhooks, and Settings. The Tags tab is selected, showing a table with one row. The table has columns for TAG, Digest, OS/ARCH, Last pull, and Compressed size. The single row shows the tag `latest`, digest `10e30552273c`, OS/ARCH `linux/amd64`, Last pull `less than 1 day`, and Compressed size `140.38 MB`. There's also a button to copy the Docker pull command: `docker pull dperezbayan/wpimagen:latest`.

4.2.6 Kubectl y K3s.

Este paso permite que se pueda desplegar la aplicación WordPress en clúster K3s. El archivo `/etc/rancher/k3s/k3s.yaml` es la configuración que kubectl necesita para comunicarse con el clúster.

Primero, hay que dar permisos al usuario para manejar el clúster:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ sudo chown -R $USER:$USER /etc/rancher/k3s/k3s.yaml
```

Este paso es clave porque, si no lo haces, kubectl devolverá errores de permisos.

Seguidamente, es hora de crear y configurar los ficheros de configuración para el despliegue del clúster en k3s.

Primero se crea la carpeta k3s, donde se alojarán todos los ficheros de configuración:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ mkdir k3s
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ ls
Dockerfile  k3s  README.md
```

Con el clúster K3s ya instalado y funcional, el siguiente paso consiste en configurar los recursos necesarios para desplegar la aplicación WordPress junto a su base de datos.

Para ello, se ha creado un directorio llamado `k3s/` dentro del proyecto, que contiene todos los ficheros `.yaml` que son necesarios para definir los distintos recursos de Kubernetes. Esta estructura permite mantener organizadas las configuraciones.

Estructura del directorio `k3s/`:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ tree -L 1
.
└── Dockerfile
    ├── ingress.yaml
    └── k3s
        └── README.md

2 directories, 3 files
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ tree k3s/
k3s/
├── mysql-deployment.yaml
├── mysql-srv.yaml
├── pvc-mysql.yaml
├── pvc-wordpress.yaml
├── wordPress-deployment.yaml
├── wordPress-srv.yaml
└── wpdates_configmap.yaml
    └── wppass_secret.yaml

1 directory, 8 files
```

Objetivo de cada fichero:

- **Deployments** (`mysql-deployment.yaml`, `wordPress-deployment.yaml`): Definen qué contenedores correr, cómo deben correr, y qué hacer si fallan, lo que permite un despliegue automático y escalable
- **Services** (`mysql-srv.yaml`, `wordPress-srv.yaml`): Exponen los pods internamente en el clúster para permitir la comunicación entre ellos.
- **PersistentVolumeClaims** (`pvc-mysql.yaml`, `pvc-wordpress.yaml`): Evitan la pérdida de datos si se reinician los contenedores.
- **ConfigMap y Secret**: Sirven para gestionar la configuración externa de las aplicaciones. El ConfigMap contiene el nombre y usuario de la base de datos y el Secret almacena la contraseña codificada en base64.

- **Ingress** (`ingress.yaml`): define una regla de entrada para acceder a WordPress desde el navegador mediante un nombre de dominio personalizado (wordpress.local).

Aplicación de los recursos:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl apply -f k3s/
```

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl apply -f ingress.yaml
```

Esto permite crear en el clúster todos los componentes necesarios de forma automatizada.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get pods
```

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get svc
```

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get pvc
```

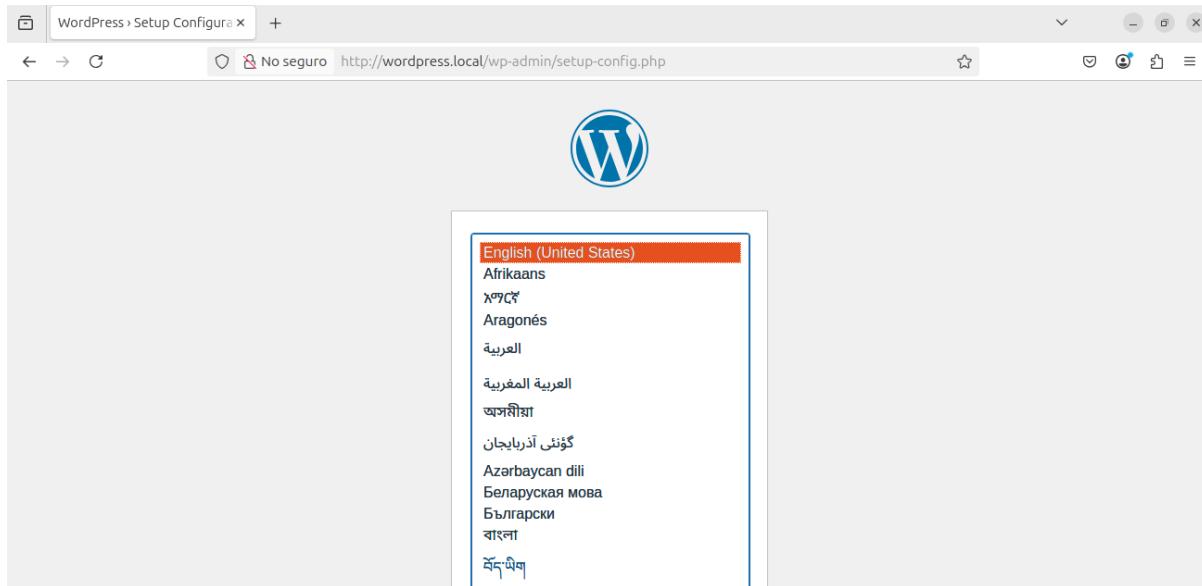
```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get ingress
```

El despliegue permite lanzar WordPress usando los valores de configuración definidos, y acceder al CMS desde el navegador a través de `http://wordpress.local`, configurando previamente el archivo `/etc/hosts`.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 danieltfg-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1 wordpress.local
```

Este enfoque permite replicar el entorno en otros sistemas con sólo aplicar los ficheros YAML, facilitando la automatización y el mantenimiento del entorno de producción o desarrollo.



4.2.7 APPLICACIÓN DE INTEGRACIÓN CONTINUA Y DESPLIEGUE CONTINUO (CI/CD)

Con el objetivo de automatizar el despliegue de la aplicación, se ha aplicado un sistema de integración y despliegue continuo utilizando Jenkins.

Este sistema permite que, tras realizar un cambio en el repositorio de código, se ejecuten automáticamente tareas como la reconstrucción de imágenes, ejecución de pruebas y actualización del clúster con la nueva versión de la aplicación, eliminando errores manuales y mejorando la eficiencia del despliegue.

4.2.8 INSTALACIÓN Y CONFIGURACIÓN DE Jenkins EN K3s MEDIANTE Helm.

Jenkins simplifica la automatización de las fases más importantes del desarrollo y la implementación de aplicaciones.

- Integración continua (CI): Ejecuta automáticamente tareas (construcción de imágenes) cada vez que se detectan cambios en el repositorio Git.
- Despliegue continuo (CD): Automatiza la implementación de la aplicación en el clúster, reduciendo errores y tiempos de espera.
- Pipelines: Jenkins permite definir procesos complejos en secuencias estructuradas, como: clonar repositorios, construir imágenes y aplicar despliegues con kubectl.

Jenkins se ha instalado a través de Helm, una herramienta de gestión de paquetes para Kubernetes.

Para la instalación de helm:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

Se verifica que Helm está correctamente instalado:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ helm version
version.BuildInfo{Version:"v3.18.0", GitCommit:"cc58e3f5a3aa615c6a86275e6f4444b5fdd3cc4e", GitTreeState:"clean", GoVersion:"go1.24.3"}
```

Para la instalación de Jenkins, se añade a los repositorios de helm el repositorio oficial de Jenkins:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ helm repo add jenkins https://charts.jenkins.io
"jenkins" has been added to your repositories
```

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "jenkins" chart repository
Update Complete. *Happy Helming!*
```

Después, es recomendable antes de instalar Jenkins crear un namespace para mantener una organización dentro del clúster:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl create namespace jenkins
namespace/jenkins created
```

Con el namespace creado, se instala Jenkins desde los repositorios de helm. Esto creará todos los recursos necesarios: pods, servicios, volúmenes, etc.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ helm install jenkins jenkins/jenkins -n jenkins
```

Se comprueba que todo se ha instalado correctamente:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get all -n jenkins
NAME           READY   STATUS    RESTARTS   AGE
pod/jenkins-0  2/2     Running   0          4m26s

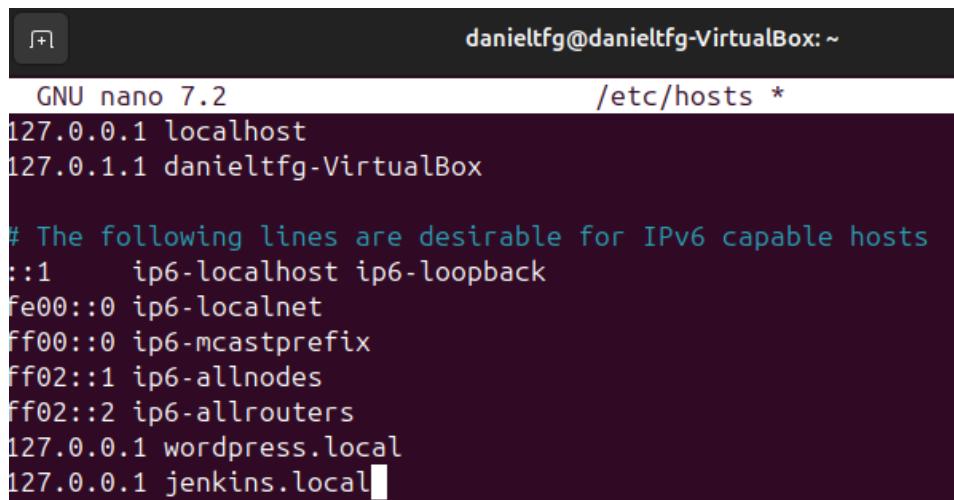
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/jenkins       ClusterIP   10.43.161.102  <none>        8080/TCP   4m26s
service/jenkins-agent ClusterIP   10.43.244.160  <none>        50000/TCP  4m26s

NAME           READY   AGE
statefulset.apps/jenkins  1/1     4m26s
```

Una vez instalado, es necesario obtener la contraseña del administrador para acceder por primera vez a la interfaz web de Jenkins. Esta contraseña se encuentra en un secreto dentro del clúster:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ jsonpath=".data.jenkins-admin-password"
secret=$(kubectl get secret -n jenkins jenkins -o jsonpath=$jsonpath)
echo $(echo $secret | base64 --decode)
P0ogzhMr3xFvFoQnfiHdhQ
```

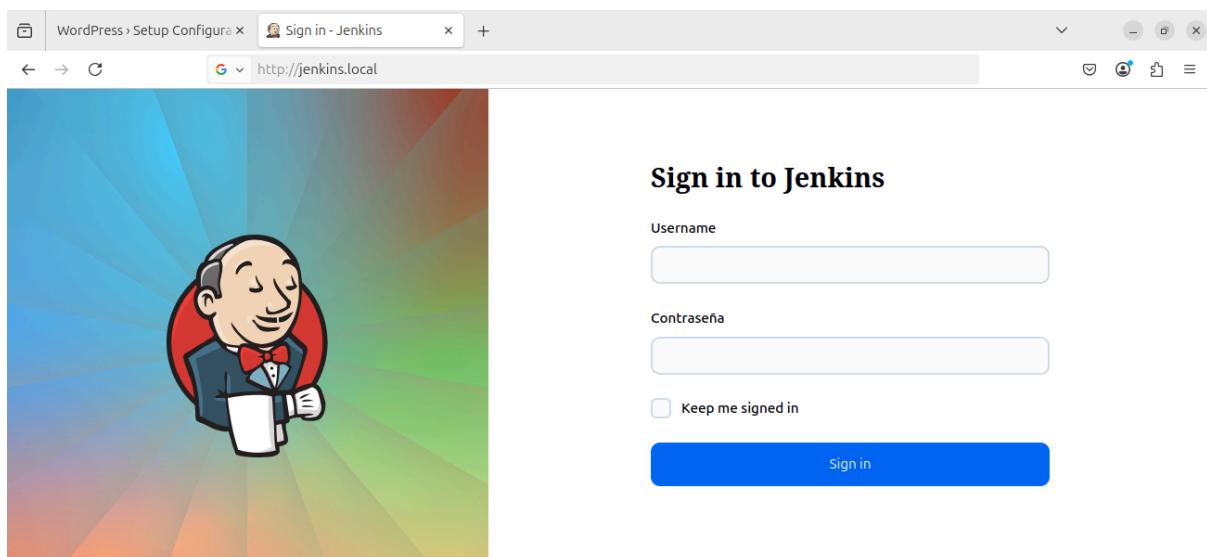
Una vez se obtiene la contraseña, se añade al archivo `/etc/hosts`:



```
GNU nano 7.2          /etc/hosts *
127.0.0.1 localhost
127.0.1.1 danieltfg-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1 wordpress.local
127.0.0.1 jenkins.local
```

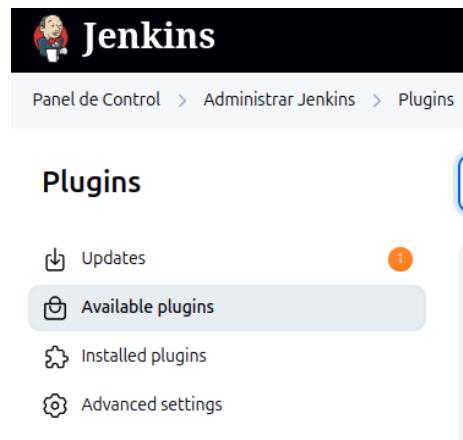
Gracias a esta configuración, se puede acceder a Jenkins desde el navegador.



Como usuario pondremos: **admin**

Como contraseña utilizaremos: **POogzhMr3xFVFoQnfiHdhQ**

Una vez está desplegado Jenkins, antes de configurar el pipeline, es necesario instalar algunos plugins. Para ello, desde el panel de control de Jenkins:



Y desde el buscador, introducir los nombres de los plugins que se van a necesitar, para este proyecto se han instalado los siguientes plugins:

- **Kubernetes plugin.** Permite integrar Jenkins con clústeres de Kubernetes como agente de ejecución.

Kubernetes plugin 4340.v345364d31a_2a_
This plugin integrates Jenkins with [Kubernetes](#)
[Report an issue with this plugin](#)



- **Kubernetes CLI Plugin.** Permite ejecutar comandos kubectl directamente desde las tareas de Jenkins.

Kubernetes CLI Plugin 1.364.vadef8cb8b823
Configure kubectl for Kubernetes
[Report an issue with this plugin](#)



- **GitHub plugin.** Facilita la integración del repositorio de GitHub con Jenkins, permitiendo usar webhooks.

[GitHub plugin 1.43.0](#)

This plugin integrates [GitHub](#) to Jenkins.

[Report an issue with this plugin](#)



- **Git plugin.** Permite clonar y manejar repositorios Git dentro de los pipelines.

[Git plugin 5.7.0](#)

This plugin integrates [Git](#) with Jenkins.

[Report an issue with this plugin](#)



- **GitHub Integrations.** Mejora la compatibilidad con GitHub como fuente de control de versiones y automatización.

[GitHub Integration Plugin 0.7.2](#)

GitHub Integration Plugin for Jenkins



- **SSH Agent Plugin.** Necesario para usar claves SSH para autenticación.

[SSH Agent Plugin 386.v36cc0c7582f0](#)

This plugin allows you to provide SSH credentials to builds via a ssh-agent in Jenkins.

[Report an issue with this plugin](#)



- **Pipeline Plugin.** Esencial para crear y ejecutar pipelines definidos mediante Jenkinsfile.

[Pipeline 608.v67378e9d3db_1](#)

A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code with Jenkins](#) for more details.

[Report an issue with this plugin](#)



CREDENCIALES A CONFIGURAR

A continuación, Jenkins necesita credenciales para la automatización de la integración y despliegue continuo, para el proyecto se han creado las siguientes:

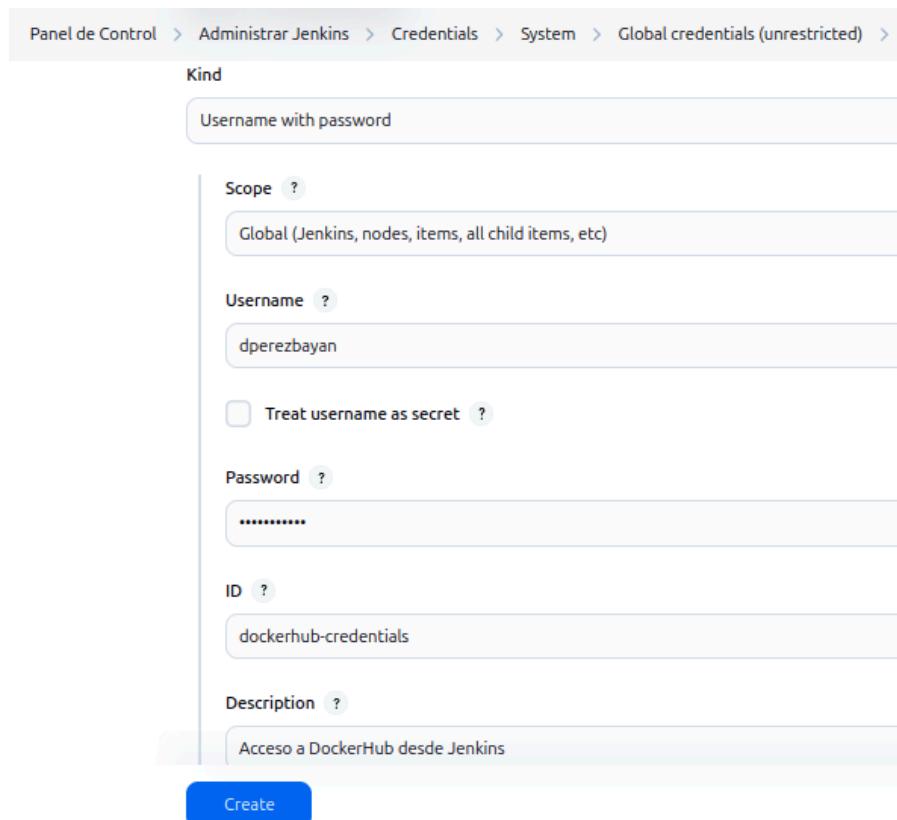
- **GITHUB:**

Permite a Jenkins acceder al repositorio de Github para realizar la operación de clonado del código o detectar cambios en la integración continua automáticamente.

The screenshot shows the Jenkins Global credentials configuration interface. The path in the top navigation bar is: Panel de Control > Administrar Jenkins > Credentials > System > Global credentials (unrestricted). The 'Kind' dropdown is set to 'Username with password'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'dperezbayan'. The 'Password' field is filled with dots. The 'ID' field is 'github-credentials'. The 'Description' field is 'Acceso al repositorio de GitHub desde Jenkins'. A blue 'Create' button is at the bottom left.

- **DOCKER_HUB:**

Al igual que el anterior, se proporcionan los datos de Docker Hub para permitir subir o descargar imágenes Docker.



The screenshot shows the Jenkins 'Global credentials (unrestricted)' configuration page. The 'Kind' is set to 'Username with password'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'dperezbayan'. The 'Password' field contains a redacted password. The 'ID' is 'dockerhub-credentials'. The 'Description' is 'Acceso a DockerHub desde Jenkins'. A blue 'Create' button is at the bottom.

To use the access token from your Docker CLI client:

1. Run

```
$ docker login -u dperezbayan
```

[Copy](#)

2. At the password prompt, enter the personal access token.

```
dckr_pat_7Rzn09rg9jJfJvJ8iYQzB5ap9CI
```

[Copy](#)

[Back to access tokens](#)

- **VPS_SSH:**

Se permite a Jenkins poder conectarse por SSH a la máquina de producción usando una clave privada creada para la conexión.

Este tipo de credencial es de usuario con clave privada SSH, por lo que se le proporciona el usuario y la clave privada de la máquina.

Esta credencial se configura para desplegar automáticamente mediante comandos remotos la aplicación en el entorno de producción.

Se generan el par de claves SSH en la máquina de desarrollo.

```
danieltfg@danieltfg-VirtualBox:~$ ssh-keygen -t rsa -b 4096 -C "jenkins-vps"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/danieltfg/.ssh/id_rsa):
/home/danieltfg/.ssh/id_rsa already exists.
Overwrite (y/n)?
danieltfg@danieltfg-VirtualBox:~$ ls ~/.ssh
authorized_keys  id_rsa  id_rsa.pub
```

Ahora hay que copiar la clave pública a la máquina de producción. Con esto Jenkins podrá autenticarse por SSH:

En primer lugar se instala en la máquina de desarrollo openssh, para hacer posible la conexión.

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt update
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt install openssh-server
```

Después de eso, se inicia el servicio:

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo systemctl start ssh
danieltfg2@danieltfg2-VirtualBox:~$ sudo systemctl enable ssh
```

Ahora, desde la máquina de desarrollo, para verificar que funciona:

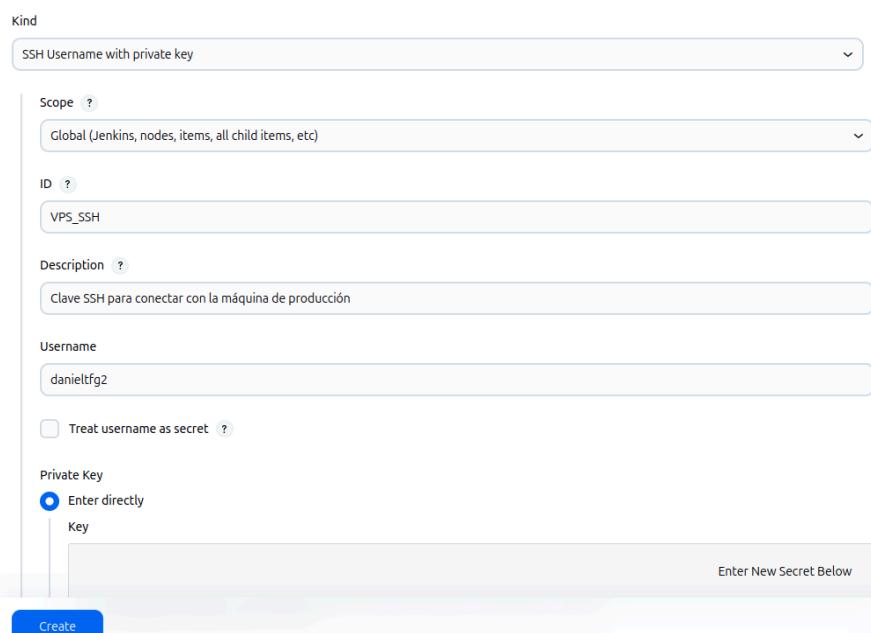
```
danieltfg@danieldfg-VirtualBox:~$ ssh danieldfg2@192.168.100.2
The authenticity of host '192.168.100.2 (192.168.100.2)' can't be established.
ED25519 key fingerprint is SHA256:9xGf2+P/AlaDyd4CJrvtoCdX1I4jyvEMlBPPBewG/IyM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.100.2' (ED25519) to the list of known hosts.
danieldfg2@192.168.100.2's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-26-generic x86_64)
```

```
danieltfg2@danieldfg2-VirtualBox:~$ exit
cerrar sesión
Connection to 192.168.100.2 closed.
danieltfg@danieldfg-VirtualBox:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub danieldfg2@19
2.168.100.2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/danieldfg/.
ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
danieldfg2@192.168.100.2's password:

Number of key(s) added: 1

Now try logging into the machine, with:    "ssh 'danieldfg2@192.168.100.2'"
and check to make sure that only the key(s) you wanted were added.
```

Ahora, hay que crear la credencial en Jenkins (VPS_SSH)



The screenshot shows the Jenkins configuration page for a 'SSH Username with private key' credential. The fields filled in are:

- Kind:** SSH Username with private key
- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- ID:** VPS_SSH
- Description:** Clave SSH para conectar con la máquina de producción
- Username:** danieldfg2
- Treat username as secret:** Unchecked
- Private Key:** Enter directly (radio button selected)
- Key:** A large text area containing the SSH private key.
- Enter New Secret Below:** A link to enter a new secret.

Tras la configuración de credenciales que necesita Jenkins para poder ejecutar el pipeline, se crea una nueva tarea:

Global credentials (unrestricted)

[+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description	Action
 github-credentials	dperezbayan/***** (Acceso al repositorio de GitHub desde Jenkins)	Username with password	Acceso al repositorio de GitHub desde Jenkins	
 dockerhub-credentials	dperezbayan/***** (Acceso a DockerHub desde Jenkins)	Username with password	Acceso a DockerHub desde Jenkins	
 VPS_SSH	danielcfg2 (Clave SSH para conectar con la máquina de producción)	SSH Username with private key	Clave SSH para conectar con la máquina de producción	

New Item

Enter an item name

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



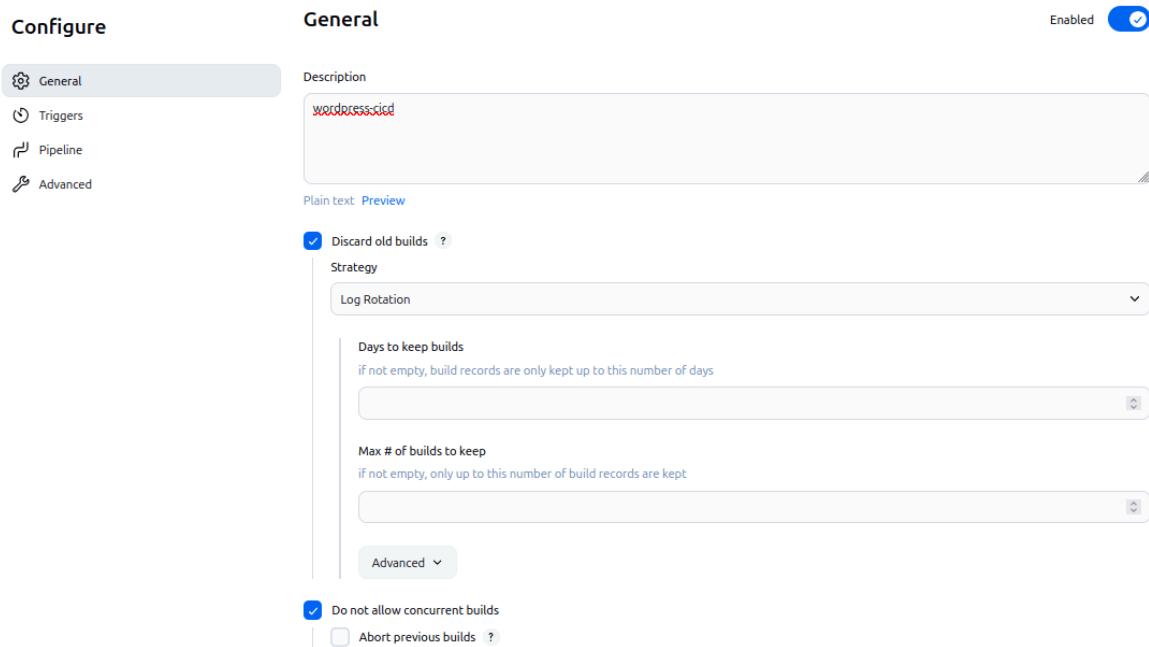
Multibranch Pipeline

Creates a set of parallel projects according to detected branches in one SCM repository.

[OK](#)

Ahora hay que configurar la nueva tarea.

En la configuración general, se marca “Discard old builds”, para eliminar builds antiguos y evitar acumulación innecesaria. También, se marca “Do not allow concurrent builds”, para evitar que se ejecuten varios despliegues al mismo tiempo.



The screenshot shows the Jenkins 'General' configuration page for a job named 'wordpress-cicd'. The 'Enabled' checkbox is checked. The 'Description' field contains 'wordpress-cicd'. Under the 'Discard old builds' section, the 'Log Rotation' strategy is selected. The 'Days to keep builds' field is set to 7. The 'Max # of builds to keep' field is set to 10. The 'Advanced' button is visible. At the bottom, the 'Do not allow concurrent builds' checkbox is checked, while the 'Abort previous builds' checkbox is unchecked.

En la configuración de los triggers se usa el plugin **Generic Webhook Trigger** para que Jenkins detecte automáticamente cambios en GitHub. Extrae la rama del push (como main) y la guarda en una variable (git_branch), que se usa en el pipeline. También se añade un token de seguridad (despliegue2025).

Configure

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

Build after other projects are built ?

Build periodically ?

Generic Webhook Trigger ?

Is triggered by HTTP requests to http://JENKINS_URL/generic-webhook-trigger/invoke

There are example configurations in [the Git repository](#).

You can fiddle with JSONPath [here](#). You may also want to checkout the syntax [here](#).

You can fiddle with XPath [here](#). You may also want to checkout the syntax [here](#).

You can fiddle with regular expressions [here](#). You may also want to checkout the syntax [here](#).

If your job is **not parameterized**, then the resolved variables will just be contributed to the build. If your job is **parameterized**, and you resolve variables that have the same name as those parameters, then the plugin will populate the parameters when triggering job. That means you can, for example, use the parameters in combination with an SCM plugin, like GIT Plugin, to pick a branch.

Post content parameters

Variable Name of variable git_branch

Expression \$ref

JSONPath

XPath

Panel de Control > wordpress-ci-cd > Configuration

Configure

Triggers

Header parameters

Añadir

If you want value of header param1 to be contributed, you need to add "param1" here.

Request parameters

Añadir

If you want value of query parameter param1 to be contributed, you need to add "param1" here.

Token

despliegue2025

Se selecciona la opción **Pipeline script from SCM**, indicando que el pipeline estará definido en el repositorio de GitHub dentro del archivo Jenkinsfile.

- En SCM se selecciona Git ya que el proyecto está almacenado en un proyecto Git.
- Se pone la URL del repositorio GitHub.
- Se selecciona la credencial de GitHub que se creó anteriormente.
- Se pone el nombre de la rama del repositorio de Git (main).

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

<https://github.com/dperezbayan/TFG-WordPress-K3s.git>

Credentials ?

dperzbayan/******** (Acceso al repositorio de GitHub desde Jenkins)

+ Add

Advanced ▾

Add Repository

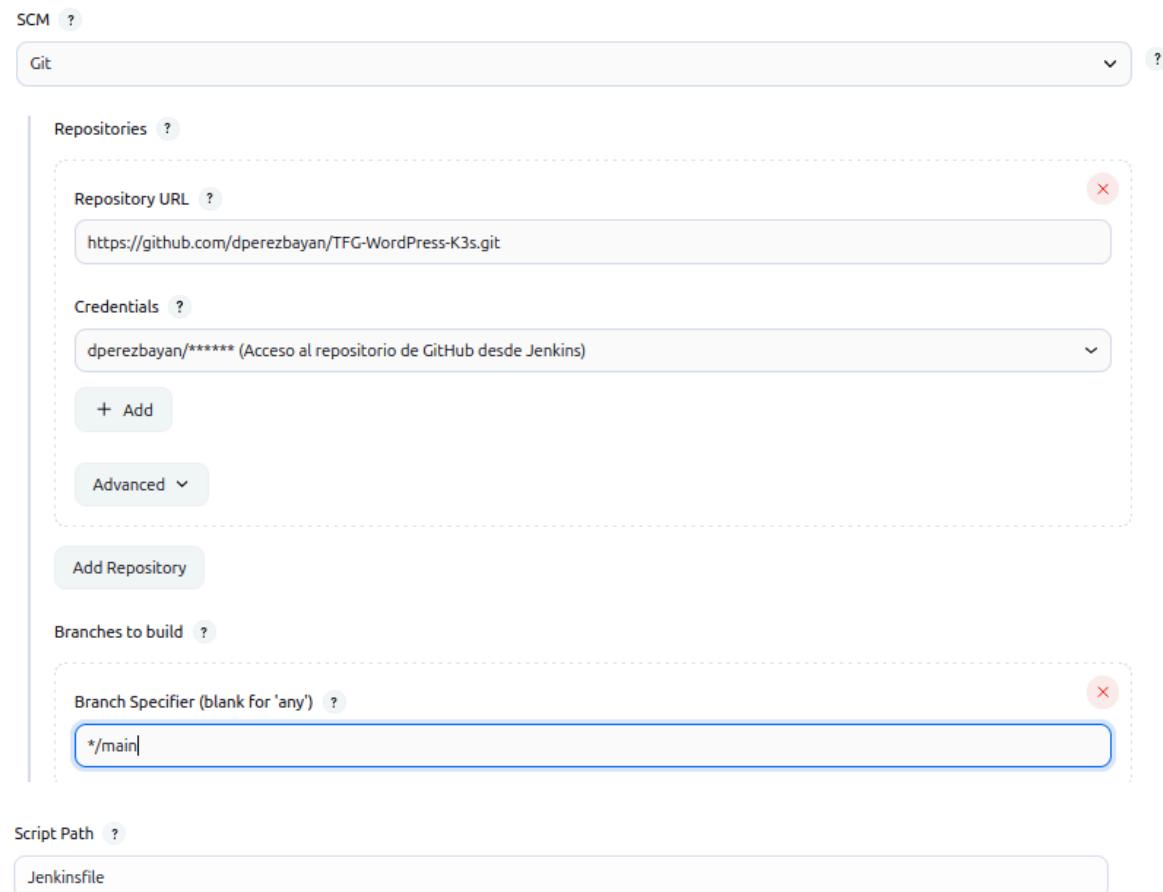
Branches to build ?

Branch Specifier (blank for 'any') ?

*/main|

Script Path ?

Jenkinsfile



Para permitir la creación y subida de imágenes desde Jenkins, se utilizará. Aunque Buildah no requiere de un demonio como Docker para su funcionamiento, es necesario usar Docker para construir una imagen personalizada que incluya Jenkins y Buildah preinstalado. Se crea: [buildah:v2](#). se sube a DockerHub y se utilizan los pods de Jenkins para ejecutar el pipeline sin depender de Docker

Docker solo se usó una vez para preparar una imagen personalizada
Buildah se usará en esa imagen para construir otras imágenes sin Docker
El objetivo es usar Buildah dentro del pipeline, no depender de Docker en producción o en Jenkins.

Se actualiza el sistema:

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt update
```

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt upgrade -y
```

Se instalan las dependencias necesarias:

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Se añade la clave GPG oficial de Docker.

```
danieltfg@danieltfg-VirtualBox:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
    sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
[sudo] contraseña para danieltfg:
```

Añadir el repositorio de Docker:

```
danieltfg@danieltfg-VirtualBox:~$ echo \
    "deb [arch=$(dpkg --print-architecture) \
    signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Actualizar e instalar Docker:

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt update
```

```
danieltfg@danieltfg-VirtualBox:~$ sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

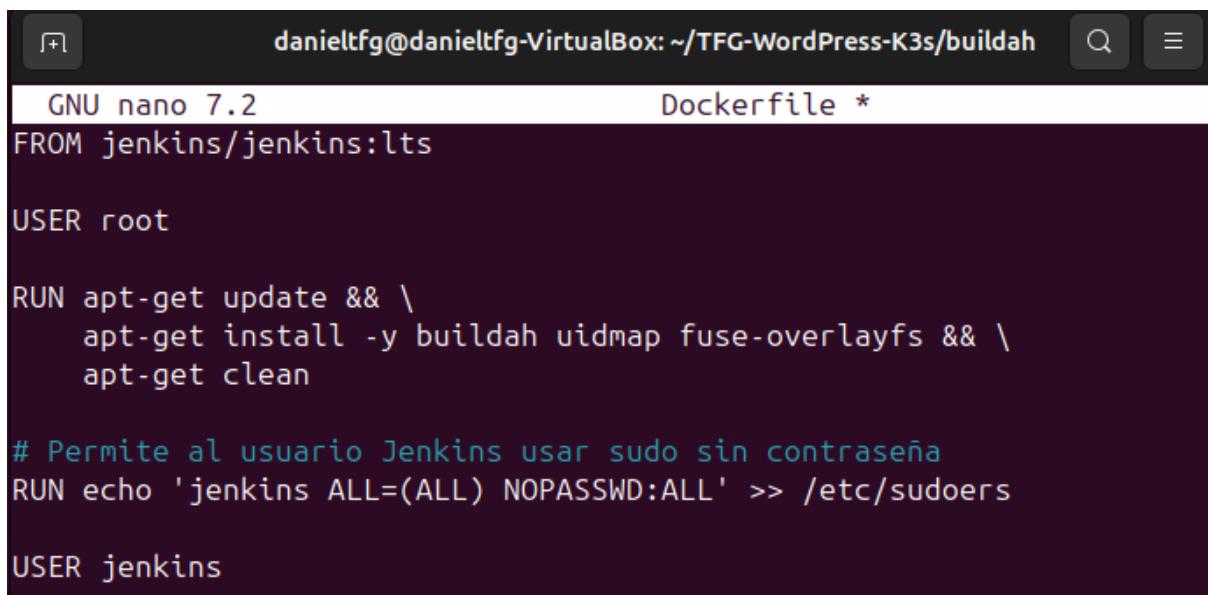
Ahora, se seguirán unos pasos para crear y subir buildah:v2

Se crea la carpeta **buildah** para subir el **Dockerfile**.

```
danieltfg@danieltfg-VirtualBox:~$ cd TFG-WordPress-K3s/  
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ mkdir buildah  
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ ls  
buildah Dockerfile ingress-jenkins.yaml ingress.yaml k3s README.md
```

Se crea el archivo **Dockerfile**, que se resume en que usa Jenkins oficial como base.

Añade Buildah y sus dependencias. Da permisos al usuario Jenkins. Deja todo preparado para que el pipeline pueda crear imágenes sin errores ni limitaciones de permisos.



```
GNU nano 7.2                               Dockerfile *
FROM jenkins/jenkins:lts

USER root

RUN apt-get update && \
    apt-get install -y buildah uidmap fuse-overlayfs && \
    apt-get clean

# Permite al usuario Jenkins usar sudo sin contraseña
RUN echo 'jenkins ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

USER jenkins
```

Ahora, se construye la imagen con Docker:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s/buildah$ sudo docker build -t buildah:v2 .
[+] Building 59.7s (7/7) FINISHED
          docker:default
```

Por último hay que subir la imagen a Docker Hub:

Hay que loguearse en Docker:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s/buildah$ docker login
USING WEB-BASED LOGIN
Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: RQSK-NWMV
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...

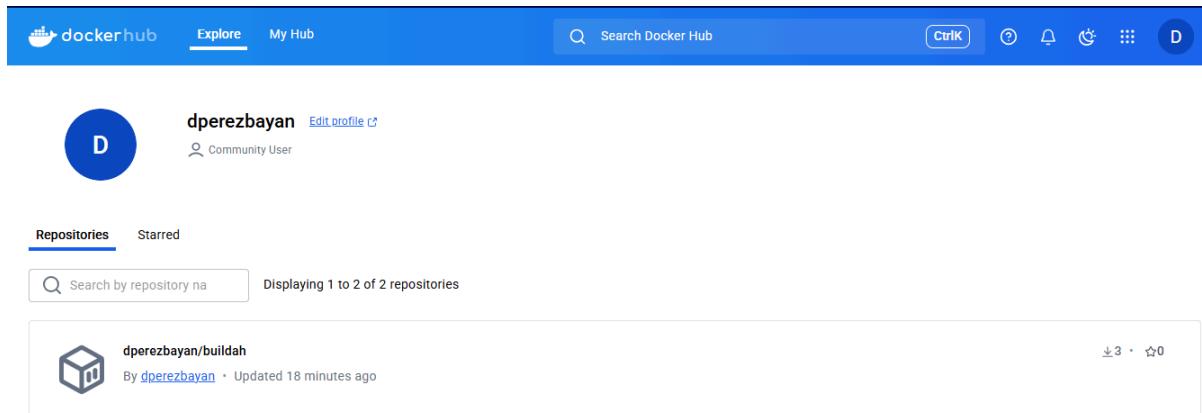
WARNING! Your credentials are stored unencrypted in '/home/danieltfg/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
```

Se etiqueta la imagen con el usuario de Docker Hub y se sube la imagen.

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s/buildah$ sudo docker tag buildah:v2 dperezbayan/buildah:v2
sudo docker push dperezbayan/buildah:v2
The push refers to repository [docker.io/dperezbayan/buildah]
f9ca76392af3: Pushed
0eadcef80254: Pushed
363e016c5f0b: Mounted from jenkins/jenkins
4565a8899313: Mounted from jenkins/jenkins
698edb719999: Mounted from jenkins/jenkins
65f1f97e0e3b: Mounted from jenkins/jenkins
41f03a185bbb: Mounted from jenkins/jenkins
82170217dd67: Mounted from jenkins/jenkins
e5125a458148: Mounted from jenkins/jenkins
97b5962aa379: Mounted from jenkins/jenkins
57c23611249a: Mounted from jenkins/jenkins
e649b9d6f7db: Mounted from jenkins/jenkins
1ba1a28d4a04: Mounted from jenkins/jenkins
247ffffb7158d: Mounted from jenkins/jenkins
v2: digest: sha256:841b1462e7fc165e76472dcfd79aeadf1e2f36e83cd6959e1e427e3bc1ce1172 size: 3252
```

Se comprueba que se ha subido la imagen a Docker Hub:



The screenshot shows the Docker Hub interface. At the top, there's a blue header bar with the Docker Hub logo, navigation links for 'Explore' and 'My Hub', a search bar labeled 'Search Docker Hub', and various user icons. Below the header, the user profile 'dperezbayan' is displayed, showing the option to 'Edit profile'. It also indicates the user is a 'Community User'. Under the profile, there are two tabs: 'Repositories' (which is selected) and 'Starred'. A search bar below the tabs allows searching by repository name. The main content area displays '2 repositories' found, with one repository listed: 'dperezbayan/buildah'. This repository was created by 'dperezbayan' and updated 18 minutes ago. To the right of the repository card, there are icons for forks (3), stars (0), and other actions.

Una vez creada y subida la imagen a Docker Hub, se configura el fichero [Jenkinsfile](#).

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ sudo nano Jenkinsfile
```

Este pipeline se desarrolla de la siguiente forma: [Jenkinsfile](#)

Ahora, se verá una revisión de los bloques que componen este fichero tan importante.

1. Variables globales reutilizables.

Al inicio, se define el bloque `environment`, en el que se declaran variables reutilizables como el nombre de la imagen (`IMAGE`), la URL del repositorio (`REPO_URL`), las rutas locales (`BUILD_DIR` y `KUBE_CONFIG`), las credenciales de Docker Hub (`DOCKER_HUB`) y la rama que activa la ejecución del pipeline (`GIT_BRANCH`).

2. Agente que crea un pod temporal con Buildah

Mediante la directiva `agent { kubernetes { ... } }` se configura la ejecución del pipeline dentro de un pod temporal gestionado por K3s. El pod utiliza la imagen `dprezbayan/buildah:v2` que contiene la herramienta Buildah. Además, se establece el modo `privileged:true` para la ejecución de posibles operaciones con contenedores dentro del pod

3. Opciones del pipeline

En la sección de `options` para optimizar la ejecución del pipeline:

- `buildDiscarder(logRotator(numToKeepStr: '10'))`, mantiene los últimos 10 builds para optimizar el uso de espacio en disco
- `durabilityHint('PERFORMANCE_OPTIMIZED')`, reduce la frecuencia con la que Jenkins guarda el estado del pipeline, mejorando su rendimiento
- `disableConcurrentBuilds()`, evita la ejecución concurrente del pipeline.

4. Verificación de rama y clonación

En `Verificar rama`, se comprueba que el pipeline ha sido disparado por un push en la rama `main`. Si al verificar sale todo bien, se procede a `Clonar repositorio`, donde se descarga el contenido del repositorio de GitHub en el directorio especificado.

5. Creación y subida de la imagen con Buildah

En `Crear y subir imagen`, se utiliza Buildah para construir una imagen Docker a partir del `Dockerfile` del proyecto. Posteriormente, se loguea en Docker Hub usando las credenciales almacenadas en Jenkins y se sube la imagen etiquetada con el número de build

6. Migración de la base de datos desde la máquina de desarrollo

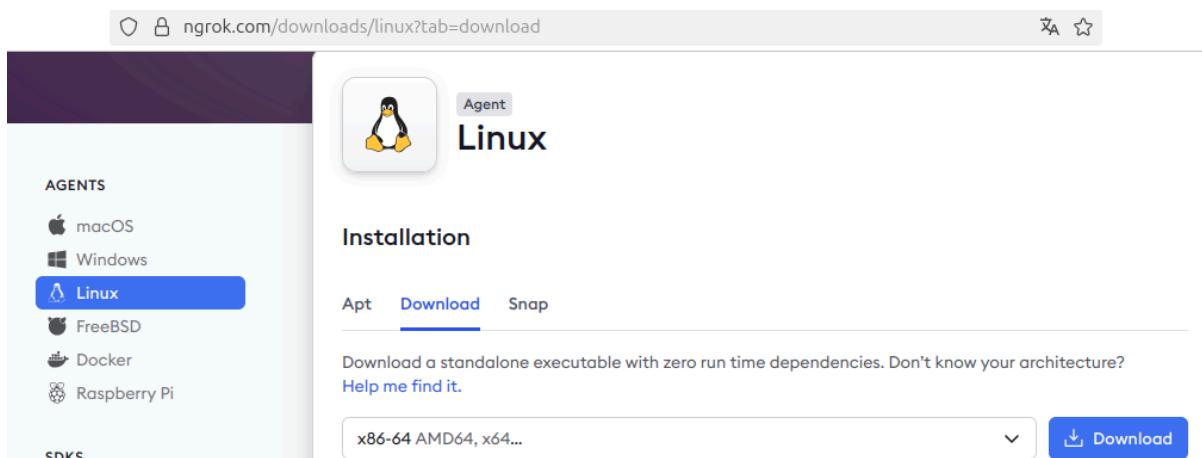
Se utiliza la credencial **VPS_SSH** para establecer una conexión SSH con la máquina de desarrollo. Desde esta, se ejecuta un script que realiza un volcado de la bbdd (**.sql**) y lo transfiere a la máquina de producción mediante **scp**

7. Despliegue en producción con K3s

Por último, en **Deployment**, se conectan por SSH a la máquina de producción, donde se actualiza el repositorio local (**git pull**), se ajustan permisos y se aplica la configuración de Kubernetes usando **kubectl** con el fichero **k3s.yaml**. Además, se ejecuta un script que importa la base de datos y finaliza el despliegue

4.2.9 Ngrok y webhook.

En primer lugar, hay que descargar y descomprimir el archivo ngrok.



```
danieltfg@danieltfg-VirtualBox:~/Descargas$ tar -xvzf ngrok-v3-stable-linux-amd64.tgz
ngrok
danieltfg@danieltfg-VirtualBox:~/Descargas$ sudo mv ngrok /usr/local/bin/
[sudo] contraseña para danieltfg:
danieltfg@danieltfg-VirtualBox:~/Descargas$ ngrok version
ngrok version 3.22.1
```

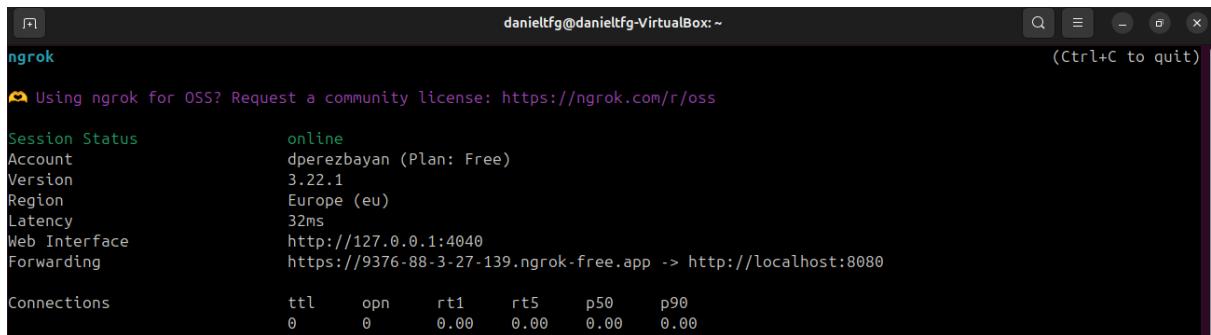
```
danieltfg@danieltfg-VirtualBox:~$ ngrok config add-authtoken 2xhJMFagsQ0wfpXLehI  
h6gdslr5_4sMubEaRH8qbL2JM47Nx  
Authtoken saved to configuration file: /home/danieltfg/.config/ngrok/ngrok.yml
```

De esta manera, Ngrok ya está autenticado correctamente.

Si se ejecuta este comando, se abre un túnel a Jenkins, donde se podrá ver la URL pública.

```
danieltfg@danieltfg-VirtualBox:~$ ngrok http 8080
```

en este caso: <https://9376-88-3-27-139.ngrok-free.app> -> <http://localhost:8080>.



```
danieltfg@danieltfg-VirtualBox:~  
ngrok  
🕒 Using ngrok for OSS? Request a community license: https://ngrok.com/r/oss  
  
Session Status      online  
Account            dperezbayan (Plan: Free)  
Version             3.22.1  
Region              Europe (eu)  
Latency             32ms  
Web Interface      http://127.0.0.1:4040  
Forwarding          https://9376-88-3-27-139.ngrok-free.app -> http://localhost:8080  
  
Connections        ttl     opn     rt1     rt5     p50     p90  
                   0       0     0.00    0.00    0.00    0.00
```

Ahora hay que configurar el Webhook en GitHub:

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type *

Secret

SSL verification
 By default, we verify SSL certificates when delivering payloads.
 Enable SSL verification **Disable (not recommended)**

Which events would you like to trigger this webhook?

Just the **push** event.
 Send me **everything**.
 Let me select individual events.

Active
We will deliver event details when this hook is triggered.

Add webhook

Hay que tener en cuenta que la URL del webhook en GitHub deja de funcionar si se reinicia Ngrok, debido a las limitaciones del plan gratuito de Ngrok, por lo tanto habría que estar actualizando la URL cada vez que se cierre Ngrok.

Webhooks

[Add webhook](#)

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

- [https://c6af-88-3-27-139.ngrok-free... \(push\)](https://c6af-88-3-27-139.ngrok-free... (push))

[Edit](#) [Delete](#)

This hook has never been triggered.

Una vez añadida la URL en webhook, GitHub debería enviar un POST a esa URL y Jenkins debería ejecutar el pipeline automáticamente.

```
danielfg@danielfg-VirtualBox: ~
(Ctrl+C to quit)

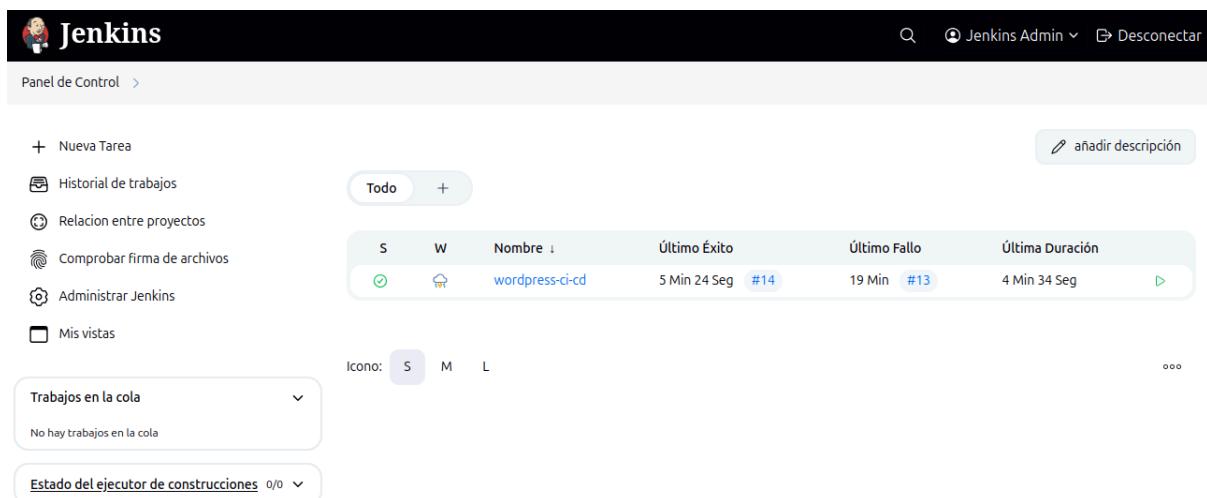
ngrok
Using ngrok for OSS? Request a community license: https://ngrok.com/r/oss

Session Status      online
Account             dperezbayan (Plan: Free)
Version             3.22.1
Region              Europe (eu)
Latency             30ms
Web Interface       http://127.0.0.1:4040
Forwarding          https://bcb2-212-169-233-166.ngrok-free.app -> http://localhost:8080

Connections         ttl     opn      rt1      rt5      p50      p90
                    1        0       0.01    0.00    31.42    31.42

HTTP Requests
-----
09:54:28.036 CEST POST /generic-webhook-trigger/invoke 200 OK
```

Por último, se puede acceder a Jenkins a través de la URL que proporciona Ngrok. Con el Jenkinsfile ordenado y elaborado de la mejor manera posible, además de haber creado las credenciales correctamente, se habrá conseguido que todo el pipeline CI/CD funcione de principio a fin.



The screenshot shows the Jenkins dashboard with the following details:

- Jobs:**wordpress-ci-cd
- Last Success:** 5 Min 24 Seg
- Last Failure:** 19 Min #13
- Duration:** 4 Min 34 Seg

El resultado completo de la ejecución del pipeline puede consultarse en:

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>

Ahora se comentará brevemente cada stage del Jenkinsfile:

- Stage: **Verificar rama**

Evita ejecuciones en ramas no deseadas. Solo permite que se ejecute el pipeline si se hace push a `main`.

- Stage: **Clonar repositorio**

Se realiza una copia del repositorio para acceder a los archivos necesarios, como el `Dockerfile` y los manifiestos de Kubernetes.

- Stage: **Crear imagen Docker con Buildah**

Se construye una imagen de WordPress personalizada usando Buildah. La imagen se etiqueta con un número incremental.

- Stage: **Subir imagen a Docker Hub**

Autentica contra Docker Hub usando credenciales seguras y sube la imagen creada.

- Stage: **Exportar base de datos**

Desde la máquina de desarrollo, realiza un `mysqldump` de la base de datos de WordPress y transfiere el archivo `.sql` a la máquina de producción por `scp`.

- Stage: **Desplegar en producción**

En la máquina de producción:

- Se hace `git pull` del repositorio.
- Se importa la base de datos (`mysql < backup.sql`).
- Se aplican los manifiestos de Kubernetes (`kubectl apply -f k3s/`).

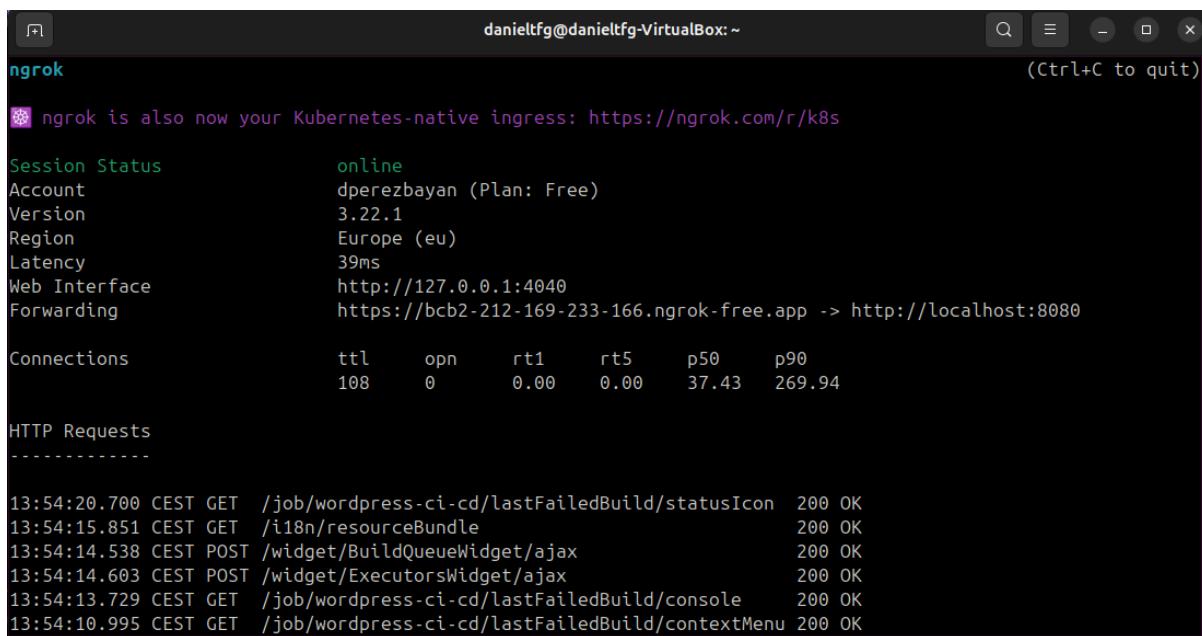
Un punto muy importante para que Jenkins pueda recibir correctamente las peticiones desde GitHub (mediante el webhook configurado), es necesario exponer temporalmente el servicio de Jenkins a Internet. Gracias a ngrok, que actúa como túnel entre un puerto local y una URL pública. Es imprescindible mantener esta terminal abierta. Si se reinicia la máquina o se apaga, a parte de que habrá que volver a exponer el servicio. Hay que comprobar qué proceso está usando el puerto 8080 (`sudo lsof -i :8080`). Matar el proceso (`sudo kill [PID]`). Y se vuelve a exponer el servicio.

```
danieltfg@danieltfg-VirtualBox:~$ kubectl port-forward -n jenkins service/jenkins 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
```

Por otro lado, para crear un túnel seguro desde una URL pública hasta el localhost:8080, se utiliza:

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ ngrok http 8080
```

También es imprescindible mantener esta terminal abierta.



```
danieltfg@danieltfg-VirtualBox:~
```

```
ngrok
```

```
Session Status      online
Account            dperezbayan (Plan: Free)
Version           3.22.1
Region            Europe (eu)
Latency          39ms
Web Interface    http://127.0.0.1:4040
Forwarding        https://bcb2-212-169-233-166.ngrok-free.app -> http://localhost:8080

Connections       ttl     opn      rt1      rt5      p50      p90
                  108      0     0.00     0.00    37.43   269.94

HTTP Requests
-----
13:54:20.700 CEST GET  /job/wordpress-ci-ci/lastFailedBuild/statusIcon  200 OK
13:54:15.851 CEST GET  /i18n/resourceBundle  200 OK
13:54:14.538 CEST POST /widget/BuildQueueWidget/ajax  200 OK
13:54:14.603 CEST POST /widget/ExecutorsWidget/ajax  200 OK
13:54:13.729 CEST GET  /job/wordpress-ci-ci/lastFailedBuild/console  200 OK
13:54:10.995 CEST GET  /job/wordpress-ci-ci/lastFailedBuild/contextMenu 200 OK
```

Para visualizar WordPress en la máquina de desarrollo:

Una vez desplegado Wordpress en el clúster K3s y funcionando de una manera adecuada, se crea un Ingress adicional que permite el acceso vía IP.

```
GNU nano 7.2                                     ingress-dev.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: wordpress-ingress-dev
  annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: web
spec:
  rules:
  - http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: wordpress
          port:
            number: 80
```

```
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get ingress
NAME           CLASS   HOSTS          ADDRESS   PORTS   AGE
wordpress-ingress   traefik  wordpress.local  10.0.2.15  80      8d
wordpress-ingress-dev   traefik  *              10.0.2.15  80      2m
danieltfg@danieltfg-VirtualBox:~/TFG-WordPress-K3s$ kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/mysql-7bbd687c88-cdstz   1/1     Running   15 (52m ago)  8d
pod/wordpress-648b66589b-tqwh4 1/1     Running   15 (52m ago)  8d

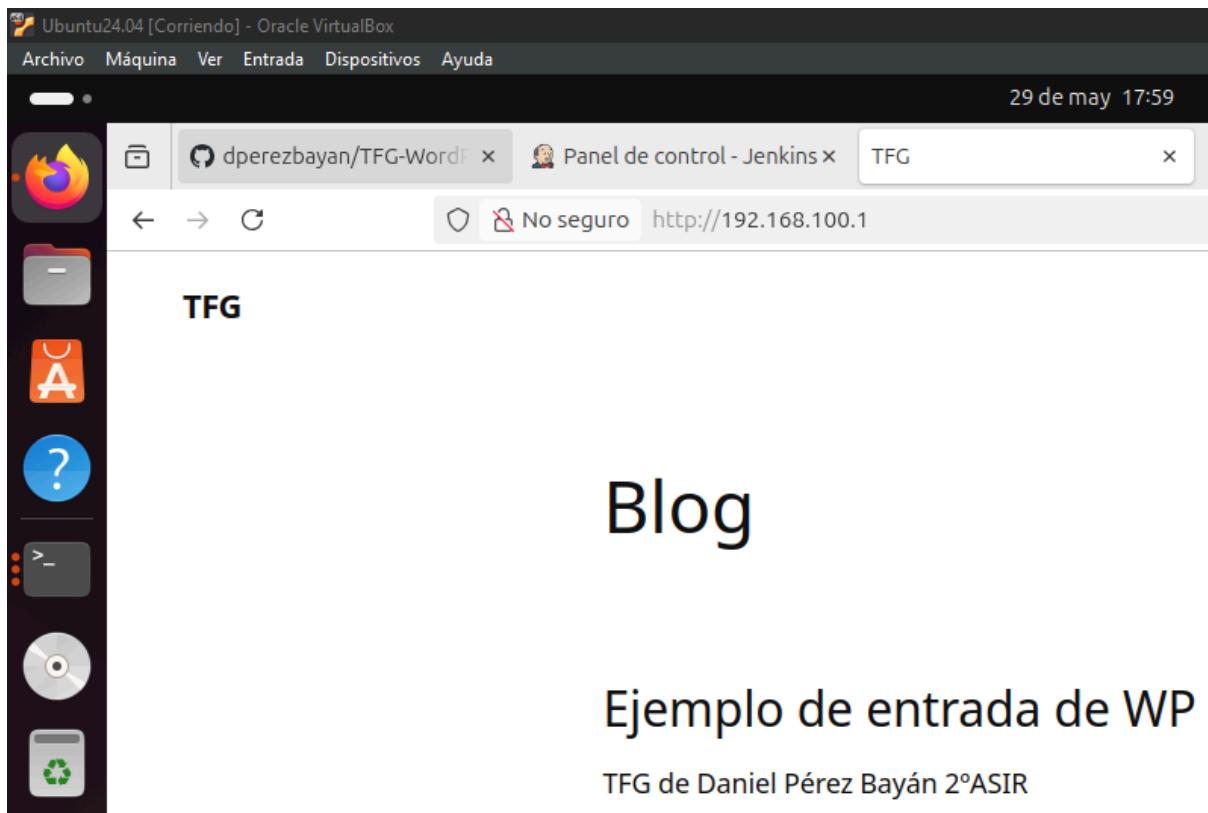
NAME             TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP  10.43.0.1   <none>        443/TCP   31d
service/mysql-service   ClusterIP  10.43.98.234 <none>        3306/TCP  8d
service/wordpress   ClusterIP  10.43.216.132 <none>        80/TCP   8d

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mysql   1/1      1           1           8d
deployment.apps/wordpress 1/1      1           1           8d

NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/mysql-7bbd687c88   1         1         1         8d
replicaset.apps/wordpress-648b66589b 1         1         1         8d
```

Puerto 80 accesible desde el navegador

Desde la máquina de desarrollo, se accede en el navegador a <http://192.168.100.1>



4.3 PREPARACIÓN DEL ENTORNO DE PRODUCCIÓN.

Para simular el entorno de producción dentro del proyecto, se ha creado una segunda máquina virtual (**ubuntupro**). En lugar de repetir todo el proceso de instalación desde cero, se ha seguido el mismo procedimiento descrito en el apartado del entorno de desarrollo.

Para permitir la comunicación entre la máquina de desarrollo y la máquina de producción se ha configurado el adaptador de red de la siguiente forma:

En cuanto a la configuración de red de la máquina tendremos 2 adaptadores

Adaptador	Tipo	Uso
Adaptador 1	NAT	Acceso a Internet (apt, git)
Adaptador 2	Red Interna (intranet)	Comunicación con la VM de desarrollo

Se configura la IP de las máquinas manualmente, ya que sino, las VMs no tendrán dirección IP y no podrán comunicarse entre sí. Esta vez se edita en la máquina de producción.

```

GNU nano 7.2          /etc/netplan/01-network-manager-all.yaml *
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    enp0s8:
      dhcp4: no
      addresses:
        - 192.168.100.2/24
      gateway4: 192.168.100.1
      nameservers:
        addresses:
          - 8.8.8.8

```

4.3.1 INSTALACIÓN DE HERRAMIENTAS NECESARIAS.

Para la integración y despliegue continuo se instalan los siguientes paquetes esenciales:

openssh-server. Permite conexiones SSH desde Jenkins.

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt update
```

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt install openssh-server -y
```

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo systemctl enable --now ssh
```

git. Para clonar el repositorio del proyecto desde GitHub.

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt install git -y
```

mysql-client. Para importar la base de datos desde el archivo .sql exportado

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo apt install mysql-client -y
```

kubectl. Que es la herramienta que se utiliza para controlar el clúster.

```
danieltfg2@danieltfg2-VirtualBox:~$ curl -sL https://get.k3s.io | sh -
```

```
danieltfg2@danieltfg2-VirtualBox:~$ chmod +x kubectl
```

```
danieltfg2@danieltfg2-VirtualBox:~$ sudo mv kubectl /usr/local/bin/
```

4.3.2 CONFIGURACIÓN DE CLAVE SSH.

Se copia la clave pública SSH del usuario danieltfg (máquina de desarrollo), para permitir la conexión automatizada sin contraseña desde Jenkins.

```
danieltfg2@danieltfg2-VirtualBox:~$ ssh-copy-id danieltfg2@192.168.100.2
```

Además, se ha creado la credencial (VPS_SSH) en Jenkins que utiliza la clave privada permitiendo ejecutar comandos remotos como parte del pipeline.

4.3.3 ESTRUCTURA DEL PROYECTO

En /home/danieltfg2/TFG-Wordpress-K3s se ha clonado el repositorio principal

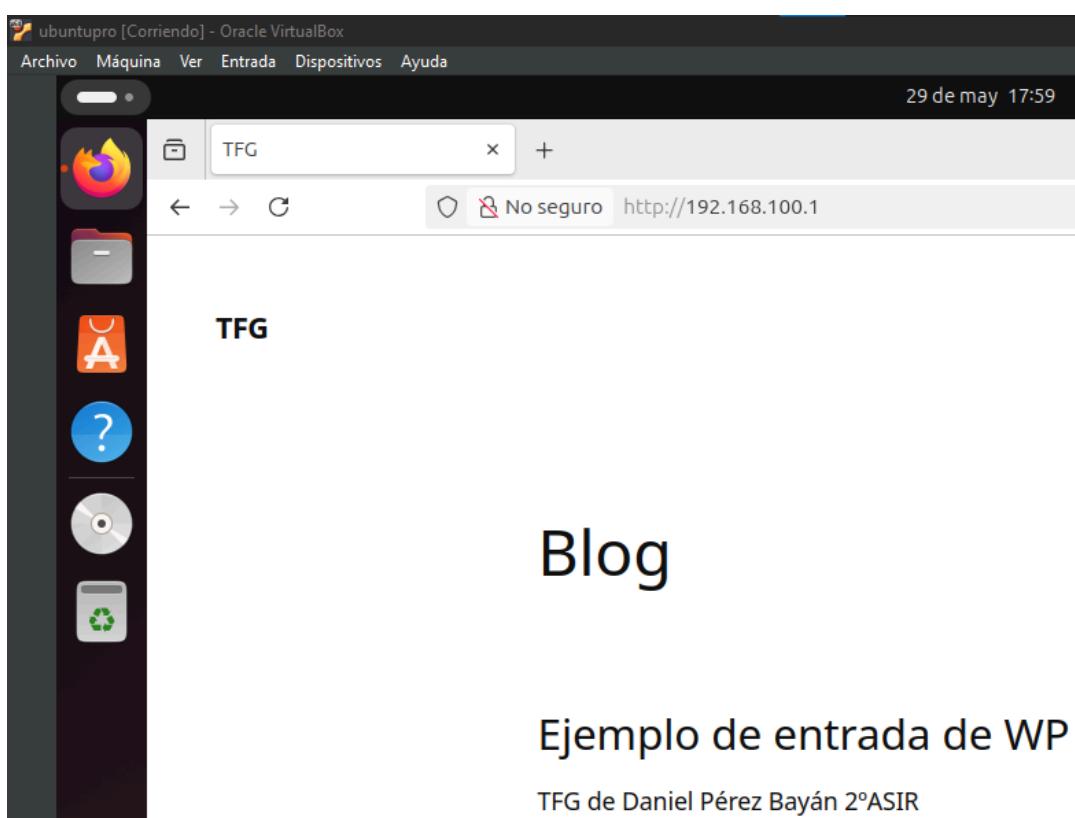
```
danieltfg2@danieltfg2-VirtualBox:~$ git clone https://github.com/dperezbayan/TFG-Wordpress-K3s.git
```

4.3.4 K3s Y ACCESO AL CLÚSTER

Es importante asegurarse de que el usuario de la máquina de producción tenga acceso de lectura al fichero /etc/rancher/k3s/k3s.yaml. Si este fichero es regenerado por una reinstalación de K3s o restauración del sistema, será necesario aplicar de nuevo los permisos mediante

```
sudo chmod +r /etc/rancher/k3s/k3s.yaml  
sudo chown danieltfg2 /etc/rancher/k3s/k3s.yaml
```

Desde la máquina de producción, se accede en el navegador a http://192.168.100.1



Ejemplo de entrada de WP

TFG de Daniel Pérez Bayán 2ºASIR

5. RESULTADOS Y DISCUSIÓN.

5.1 RESULTADO FINAL Y PRUEBAS REALIZADAS.

El objetivo principal del proyecto ha sido implementar un sistema de integración y despliegue continuo utilizando Jenkins, Buildah y K3s para desplegar una aplicación de WordPress en entornos de desarrollo y producción. El resultado ha sido satisfactorio, ya que durante el proceso se han realizado diferentes pruebas para verificar la correcta ejecución de cada fase del proyecto

PRUEBA DEL PIPELINE DE JENKINS.

Durante el desarrollo del proyecto, se ejecuta el pipeline de Jenkins varias veces con el objetivo de comprobar que cada fase funcionaba bien.

1. Construcción de la imagen Docker:

Cada ejecución del pipeline genera una nueva imagen Docker utilizando Buildah.

El log del pipeline donde aparece:

```
Successfully tagged localhost/dperezbayan/wpimagen:14
96d24a7b76994daec5bc2910d44f65c6f6d0d1fd272b26cd449f28f9f756b805
[Pipeline] withCredentials
Masking supported pattern matches of $DOCKER_PASS
[Pipeline] {
[Pipeline] sh
+ echo ****
+ buildah login -u dperezbayan --password-stdin docker.io
Login Succeeded!
[Pipeline] sh
+ buildah push dperezbayan/wpimagen:14 docker://docker.io/dperezbayan/wpimagen:14
```

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>

2. Subida a Docker Hub.

Tras la construcción de la imagen, es autenticada y subida automáticamente al repositorio Docker Hub.

El log del pipeline donde aparece:

```
Login Succeeded!
[Pipeline] sh
+ buildah push dperezbayan/wpimagen:14 docker://docker.io/dperezbayan/wpimagen:14
Getting image source signatures
Copying blob sha256:9a4eb75731c70b74cee93d17b430f48582f130afda047f00250687e0bdbff751
Copying blob sha256:2f7436e79a0bc6cdec6536da54ae6a5863c8e3b5f145e0f8ac0b96306baddbc9
Copying config sha256:96d24a7b76994daec5bc2910d44f65c6f6d0d1fd272b26cd449f28f9f756b805
Writing manifest to image destination
Storing signatures
```

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>

3. Exportación e importación de la base de datos.

La bbdd de WordPress se exporta desde la máquina de desarrollo mediante mysqldump y se transfiere vía SCP a la máquina de producción. Allí se importa con el cliente mysql automáticamente.

El log del pipeline donde aparece:

```
+ ssh danielcfg@192.168.100.1 mysqldump -u wordpress -pwordpress wordpress
mysqldump: [Warning] Using a password on the command line interface can be insecure.
+ scp backup.sql danielcfg@192.168.100.2:/home/danielcfg2/
[Pipeline]
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 166 killed;
[ssh-agent] Stopped.
[Pipeline] // sshagent
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Desplegar en producción)
[Pipeline] sshagent
[ssh-agent] Using credentials danielcfg2
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-tEFSmm7ojq6C/agent.187
SSH_AGENT_PID=189
Running ssh-add (command line suppressed)
Identity added: /home/jenkins/agent/workspace/wordpress-ci-cd@tmp/private_key_15193354772810627941.key (jenkins-vps)
[ssh-agent] Started.
[Pipeline]
[Pipeline] sh
+ ssh danielcfg2@192.168.100.2 cd /home/danielcfg2/TFG-WordPress-K3s && git pull
Ya está actualizado.
[Pipeline] sh
+ ssh danielcfg2@192.168.100.2 mysql -u wordpress -pwordpress wordpress < ~/backup.sql
```

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>

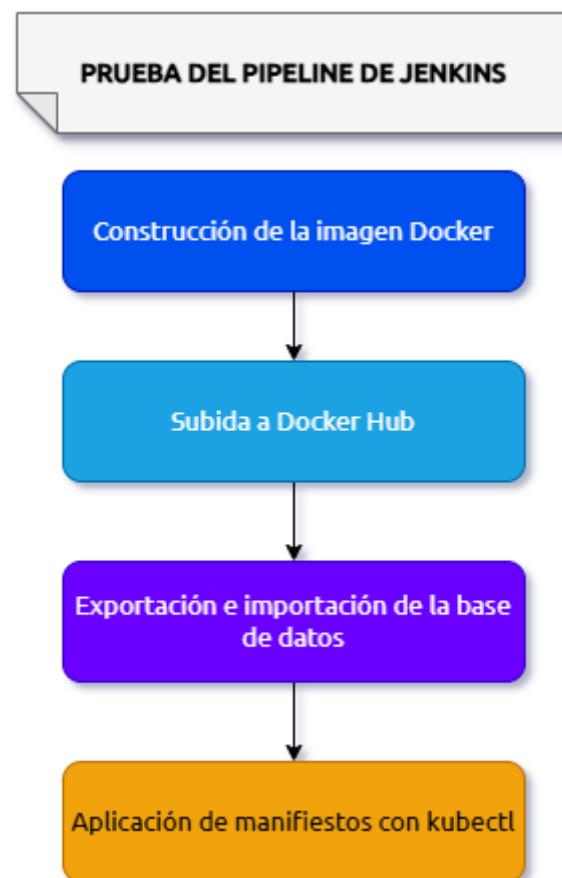
4. Aplicación de manifiestos con kubectl.

Los manifiestos de Kubernetes son aplicados en la máquina de producción para completar el despliegue.

El log del pipeline donde aparece:

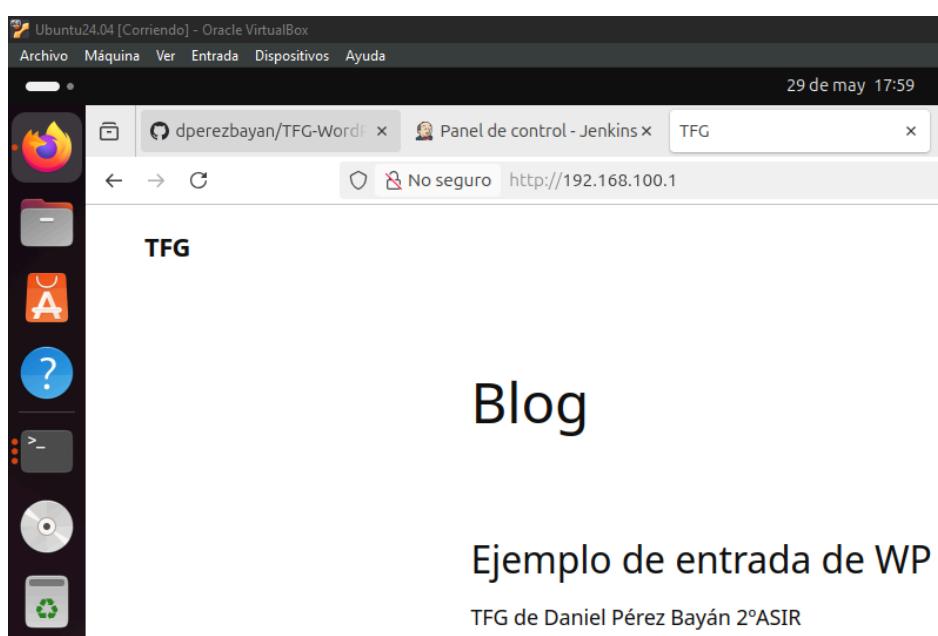
```
+ ssh danielcfg2@192.168.100.2 kubectl apply -f /home/danielcfg2/TFG-WordPress-K3s/k3s/ --kubeconfig=/etc/rancher/k3s/k3s.yaml
deployment.apps/mysql created
service/mysql-service created
persistentvolumeclaim/mysql-pvc created
persistentvolumeclaim/wordpress-pvc created
deployment.apps/wordpress created
service/wordpress created
configmap/wpdates created
```

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>



VERIFICACIÓN VISUAL DE WORDPRESS.

Desde la máquina de desarrollo, se accede en el navegador a <http://192.168.100.1>



Desde la máquina de producción, se accede en el navegador a <http://192.168.100.1>

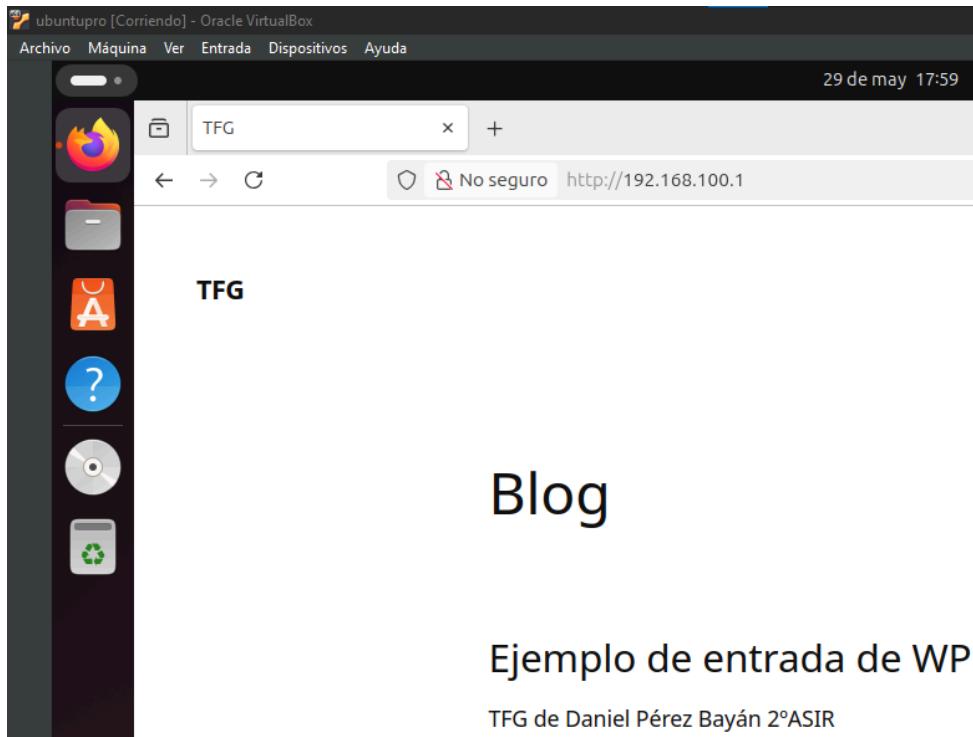


TABLA COMPARATIVA DEL PROCESO DE SINCRONIZACIÓN.

Fase	Entorno de desarrollo (192.168.100.1)	Jenkins (CI/CD)	Entorno de producción (192.168.100.2)
Modificación del contenido	Se añade una nueva entrada en WP desde el panel de administración	-	-
Commit en GitHub	Se hace git add, commit y push al repositorio remoto	Jenkins detecta el webhook (o se lanza manualmente el pipeline)	-
Construcción de la imagen	-	Se crea una nueva imagen Docker con Buildah usando el Dockerfile	-
Publicación en DockerHub	-	La imagen se etiqueta y se sube a DockerHub con credenciales seguras	-
Exportación de la base de datos	Se genera contenido real en Wordpress	mysqldump exporta la base de datos a backup.sql	-
Transferencia del dump	-	Se transfiere backup.sql mediante SSH al entorno de producción	-
Importación del dump	-	-	Se ejecuta mysql < backup.sql y se actualiza la base de datos
Despliegue en K3s	-	Se aplican los manifiestos YAML (kubectl apply -f)	Wordpress se despliega con la nueva base de datos

6. CONCLUSIONES.

Durante el proyecto he conseguido implementar un sistema de integración y despliegue continuo funcional. Utilizando herramientas como Jenkins, Buildah, GitHub y K3s, he logrado automatizar todo el proceso: desde los cambios en el código, hasta el despliegue de la aplicación WordPress en producción.

En resumen, he alcanzado los objetivos propuestos, aprendiendo a integrar diferentes tecnologías para desplegar aplicaciones de forma automática, clara y personalizada.

7. COSTE DESGLOSADO DEL PROYECTO.

INFRAESTRUCTURA Y HARDWARE.

- Portátil del trabajo. 400€ como coste estimado.
- Máquina de producción. 0€ ya que es una VM local.

RECURSOS EN LA NUBE.

- GitHub. Gratuito.
- DockerHub. Gratuito en versión básica.
- Ngrok. Gratuito con limitaciones (sin tener un dominio personalizado).

SOFTWARE Y HERRAMIENTAS.

- Sistema operativo (Ubuntu). Gratuito.
- Jenkins, Buildah, K3s. Código abierto y gratuito.

TIEMPO DE TRABAJO ESTIMADO.

Se han dedicado aproximadamente 100 horas al desarrollo del proyecto, incluyendo documentación, instalación, configuración, pruebas y redacción.

Valorando el coste/hora en 15 €, el coste estimado sería de 1.500 €, más el coste del portátil, el coste sería alrededor de 2000€.

Aunque durante el desarrollo del proyecto se han utilizado recursos gratuitos (como Jenkins, DockerHub, GitHub, etc.), en un entorno profesional estos componentes podrían tener un coste. Por ello, se realiza una estimación orientativa del coste si se desplegara en un entorno empresarial.

8. REFERENCIAS BIBLIOGRÁFICAS.

Jenkins documentación. Disponible en: <https://www.jenkins.io/doc/>

Kubernetes documentación. Disponible en: <https://kubernetes.io/docs/>

<https://www.youtube.com/watch?v=46tF3xSg-rq>

<https://www.youtube.com/watch?v=uupvnmWAx0g>

Docker Hub documentación (imágenes y repositorios). Disponible en:

<https://docs.docker.com/docker-hub/>

Buildah documentación. Disponible en: <https://buildah.io/>

<https://www.youtube.com/watch?v=23L69PyTh7A>

GitHub. <https://github.com/dperezbayan/TFG-WordPress-K3s>

Ubuntu documentación. Disponible en: <https://ubuntu.com/tutorials>

<https://www.youtube.com/watch?v=qH9JuCjCKow&t=296s>

Wordpress documentación. Disponible en: <https://wordpress.org/support/>



Helm documentación. Disponible en: <https://helm.sh/docs/>

Red Hat (CI/CD pipelines). Disponible en:

<https://www.redhat.com/en/topics/devops/what-cicd-pipeline>

<https://www.youtube.com/watch?v=oUTGnoQzciU>

<https://www.youtube.com/watch?v=88bnb9eTRNo>

Ngrok documentación. Disponible en: <https://ngrok.com/docs>

Plugins y Credenciales Jenkins. Disponible en: <https://plugins.jenkins.io/github/>

ChatGPT (OpenAI). Resolución de dudas y ayuda con la redacción del TFG.

<https://chat.openai.com>

Jenkins, GitHub Webhook

integración.<https://www.raulprietofernandez.net/blog/devops/como-crear-un-webhook-entre-github-y-jenkins>

<https://www.youtube.com/watch?v=CvJfD7bwHIQ>

Drawio . Para realizar un esquema visual:

<https://app.diagrams.net/>

Proyecto referencia documentación:

https://dit.gonzalonazareno.org/gestionar/proyectos/2024-25/ICDC_Jairo_Dominguez.pdf

<https://github.com/JairoDH/Buildah-k3s/tree/ba2cd9a118a82abb4c205160a3dfaef38f03e7bd2>

IES Gabriel García Márquez. Normativa del TFG de ASIR.

https://aulavirtual33.educa.madrid.org/ies.garciamarquez.leganes/pluginfile.php/55091/mod_resource/content/0/02_Detalles%20sobre%20la%20Memoria%20de%20Proyecto.pdf

https://aulavirtual33.educa.madrid.org/ies.garciamarquez.leganes/pluginfile.php/11902/mod_resource/content/0/Criterios%20de%20calificaci%C3%B3n%20del%20m%C3%B3dulo%20de%20Proyecto.pdf

9. ANEXOS.

- Jenkinsfile completo

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/Jenkinsfile>

- Capturas del pipeline ejecutado

Incluidas en el apartado 5. Resultados y Discusión.

[Prueba del pipeline Jenkins:](#)

- Archivos YAML del clúster Kubernetes (K3s)

Se muestra y explica en el apartados 4.2.6

[4.2.6 Kubectl y K3s.](#)

<https://github.com/dperezbayan/TFG-WordPress-K3s/tree/main/k3s>

- Credenciales utilizadas

Documentadas en el apartado 4.2.8

[4.2.8 INSTALACIÓN Y CONFIGURACIÓN DE Jenkins EN K3s](#)

[MEDIANTE Helm.](#)

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description	
 github-credentials	dperezbayan/******** (Acceso al repositorio de GitHub desde Jenkins)	Username with password	Acceso al repositorio de GitHub desde Jenkins	
 dockerhub-credentials	dperezbayan/******** (Acceso a DockerHub desde Jenkins)	Username with password	Acceso a DockerHub desde Jenkins	
 vps_SSH	danieltfg2 (Clave SSH para conectar con la máquina de producción)	SSH Username with private key	Clave SSH para conectar con la máquina de producción	

- Resultado completo del pipeline exitoso

Ver pipeline-exitoso.txt

<https://github.com/dperezbayan/TFG-WordPress-K3s/blob/main/pipeline-exitoso.txt>