Integración y despliegue continuo

integrando

Buildah en Jenkins, K3s y Git

2º ASIR

Jairo Domínguez Hidalgo **IES Gonzalo Nazareno**

Índice

1. Objetivos a conseguir

1.1 Objetivos iniciales

- Desplegar una aplicación (*Wordpress)* en un clúster de ***Kubernetes*** utilizando ***k3s***.
- Implementar la integración y el despliegue continuo con ***Jenkins*** para automatizar los despliegues.
- Crear, mantener y alojar imágenes de contenedores con ***Buildah*** y ***Docker Hub***.
- Controlar el código y configuraciones con ***Git***.

1.2 Objetivos conseguidos

- Automatización completa del despliegue mediante *Jenkins* y *Git*.
- Flujo continuo de integración y despliegue.
- Creación de imágenes con ***Buildah*** en *Jenkins*.

2. Descripción del escenario

2.1 Parte hardware del escenario

- Para la parte desarrollo se ha empleado una máquina local con ***Ubuntu 24.04*** con las siguientes características:

  - ➢ 8 unidades de CPU
  - ➢ 16GB de memoria RAM
  - ➢ 50GB de almacenamiento

- Para producción se ha empleado una máquina virtual en la nube ***(VPS)*** con sistema ***Debian 12*** y las siguientes características:

  - ➢ 2 vCore de CPU
  - ➢ 4GB de memoria RAM
  - ➢ 120GB de almacenamiento

2.2 Parte software del escenario

- **Clúster de Kubernetes (k3s)**

  Implementación de un clúster ***Kubernetes ligero*** usando k3s para un menor consumo de recursos. Este clúster se ha utilizado para orquestar los contenedores que alojan la aplicación y sus dependencias como la base de datos ***MySQL***.

  Además se han empleado herramientas para la gestión de paquetes como ***Helm*** y ***Kubectl*** para la línea de comandos que ejecuta órdenes para crear, monitorear y gestionar los recursos del clúster.

- **Creación y gestión de imágenes**
  - ➢ **Buildah:** Utilizado para crear y gestionar imágenes de contenedores, la imagen de *Wordpress* ha sido creada a partir de un *Dockerfile* personalizado.
  - ➢ **DockerHub:** Utilizado para almacenar y compartir las imágenes generadas.

- **Gestión de código con Git y GitHub**

  Con ***Git*** se gestiona todo el código fuente y archivos de configuración en un repositorio en ***GitHub***, además de automatizar con cada ***push*** de la rama ***main*** el despliegue de la aplicación.

- **Integración y despliegue continuo (CI/CD)**
  - ➢ **Jenkins:** se utiliza como herramienta principal para la integración y el despliegue continuo gestionando el proceso de construcción y prueba del código.

3. Fundamentos teóricos y conceptos

En este apartado se explican los conceptos y fundamentos teóricos necesarios para comprender y entender cómo funciona la aplicación empleando los siguientes programas y servicios.

3.1 Contenedores y Orquestación

3.1.1 Contenedores

Cada parte de la aplicación (*wordpress* y *Mysql)* se ejecuta en contenedores diferentes para proporcionar aislamiento (*en caso de que si uno falla no afecte directamente al otro*) y portabilidad, encapsulando todo lo necesario para ejecutar la aplicación para facilitar la migración entre diferentes servidores o entornos.

3.1.2 Kubernetes y k3s

Kubernetes es una plataforma de **orquestación** que gestiona el ciclo de vida de los contenedores, encargándose del escalado, la disponibilidad y el mantenimiento de las aplicaciones distribuidas en los contenedores.

Para este proyecto se ha elegido ***k3s*** , una versión más ligera y optimizada de ***kubernetes***, pudiendo así ejecutar un orquestador de contenedores usando pocos recursos.

3.2 Creación y gestión de imágenes

Para que una aplicación se despliegue usando un contenedor, es necesario crear una fichero de texto con instrucciones llamado ***Dockerfile***.

Para la creación de este fichero se ha usado la herramienta ***Buildah*** y para almacenar y compartir la imagen ***DockerHub.***

3.2.1 Buildah

Es una herramienta de línea de comandos para crear, modificar y gestionar contenedores e imágenes de contenedores. Se ha usado esta herramienta porque permite crear imágenes sin requerir de un demonio en ejecución en segundo plano.

Con esto se consigue más flexibilidad y seguridad, ya que no depende de un servicio adicional activo y reduce la superficie de ataque al no tener un demonio de alto privilegio constantemente en ejecución, reduciendo las vulnerabilidades en el sistema.

3.2.2 Docker Hub

Este servicio nos permite almacenar y compartir las imágenes que creemos con ***Buildah***, pudiendo así distribuir las imágenes necesarias entre los diferentes tipos de entornos.

3.3 Control y gestión del código fuente

Para llevar un control de versiones del código fuente de la aplicación es necesario emplear la herramienta ***Git***, conjunto con el servicio ***GitHub***, donde se almacena los ficheros necesarios para el despliegue de la aplicación, y poder implementar la integración y desarrollo continuo.

3.3.1 Git

Es un software de control de versiones que permite gestionar el código fuente de la aplicación de manera eficiente, haciendo un seguimiento de los cambios, estos cambios se guardan en el repositorio creado en ***GitHub***.

3.3.2 GitHub

Es una plataforma en línea que aloja repositorios de ***Git,*** proporcionando herramientas para compartir, revisar y colaborar en proyectos.

GitHub sirve como repositorio centralizado para la integración continua de la aplicación.

3.4 Integración y Despliegue continuo

Para automatizar todo el proceso de desarrollo y despliegue de la aplicación mejorando la eficiencia y reduciendo errores se emplea la integración y despliegue continuo (CI/CD).

En la fase de integración continua, se realizan pruebas automáticas a medida que se integran cambios de código en el repositorio.
En la fase de despliegue continuo, se automatizan las actualizaciones y la publicación de nuevas versiones en producción.

Con esto garantiza que cada cambio de código se pruebe y despliegue de manera fácil y fiable, manteniendo la estabilidad y calidad del servicio.

3.4.1 Jenkins

Jenkins es una herramienta que permite automatizar y optimizar el flujo de trabajo en desarrollo y despliegue de la aplicación.

El objetivo principal de **Jenkins** es la integración continua (**CI**), que consiste en gestionar el proceso de construcción y prueba del código, garantizando que cualquier cambio realizado en el repositorio sea validado.

4. Descripción detallada del proceso.

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 Preparación del entorno de desarrollo.

Para preparar el escenario del entorno de desarrollo se ha empleado un PC con el sistema operativo **Ubuntu 24.04 LTS Noble Numbat** y las siguientes características:

- ★ CPU : 8 núcleos
- ★ RAM : 16GB
- ★ Almacenamiento: 50GB

4.1.1 Instalación de herramientas, Buildah y k3s.

Una vez esté instalada y configurada la máquina local, se procede a la actualización del sistema, descarga e instalación de paquetes y herramientas necesarias.
Instalación de herramientas, creador de imagen ***Buildah*** y ***kubernetes*** (k3s) :

```
sudo apt update

sudo apt install curl git buildah mariadb-client

curl -sfL https://get.k3s.io | sh -
```

Una vez esté la máquina local actualizada y con las herramientas necesarias, es hora de crear el directorio de trabajo que 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, en este caso ***WordPress.***4.1.2 Dockerfile.

Tras la creación y sincronización del nuevo directorio con ***GitHub***, toca configurar el fichero ***Dockerfile***, este fichero es un archivo de texto que contiene un conjunto de instrucciones para crear una imagen.
Esta imagen define los pasos necesarios para construir la imagen que va a necesitar el contenedor en el despliegue del clúster con la aplicación ***WordPress***, como la base de la imagen, en este caso se ha elegido ***debian***, las dependencias que se deben instalar, configuraciones del entorno, archivos que se deben copiar, etc ..
Dockerfile

4.1.3 Creación de imagen con Buildah y DockerHub.

Después de configurar el fichero ***Dockerfile***, con la herramienta ***Buildah*** se crea la imagen, desde el mismo directorio donde se encuentra el fichero ***Dockerfile*** se emplea el siguiente comando.

```
buildah bud -t wpimagen .
```

A continuación, para que el contenedor pueda usar la imagen creada, hay que subir la imagen a la plataforma ***DockerHub***, para ello hay que identificarse con la cuenta personal y subir la imagen.

```
$ buildah login -u <user> -p docker.io

$ buildah push <nameimge> docker.io/<user>/wpimagen
Getting image source signatures
Copying blob 1796058040b7 done   |
Copying blob b2b31b28ee3c skipped: already exists
Copying config ae1040598c done   |
Writing manifest to image destination
```

4.1.4 Kubectl y K3s.

Luego, para el despliegue del clúster en **k3s**, es necesario configurar ***kubectl***, una herramienta de línea de comandos para gestionar el clúster de ***kubernetes***.

Para ello, hay que darle permisos al fichero de configuración que crea al instalar ***kubernetes (k3s en este caso)*** a nuestro usuario, este fichero se crea en el directorio específico.

```
$ sudo chown -R user:user /etc/rancher/k3s/k3s.yaml
```

Con esto podremos gestionar el clúster de ***kubernetes.***

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

El contenido de esta carpeta consta de los siguientes ficheros:

```
$ tree k3s/
k3s-dev/
├── mysql-deployment.yaml
├── mysql-srv.yaml
├── pvc-mysql.yaml
├── pvc-wordpress.yaml
├── wordPress-deployment.yaml
├── wordPress-srv.yaml
├── wpdates_configmap.yaml
└── wppass_secret.yaml
```

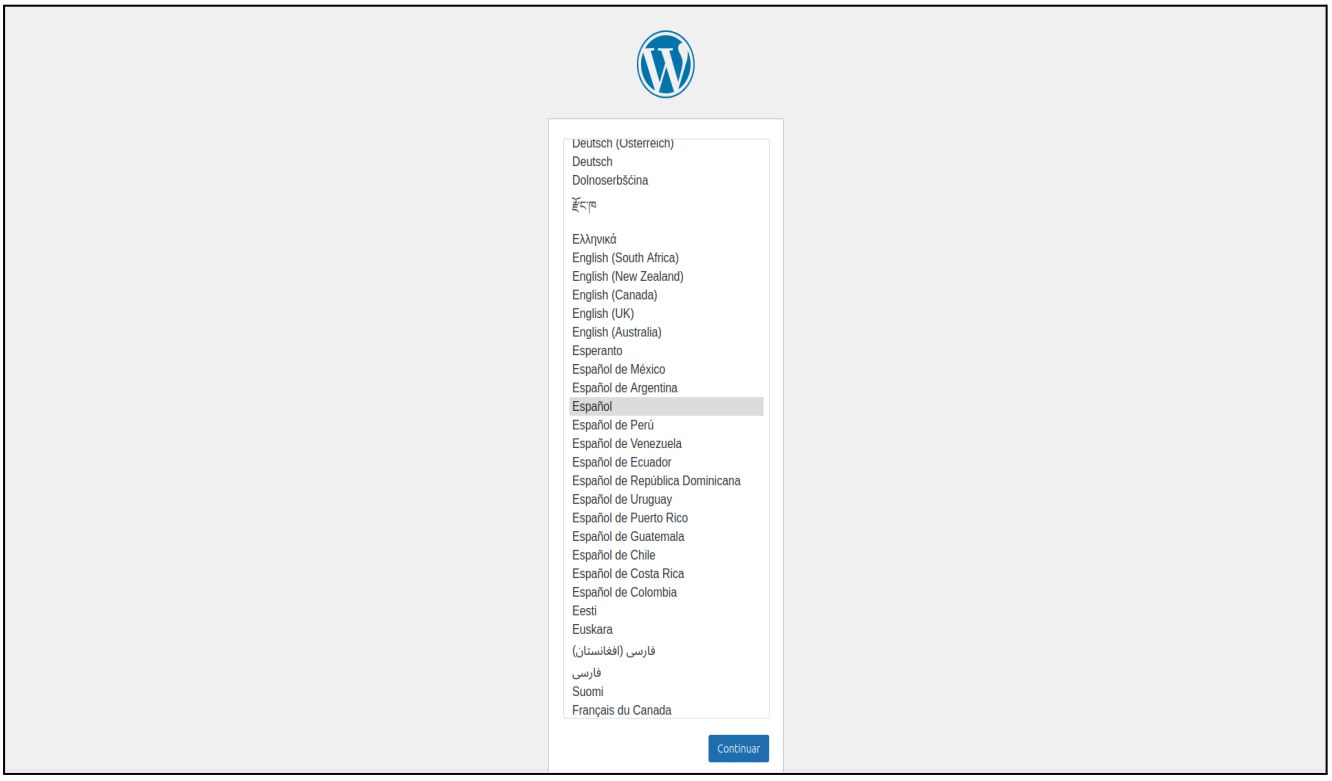
Para ver el contenido de los ficheros, Github.

Después de configurar cada uno de los ficheros de configuración del clúster, es hora de desplegar la aplicación.

```
$ kubectl apply -f k3s/ -f ingressdev.yaml

ingress.networking.k8s.io/wordpress-ingressdev created
deployment.apps/mysql created
service/mysql created
persistentvolumeclaim/mysql-pvc created
persistentvolume/wordpress created
deployment.apps/wordpress created
service/wordpress created
configmap/wpdates created
secret/wppass created
```

* Para el fichero ***ingressdev.yaml*** se ha creado fuera del directorio ***k3s*** para la gestión en el pipeline de ***Jenkins,*** más adelante explicaré este proceso.



4.1.5 Aplicar Integración Continua y Despliegue Continuo (IC/DC)

Una vez comprobado que la aplicación se despliega en desarrollo, para automatizar futuras actualizaciones o cambios en el código fuente y reducir el riesgo de introducir errores en la aplicación se aplica la integración y despliegue continuo.

Para este proceso se ha usado ***Jenkins,*** una herramienta de automatización de código abierto que permite a los desarrolladores automatizar una serie de procesos dentro del ciclo de vida del desarrollo de software, como la construcción, prueba y despliegue de aplicaciones.

Jenkins ayuda a que el software se construya y se implemente de manera eficiente, detectando errores en las primeras etapas del proceso de desarrollo.

Las principales funcionalidades que incluye *Jenkins*:

- **Integración Continua (CI):** Automatiza el proceso de integración del código, cada vez que se envíen cambios al repositorio a través de ***Git,*** *Jenkins* ejecuta automáticamente pruebas de que el código no rompa la aplicación.
- **Despliegue Continuo (CD):** Además de la integración, *Jenkins* automatiza el proceso de despliegue de la aplicación en la máquina de producción, reduciendo el tiempo y los errores en el despliegue manual, este proceso se realiza a través de ***pipelines***.
- **Automatización de tareas:** Denominado ***Pipelines***, una secuencia de tareas automatizadas. Este pipeline incluye pasos como la construcción de la imagen del Dockerfile, aplicación de variables de entorno para jenkins, clonación de repositorios, utilización de contenedores temporales, etc…

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

Para la instalación de *helm*:

```
curl -fsSL
https://raw.githubusercontent.com/helm/helm/main/scripts/get-
helm-3 | bash
```

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

```
helm repo add jenkins https://charts.jenkins.io
helm repo update
"jenkins" already exists with the same configuration,
skipping
Hang tight while we grab the latest from your chart
repositories...
...Successfully got an update from the "jenkins" chart
repository
...Successfully got an update from the "stable" chart
repository
Update Complete. ⎈Happy Helming!⎈
```

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

```
kubectl create namespace jenkins
```

Con el *namespace* creado, se instala *Jenkins* desde los repositorios de *helm*:

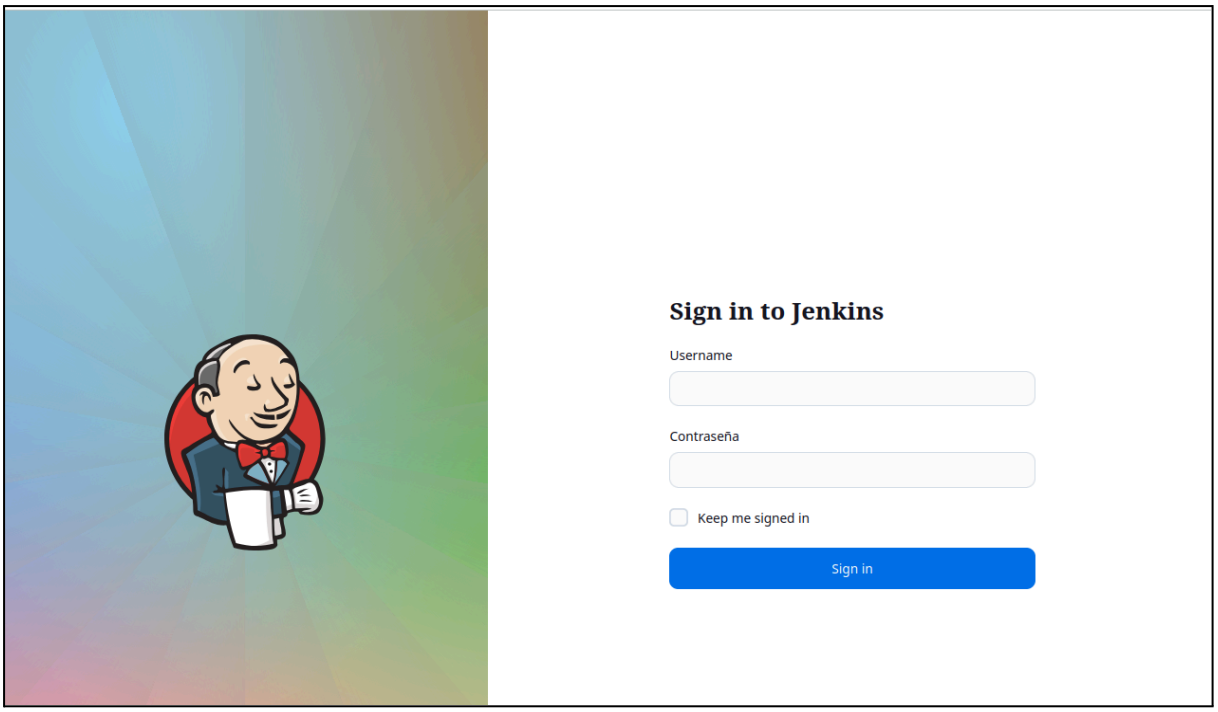
```
helm install jenkins jenkins/jenkins -n jenkins
```

Comprobar instalación:

```
kubectl get all -n jenkins
```

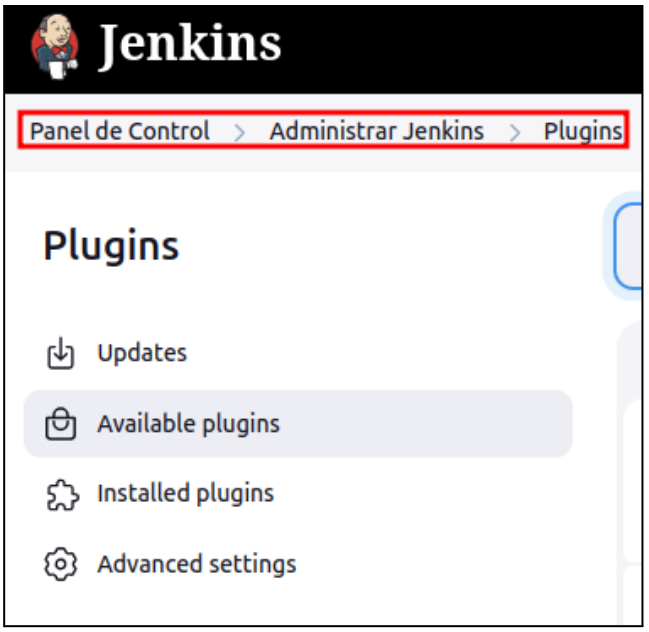
Para poder acceder a *Jenkins*, la misma aplicación, tras la instalación te ofrece el comando para obtener la contraseña:

```
jsonpath="{.data.jenkins-admin-password}"
secret=$(kubectl get secret -n jenkins jenkins -o
jsonpath=$jsonpath)
echo $(echo $secret | base64 --decode)
```



Una vez está desplegado *Jenkins*, antes de configurar el *pipeline*, es necesario instalar algunos *plugins* para aplicar los comandos de las herramientas empleadas en la integración.

Para instalar estos *plugins* 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 he tenido que instalar los siguientes:

- ➢ Kubernetes plugin
- ➢ Kubernetes CLI Plugin
- ➢ GitHub plugin
- ➢ Git plugin
- ➢ GitHub Integrations
- ➢ SSH Agent Plugin
- ➢ Pipeline Plugin

A continuación, *Jenkins* necesita credenciales para la automatización de la integración y despliegue continuo, hay diferentes opciones para la creación de credenciales, 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.
Es de tipo usuario y contraseña, por lo que se crea con el usuario y la contraseña de la cuenta de *GitHub*, con el propósito de autenticar la descarga de código fuente o enviar cambios además de permitir la configuración del ***webhook*** para la ejecución de *pipelines* cuando se actualiza el repositorio.

- **DOCKER_HUB**:

Al igual que el anterior, es de usuario y contraseña con lo que proporcionamos los datos de *Docker Hub* para permitir subir o descargar imágenes *Docker*.

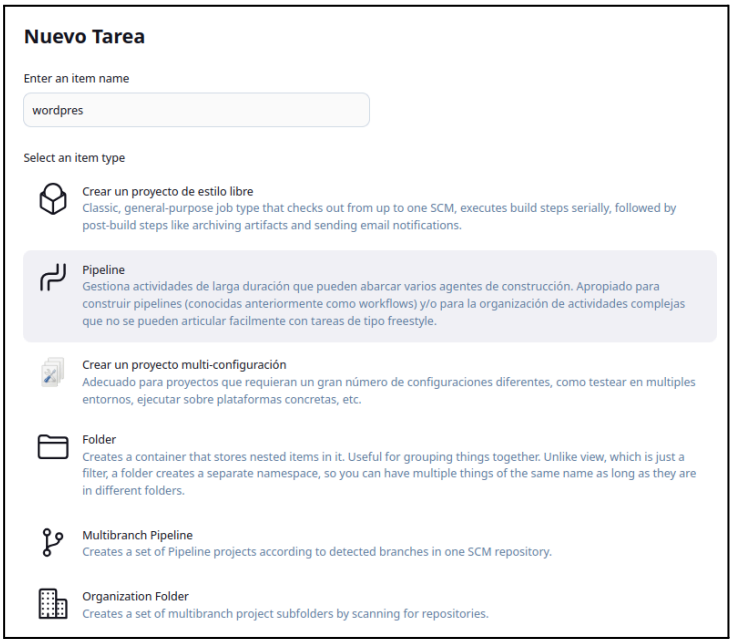
- **VPS_SSH**:

Con esta permitimos a *Jenkins* poder conectarse por *SSH* a la máquina de producción (*VPS*) 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.

- **LOCAL_SSH**:

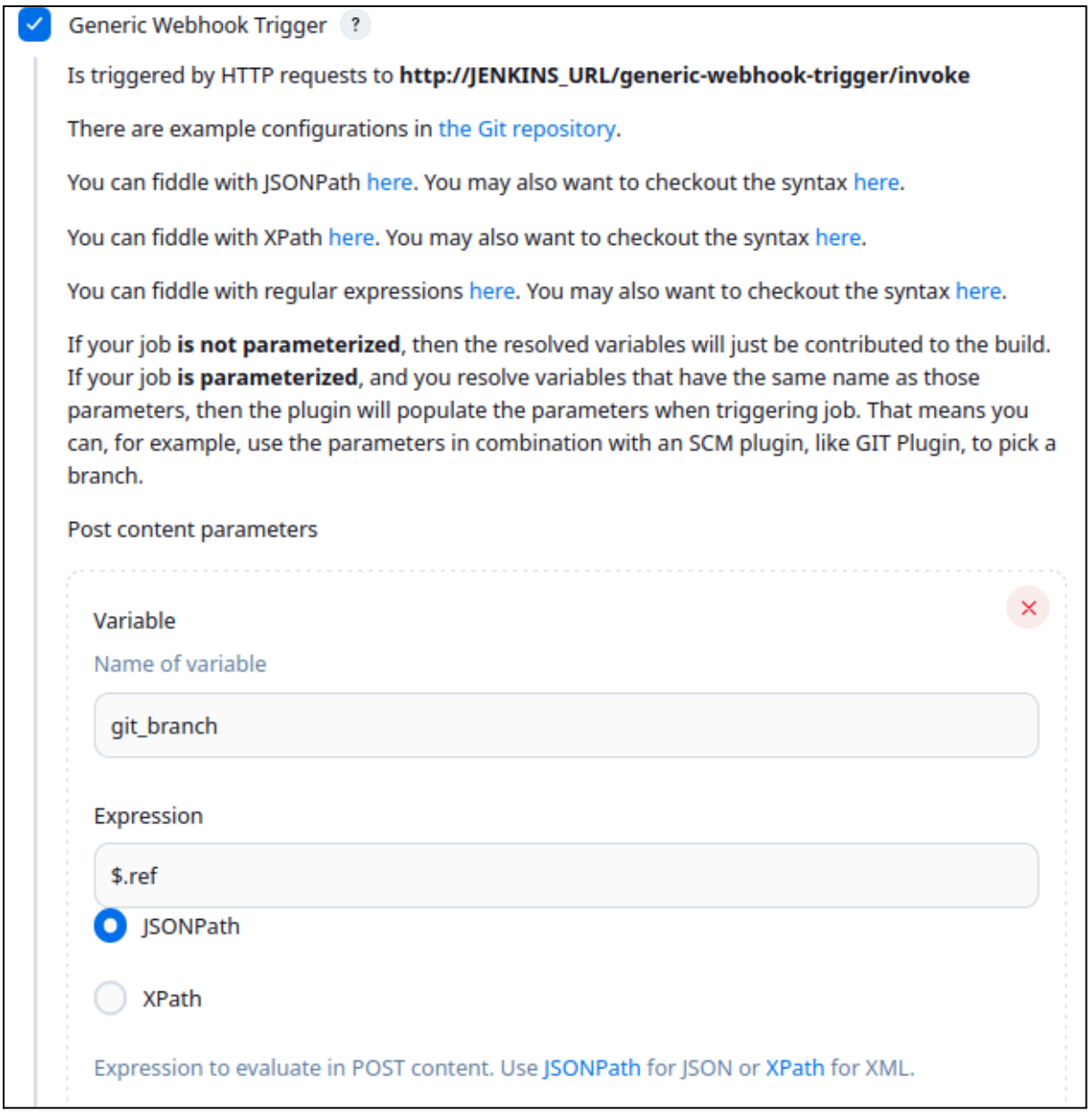
Al igual que el anterior, credencial de tipo SSH, pero esta vez se ha usado la clave privada de mi host para poder permitir la conexión a la máquina del entorno de desarrollo y poder realizar la migración del fichero *.sql* entre ambos entornos.

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



Y se configura la nueva tarea…

Al instalar los *plugins* permite añadir configuraciones al *pipeline*, con ***Generic Webhook Trigger*** podemos crear variables de entorno:



Se le está indicando al *pipelines* la variable ***git_branch*** que obtenga el valor del campo ***ref*** del contenido del ***JSON*** que llega en el POST request del *webhook*.
Este campo contiene la rama desde la cual se ha realizado el ***push***.

En el apartado ***pipeline***, le indicamos que el *pipeline* se encuentra en el repositorio de *GitHub* ***(Pipeline script from SCM)***, además de indicarle el nombre de las dos ramas posibles para ejecutar que contiene dicho repositorio, aunque para este proyecto solo se ejecutará la rama ***main***.

Con esto, habremos configurado la nueva tarea en *Jenkins*, ahora procederemos a la configuración del fichero ***Jenkinsfile*** que contiene el proceso de la integración y despliegue continuo.

Para poder usar la herramienta ***Buildah*** en el pipeline para poder crear y subir la imagen, se ha desarrollado una nueva imagen *Docker* partiendo de la imagen base que proporciona *Jenkins* en *Docker Hub* y agregando al código de la paquetería *Buildah,* esta imagen se ha llamado ***Buildah:v2***.
Contenido del fichero: ***Dockerfile***

Una vez creada y subida la imagen a *Docker Hub*, se configura el fichero *Jenkinsfile*.
Este *pipeline* se desarrolla de la siguiente forma:

Se le indica las variables globales reutilizables al comienzo, seguidamente define un agente que crea un *pod* temporal personalizado en ***kubernetes,*** el *pod* contiene la imagen *Buildah* desarrollada anteriormente.

Luego, se le añade opciones, indicando que mantenga los últimos 10 ***builds*** para ahorrar espacio, además de indicarle que guarde el estado del *pipeline* con menor frecuencia para mejorar la velocidad del *pipeline* y desactiva la ejecución concurrente, evitando que se ejecuten dos *pipelines* a la vez.

Después, realiza una verificación de la rama que se ejecuta el ***push*** en la máquina de desarrollo y realiza una clonación del repositorio en caso de que la verificación haya sido validada, seguidamente pasa a la creación y subida de la imagen *Docker* a través del contenedor temporal creado al principio, que contiene la herramienta *Buildah*.

A continuación, con las credenciales creadas anteriormente para la conexión SSH para la máquina de desarrollo, se ejecuta el *script* para la exportación del contenido de la base de datos y se transfiere a la máquina del entorno de producción.
Y por último, si todas las ejecuciones se han confirmado, pasa al despliegue en el entorno de producción haciendo uso de las credenciales *SSH* permitiendo la conexión a la máquina VPS y ejecutando una cadena de comandos finalizando por la importación de la base de datos y el despliegue del clúster en producción.

4.1.6 Ngrok y el webhook

Con la configuración del *pipeline* hecha, para que el servicio de *Jenkins* pueda ejecutar el fichero *Jenkinsfile* y así poder automatizar los cambios que se suben al repositorio ***main*** en *GitHub*, se debe configurar un ***webhook*.**

Un *webhook* es una URL que actúa como un punto de comunicación entre aplicaciones, para este proyecto se ha empleado para que *GitHub* envíe información al *clúster* donde se encuentra el servicio de *Jenkins* por cada evento (*push)* que se realice desde la rama *main*.
Como el servicio de *Jenkins* se encuentra dentro de una red privada, se hace uso de la herramienta ***ngrok***, que permite exponer aplicaciones locales al exterior de forma segura.
Para la configuración de *ngrok,* desde la terminal del entorno de desarrollo, se emplea el comando indicando la IP de la red y el puerto donde se encuentra el servicio de *Jenkins*.

```
$ ngrok http ip:puerto
```

Con la dirección URL que nos proporciona *ngrok,* hay que aplicarlo en la configuración del *webhook* del repositorio en *GitHub* permitiendo la conexión entre los servicios.

4.1.6 Certificado SSL y configuración TLS en k3s.

Se ha empleado para este proyecto el uso de certificado ***SSL*** (Secure Sockets Layer) y ***TLS*** (Tansport Layer Security) son tecnologías que cifran la comunicación entre dos sistemas, como el navegador web y un servidor.
TLS es la versión más segura y moderna de SSL, con esto permitimos que la comunicación entre usuarios y la aplicación emplee el protocolo ***HTTPS***.

Un certificado *SSL/TLS* es un archivo que valida la identidad del dominio, en este caso, ***touristmap.es***, al contratar el dominio por una empresa, me facilitan los certificados necesarios para crear el certificado.

Los certificados que me han proporcionado son:

- **Certificado del servidor**: identifica el dominio *touristmap.es*
- **Clave privada:** se usa en el servidor para descifrar y firmar los datos
- **Certificados intermedios (CA):** este certificado verifica la autenticidad del certificado mediante una cadena de confianza hacia la Autoridad Certificadora de *IONOS.*

Una vez obtenidos los certificados, se realiza la configuración en el *clúster k3s* del entorno de producción.
Para configurar *TLS* se ha creado un directorio donde se guardará los certificados en el entorno de producción.

Para combinar el certificado y los intermedios (claves privadas):

```
$ cat <dominio.es>_ssl_certificate.cer intermediate1.
cer intermediates.cer > fullchaim.pem
```

Para crear un *Secret TLS* en *K3s* de producción:

```
$ sudo kubectl create secret tls tls-cert-secret
--cert=/etc/ssl/certs/<dominio.es>.crt   /
--key=/etc/ssl/private/<dominio>.es.key  /
--namespace default
```

En este proyecto se ha utilizado dos entornos, desarrollo y producción, para el entorno de desarrollo como se va a acceder localmente no es necesario aplicar el *TLS* creado anteriormente, por lo tanto se han creado dos ficheros *ingress:*

- Fichero Ingress para el entorno de desarrollo: Desarrollo
- Fichero Ingress para el entorno de producción: Producción

También se ha creado un fichero ingress para acceder al servicio de Jenkins:

- Fichero Ingress para Jenkins: Jenkins

4.2 Preparación del entorno de producción.

En este entorno se han realizado las siguientes configuraciones para concluir con el proyecto.
Se ha alquilado un servidor (VPS) con un sistema operativo Debian 12 Bookworm y las siguientes características:

- ★ CPU: 2 vCore
- ★ RAM: 4GB
- ★ Almacenamiento: 120 GB

4.2.1 Instalación de herramientas y K3s.

Para el entorno de desarrollo se ha actualizado y aplicado la instalación y configuración de ***K3s***, igual que en el entorno de desarrollo, la herramienta ***Git*** y ***mysql-client***.

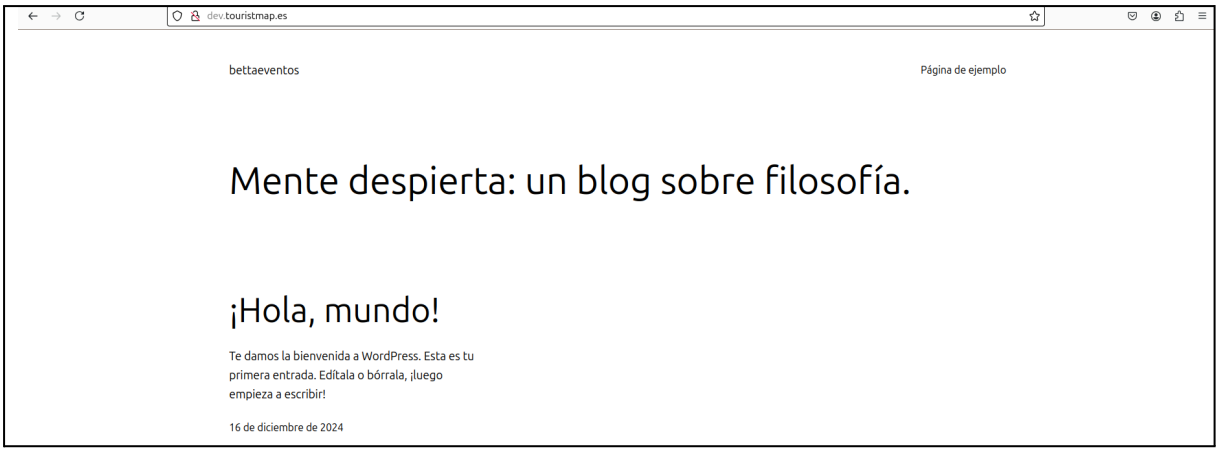
Se ha creado un directorio para que se monte el volumen de MySQL al realizar el despliegue del clúster, un usuario con privilegios y el par de claves que se configuró con Jenkins para permitir las conexiones SSH.

A continuación, se ha clonado el repositorio del proyecto de GitHub.

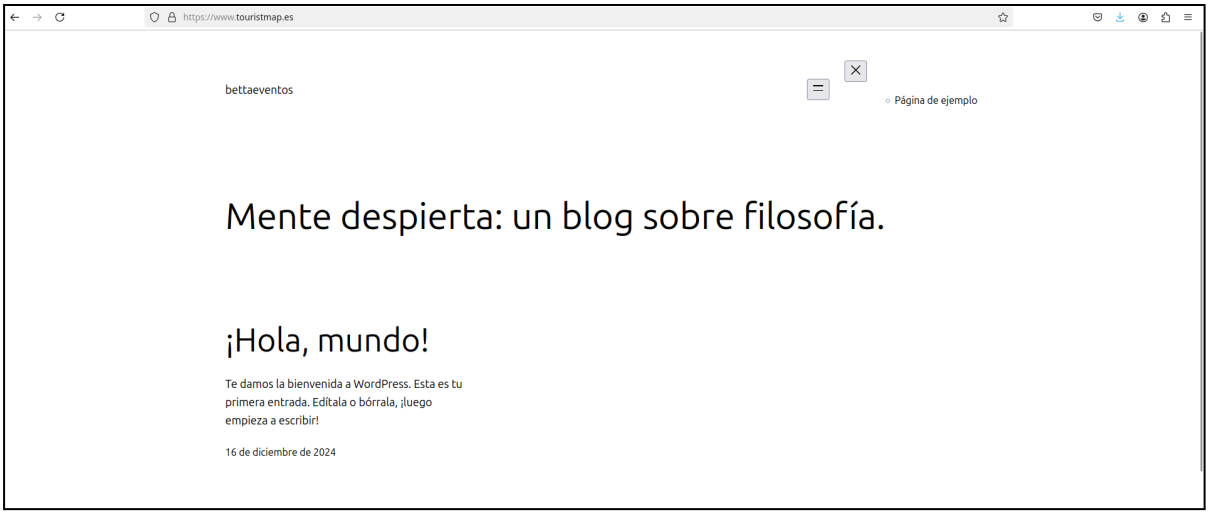
```
$ ls Buildah-k3s/
Dockerfile         Jenkinsfile  ScriptImportBackup.sh
ingressdev.yaml    k3s-dev      scriptbackup.sh
Jenkingress.yaml   README.md     buildah
ingressprod.yaml   wordpress
```

Con los certificados de SSL explicado anteriormente, se ha configurado el TLS en el clúster para emplearlo en el ingress de producción.
Con estos pasos y una configuración adecuada queda aplicada la integración y despliegue continuo entre los entornos de desarrollo y producción.

Wordpress desplegado en el entorno de desarrollo:



Wordpress desplegado en el entorno de producción:



5. Dificultades que se han encontrado.

Durante la implementación del flujo de integración y despliegue continuo (IC/DC) basado en Jenkins, Buildah, Git y Kubernetes, me he encontrado con varias dificultades que requirieron ajustes y soluciones específicas:

Para el uso de Buildah en el script de Jenkins, tuve que crear una nueva imagen basada en la que utiliza el contenedor en mi clúster para la ejecución de Jenkins, una vez localizada la imagen, desarrollarla, añadiendo herramientas para el uso de Buildah dentro del Pipeline, asignada a un contenedor temporal dentro del script.

La configuración del pipeline para gestionar los diferentes entornos, al igual que el desarrollo de scripts para la importación y exportación de la base de datos de desarrollo a producción.

6. Conclusiones.

Durante el desarrollo de este proyecto, uno de los objetivos principales fue implementar un flujo robusto de integración y despliegue continuo (IC/DC) que garantizara la coherencia entre los entornos de desarrollo y producción. Para ello, opté por una solución basada en Jenkins, Buildah y Git, que me proporcionó una estructura flexible y eficiente para gestionar el ciclo de vida de la aplicación WordPress.

El proceso comienza con el repositorio de código fuente alojado en Git, que contiene tanto el código de la aplicación como los archivos de configuración necesarios para Kubernetes (deployment, service e ingress). Cada vez que se realiza un cambio en el repositorio, Jenkins detecta automáticamente las actualizaciones mediante webhooks, lo que desencadena un pipeline.

Una de las decisiones clave ha sido utilizar Buildah como herramienta principal para la construcción de imágenes de contenedor. Buildah ofrece una solución ligera y segura que elimina la necesidad de un demonio como Docker, permitiendo generar imágenes directamente dentro de los contenedores de Jenkins.
El pipeline de Jenkins incluye un paso que emplea Buildah para construir una nueva imagen basada en el código actualizado y los archivos Dockerfile. La imagen resultante se etiqueta adecuadamente y se almacena en un registro de contenedores accesible desde los entornos de desarrollo y producción.

En conclusión, la combinación de Buildah, Jenkins y Git ha sido una solución ideal para implementar un flujo de IC/DC eficiente y confiable. Este enfoque simplifica la gestión de los despliegues y asegura que el proyecto cumple con los objetivos.

7. Bibliografía.

- Documentación de Kubernetes: https://k3s.io/
- Documentación de Jenkins: https://www.jenkins.io/
- Documentación de Buildah: https://buildah.io/
- Documentación y repositorios de helm: https://helm.sh/
- Documentación de ngrok: https://ngrok.com/
- Documentación de IONOS para el SSL: www.ionos.es
- Videos de YouTube.
- Apuntes de clase.