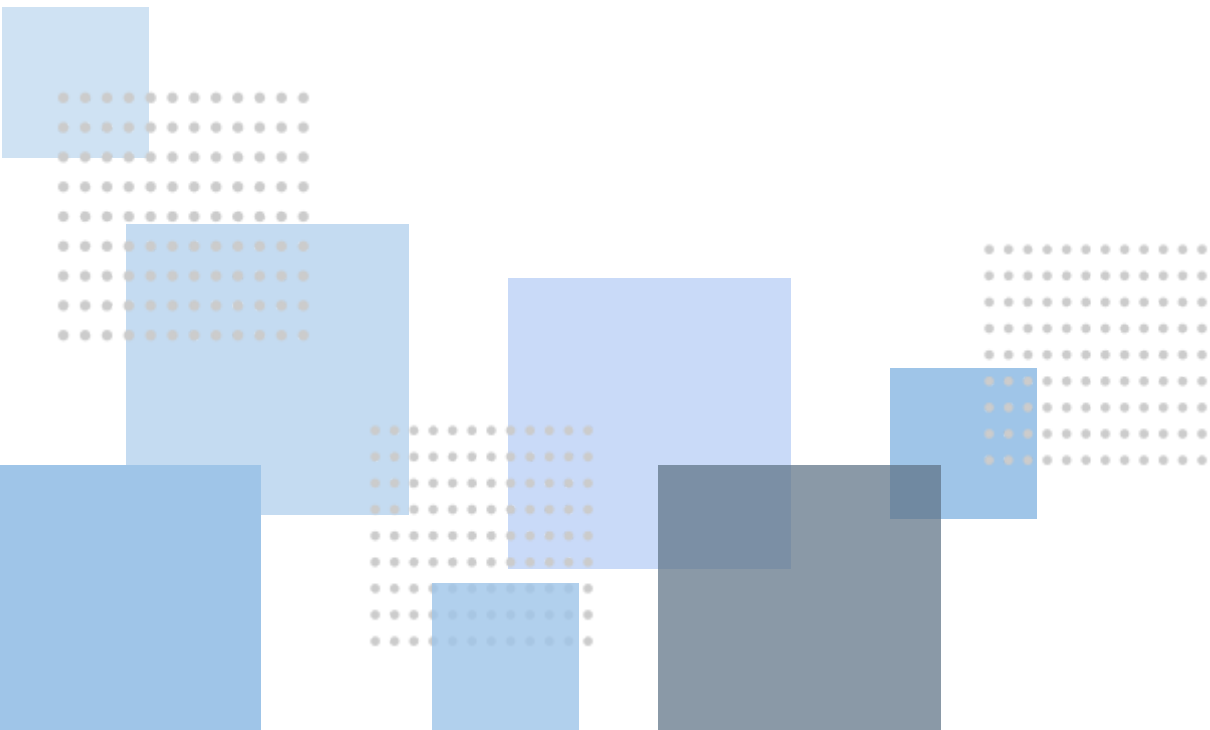


Implantación de Entornos Dinámicos

Despliegue automático de entornos usando Jenkins y Kubernetes
como herramientas principales

Belén Nazareth Durán Meléndez
2º ASIR



- ÍNDICE -

1. Objetivos	4
2. Escenario	5
2.1 Descripción	5
3. Fundamentos teóricos y conceptos	8
3.1 Software	8
3.1.1 Jenkins	8
3.1.2 Docker	9
3.1.3 Kubeadm	10
3.1.4 Git	11
3.1.5 Helm	11
3.1.6 K8s Lens	11
4. Descripción	13
4.1 Demostración del escenario	13
4.1.2 Pasos a seguir	13
4.2 Instalaciones y configuraciones	20
4.2.1 Containerd	20
4.2.2 Docker	23
4.2.3 Helm	24
4.2.4 Kubernetes	26
4.2.5 Configuración de Jenkins y Desarrollo de la Aplicación	30
5. Conclusión	37
6. Referencias	38
6.1 Recursos	38
6.1.1 Aplicación	38
6.1.2 Librería compartida (Shared Library)	38
6.1.3 Presentación	38
6.2 Bibliografía	39

“Todo empieza por un commit”

1. Objetivos

Opté por este proyecto ya que ofrece una manera sencilla de crear entornos de prueba dando libertad de verificar visualmente si los nuevos cambios en la aplicación se ejecutan con éxito entre otros objetivos, como pueden ser:

- Agilizar el desarrollo y las pruebas, permitiendo a los desarrolladores trabajar en ramas independientes sin afectar la rama principal y desplegar entornos de trabajo dinámicos en consecuencia.
- Utilizar el entorno dinámico para realizar pruebas visuales y verificar de manera efectiva que los cambios en la aplicación se ejecuten sin problemas y sean visualmente coherentes.
- Aprender a trabajar con este tipo de entornos, tanto creándolos como administrándolos.

2. Escenario

2.1 Descripción

Un entorno dinámico se puede entender como un entorno de trabajo en el que las aplicaciones, sistemas o infraestructuras pueden adaptarse y cambiar de manera ágil y automatizada. Esto puede incluir la capacidad de escalar automáticamente, gestionar recursos de manera eficiente, implementar actualizaciones sin problemas y responder a cambios en la demanda.

En este caso, se usará una aplicación almacenada en un repositorio de Github. Esta contendrá un Jenkinsfile que llamará al repositorio que hará la función de librería compartida.

En el repositorio principal (en el que se aloja la aplicación), se generarán ramas donde se podrá trabajar sobre la misma sin alterar la original. Por cada commit que se realice en una rama se compilara el **entorno dinámico** (entorno de trabajo), es decir, se ejecutará el pipeline que generará el entorno si no está creado, sino se actualizará.

El proceso de implementación de la aplicación desde el repositorio de Github hasta su despliegue en Kubernetes a través de un entorno dinámico involucra una serie de etapas interconectadas. Estas fases se definen a través de un pipeline, el cual sigue estos pasos:

- 1. Configuración de variables:** En esta fase, se definen y configuran las variables necesarias para la construcción y despliegue del entorno dinámico.
- 2. Empaquetado con Docker:** Aquí se lleva a cabo la creación de contenedores Docker que encapsulan la aplicación y sus dependencias, permitiendo una ejecución consistente en distintos entornos.
- 3. Despliegue en Kubernetes:** Una vez creados los contenedores, el pipeline ejecuta el despliegue en un clúster de Kubernetes, haciendo uso de las configuraciones y recursos proporcionados por la librería compartida.

El Jenkinsfile presente en el repositorio principal actúa como el “conductor” de este flujo de trabajo, orquestando las distintas etapas del proceso. Al generarse ramas en el repositorio principal, se posibilita el desarrollo paralelo de la aplicación sin afectar la versión original. Cada vez que se realiza un commit en una de estas ramas se ejecutará la construcción y despliegue del entorno dinámico asociado a esa rama específica.

Como se ha indicado anteriormente, para la creación de este entorno, se utilizará un segundo repositorio que actúa como una librería compartida. Este repositorio contiene una estructura organizada que engloba scripts, configuraciones y recursos necesarios para el correcto funcionamiento del pipeline. Este repositorio presentará la siguiente estructura:

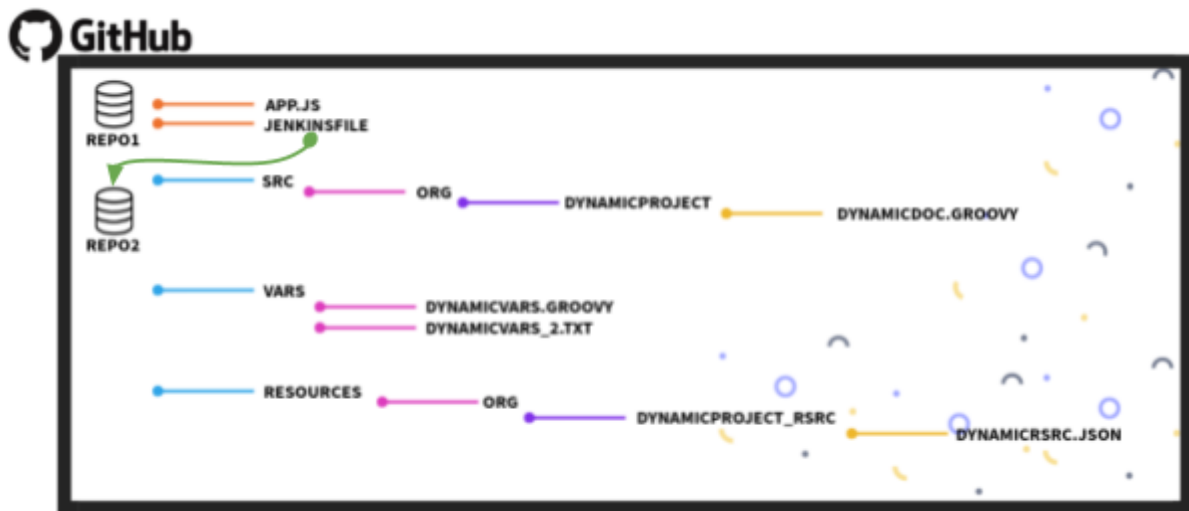
```
+-- src
|   +- org
|       +- DynamicProject
|           +- DynamicDoc.groovy
+-- vars
|   +- DynamicVars.groovy
|   +- DynamicVars_2.txt
+-- resources
|   +- org
|       +- DynamicProject_rsrc
|           +- DynamicRsrc.json
```

Donde, según la página oficial de Jenkins sobre [librerías compartidas](#), cada directorio se podrían entender como:

- “El directorio **src** debería tener una estructura estándar de directorios de código fuente Java. Este directorio se agrega al classpath al ejecutar Pipelines.”
- “El directorio **vars** alberga archivos de script que se exponen como variables en los Pipelines. El nombre del archivo es el nombre de la variable en el Pipeline. Por lo tanto, si tuvieras un archivo llamado **vars/log.groovy** con una función como **def info{message}...** dentro, puedes acceder a esta función como **log.info "hello**

world" en el Pipeline. Puedes colocar tantas funciones como desees dentro de este archivo.”

- “Un directorio de recursos (**resources**) permite que el paso **libraryResource** sea utilizado desde una biblioteca externa para cargar archivos no-Groovy asociados.”



3. Fundamentos teóricos y conceptos

3.1 Software

3.1.1 Jenkins

Jenkins es una herramienta de integración continua de código abierto cuya función principal es automatizar la construcción, prueba y despliegue de aplicaciones, proporcionando una infraestructura que facilita la integración continua y la entrega continua (CI/CD). Algunas de las características clave de Jenkins son:

- Integración Continua: Facilita la integración continua al monitorear constantemente los repositorios de código fuente y realizar automáticamente la construcción y prueba cuando se detectan cambios, asegurando una integración fluida y rápida del código.
- Automatización de Procesos: Jenkins automatiza tareas repetitivas, como la compilación y prueba de código.
- Arquitectura Basada en Plugins: Su arquitectura modular basada en plugins aporta flexibilidad y extensibilidad. Los plugins permiten la integración con una amplia variedad de herramientas y tecnologías, adaptándose a los requisitos específicos del proyecto.
- Configuración Basada en Código: Permite definir y gestionar la configuración de Jenkins como código, facilitando la reproducción y mantenimiento del entorno, además de la integración con sistemas de control de versiones.
- Soporte Multiplataforma: Es compatible con diversas plataformas y entornos, lo que lo hace adecuado para proyectos que utilizan una variedad de tecnologías.

- Despliegue Continuo: Además de la integración continua, permite la entrega continua y el despliegue automático en diferentes entornos, desde el desarrollo hasta la producción.

En resumen, Jenkins es una herramienta esencial que permite la automatización eficiente de procesos clave y facilita la implementación exitosa de prácticas de integración y entrega continuas.

3.1.2 Docker

Docker es una plataforma de código abierto que facilita la creación, implementación y ejecución de aplicaciones en contenedores. Al integrar Docker en este proyecto se pueden aprovechar varias características que mejoran la eficiencia y la consistencia del desarrollo y la implementación del mismo. Algunas de las características son:

- Contenedores: Docker utiliza contenedores para crear entornos aislados. Son portátiles y consistentes, lo que facilita la implementación en diferentes entornos sin preocuparse por las diferencias en la infraestructura.
- Portabilidad: Se garantiza que una aplicación se comporte de la misma manera, independientemente de dónde se ejecute. Esto mejora la portabilidad y facilita la transferencia de aplicaciones entre entornos de desarrollo, prueba y producción.
- Imagen de Docker: Docker utiliza imágenes, que son plantillas de solo lectura que contienen el sistema operativo, las bibliotecas y las dependencias de una aplicación.
- Fichero Dockerfile: La configuración de una imagen de Docker se realiza a través de un archivo llamado Dockerfile. Este archivo describe los pasos necesarios para construir una imagen, lo que facilita la definición y reproducción de entornos de desarrollo de manera consistente.

- **Eficiencia:** Los contenedores comparten el sistema operativo y utilizan recursos de manera más eficiente en comparación con máquinas virtuales tradicionales. Esto permite ejecutar múltiples contenedores en un solo host sin sacrificar el rendimiento.
- **Escalabilidad horizontal:** La arquitectura de contenedores facilita la escalabilidad horizontal, permitiendo agregar o eliminar instancias de aplicaciones según la demanda, lo que mejora la capacidad de respuesta y la eficiencia.

Al integrar Docker en este proyecto, se aprovechan estas características para mejorar la consistencia, la eficiencia y la flexibilidad en el desarrollo y despliegue de la aplicación. Además, la capacidad de encapsular aplicaciones en contenedores ofrece una solución integral para abordar los desafíos de la implementación de software en entornos diversos y cambiantes.

3.1.3 Kubeadm

Kubeadm es una herramienta de código abierto cuya función principal es simplificar y automatizar la tarea de inicialización y gestión de clústeres de Kubernetes. Con kubeadm, se pueden crear clústeres de contenedores de manera más sencilla, facilitando la implementación y administración de aplicaciones en entornos distribuidos. Algunas características de esta herramienta:

- Kubeadm simplifica significativamente el proceso de inicialización de clústeres de Kubernetes, proporcionando un conjunto de comandos intuitivos que facilitan la creación rápida y eficiente de entornos Kubernetes.
- Utiliza archivos de configuración en formato YAML para definir la estructura del clúster. Esta metodología asegura una configuración consistente y facilita la replicación de la configuración en diferentes nodos, contribuyendo a un entorno uniforme.

- Kubeadm permite la configuración de clústeres altamente disponibles mediante la distribución de componentes críticos en varios nodos. Esto mejora la redundancia y la resistencia del clúster, garantizando un alto nivel de disponibilidad para las aplicaciones desplegadas.

3.1.4 Git

Git es un sistema de control de versiones distribuido que permite el seguimiento eficiente de cambios en el código fuente de un proyecto a lo largo del tiempo. Con la capacidad de trabajar de forma descentralizada, cada colaborador tiene una copia completa del historial y puede contribuir independientemente. Utiliza ramas para el desarrollo paralelo y facilita la fusión de características. Los cambios se registran mediante "commits", proporcionando instantáneas del estado del código con mensajes descriptivos. Los repositorios pueden ser locales o remotos, permitiendo la sincronización en equipos distribuidos. Git también gestiona conflictos y ofrece un historial transparente, convirtiéndose en una herramienta esencial para el desarrollo colaborativo y la gestión de versiones en proyectos de software.

3.1.5 Helm

Helm es una herramienta de gestión de paquetes para aplicaciones Kubernetes que simplifica y estandariza el despliegue de aplicaciones complejas en entornos de contenedores. Funciona mediante la definición de "charts", que son paquetes preconfigurados que contienen recursos y configuraciones necesarios para implementar aplicaciones en Kubernetes. Con Helm, los desarrolladores pueden versionar, compartir y desplegar aplicaciones de manera consistente. Utiliza un modelo de plantillas para parametrizar configuraciones, permitiendo la personalización de despliegues. Además, Helm facilita las actualizaciones y el rollback de aplicaciones, simplificando la administración de ciclos de vida complejos en entornos Kubernetes.

3.1.6 K8s Lens

Lens es una herramienta de código abierto que proporciona una interfaz gráfica unificada y poderosa para gestionar y supervisar clústeres de Kubernetes. Funciona como un "IDE" para entornos Kubernetes, permitiendo a los desarrolladores y

administradores visualizar y administrar recursos de clúster de manera eficiente. Con Lens, los usuarios pueden explorar nodos, pods, servicios y otros recursos de Kubernetes de manera intuitiva, realizar inspecciones detalladas y realizar operaciones de manera centralizada. También ofrece características como el acceso rápido a los registros de aplicaciones, la visualización de métricas de rendimiento y la capacidad de trabajar con varios clústeres simultáneamente. Lens simplifica la complejidad de la administración de Kubernetes al proporcionar una experiencia visual rica y unificada para los profesionales de DevOps y desarrolladores.

4. Descripción

4.1 Demostración del escenario

Para demostrar el funcionamiento del entorno dinámico se van a seguir estos pasos:

1. Creación del entorno dinámico: Se crea una rama en base a la rama main, seguidamente se lanzará el pipeline multibranch, es decir, un pipeline que se ejecuta cuando se crea o se modifica una rama. Este pipeline se ejecuta desde el Jenkinsfile definido en el repositorio principal llamando a la librería compartida alojada en un segundo repositorio. Se generará una imagen Docker que se utilizará para desplegar la aplicación en Kubernetes, se va a usar un chart de Helm con el que se definirá los templates que se usarán para que funcione la aplicación.
2. Actualización del entorno dinámico: Cada vez que se realice un commit en la rama creada anteriormente, se va a volver a lanzar el pipeline actualizando el entorno con los cambios creados en la aplicación, es decir, se compilará y se reiniciará el deployment asociado en Kubernetes haciendo que se descargue la imagen con los nuevos cambios.
3. Borrado del entorno dinámico: Cada cierto tiempo se va a ejecutar de forma automática un pipeline que comprobará que las ramas existentes en el repositorio coinciden con los namespaces desplegados en Kubernetes. Al eliminar una rama, deja de coincidir con el namespace existente y se borrará este último.

4.1.2 Pasos a seguir

Voy a mejorar la explicación de cada paso:

1. Desactivar la Swap:

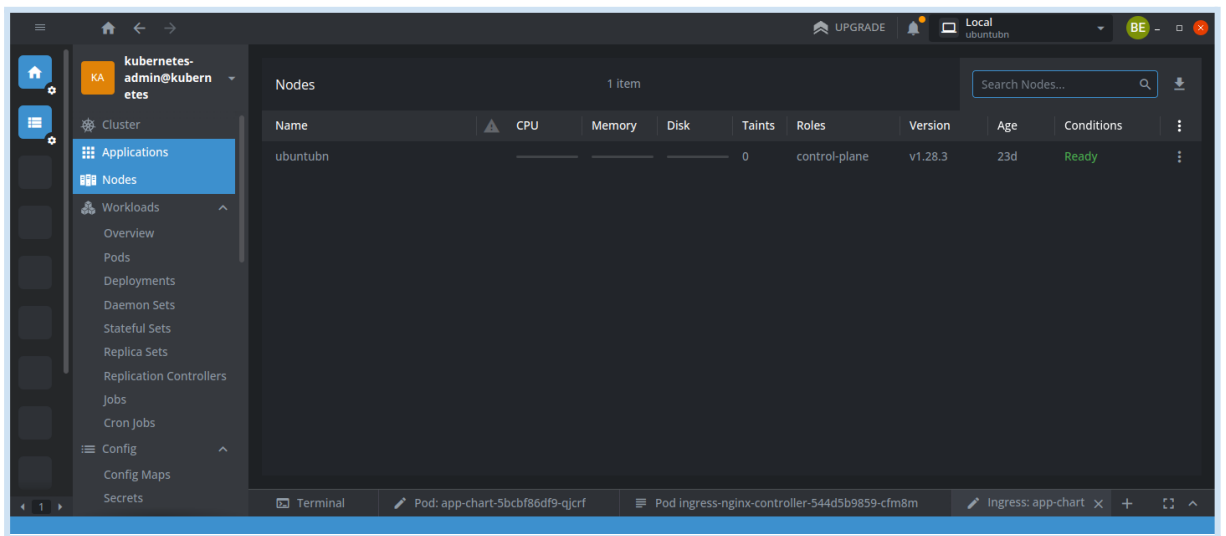
```
sudo swapoff -a
```

2. Iniciar el Escenario Kubernetes:

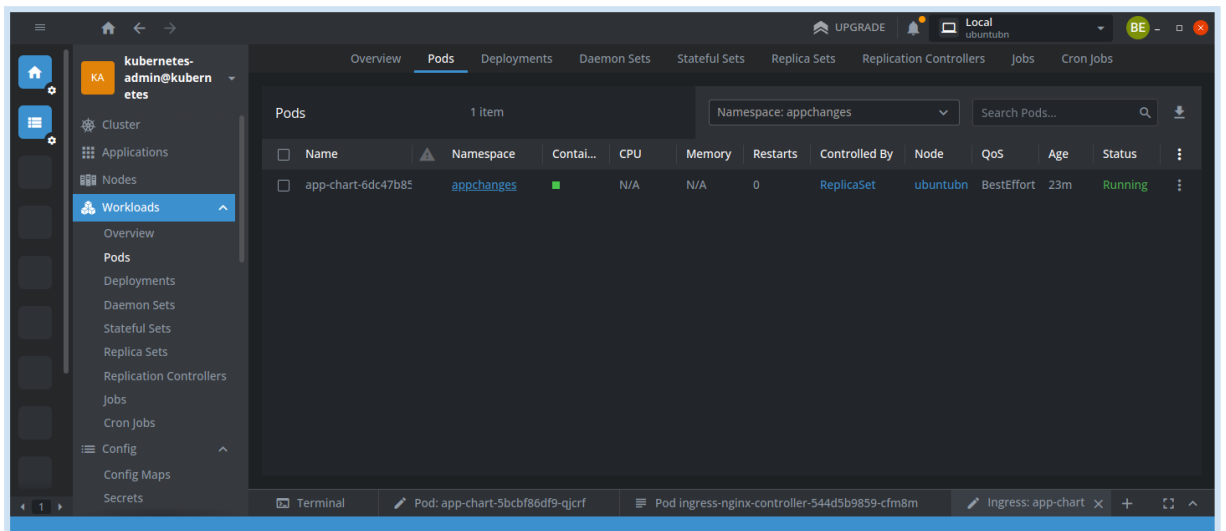
```
kubeadm init
```

Se puede verificar el estado del clúster de Kubernetes de manera más visual y directa utilizando K8s Lens:

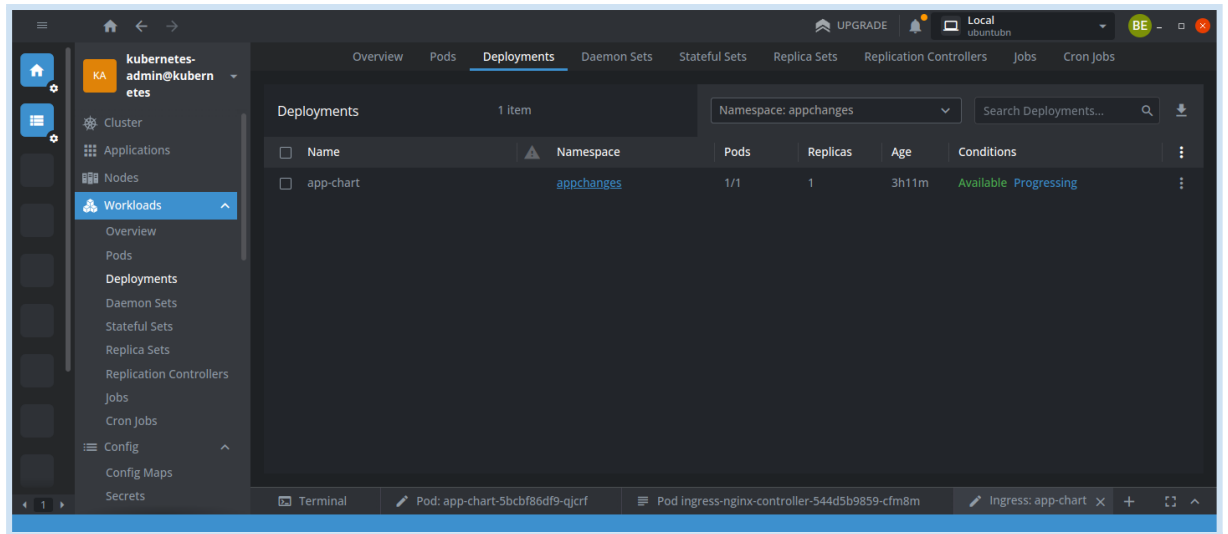
- Nodos:



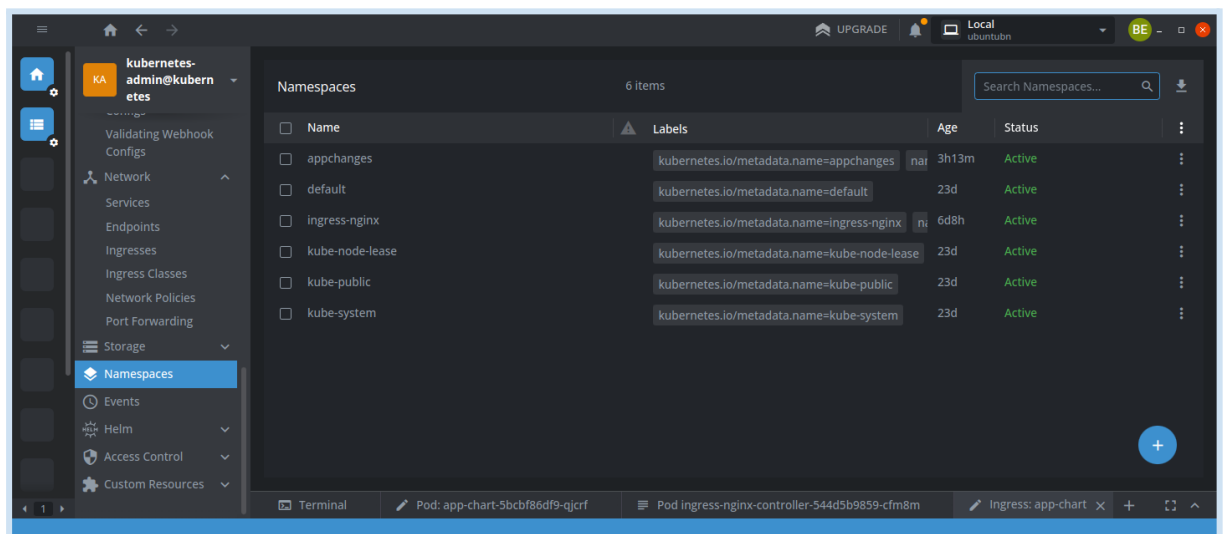
- Pods:



- Deployments:



- Namespaces:



3. Verificar el Estado de Jenkins:

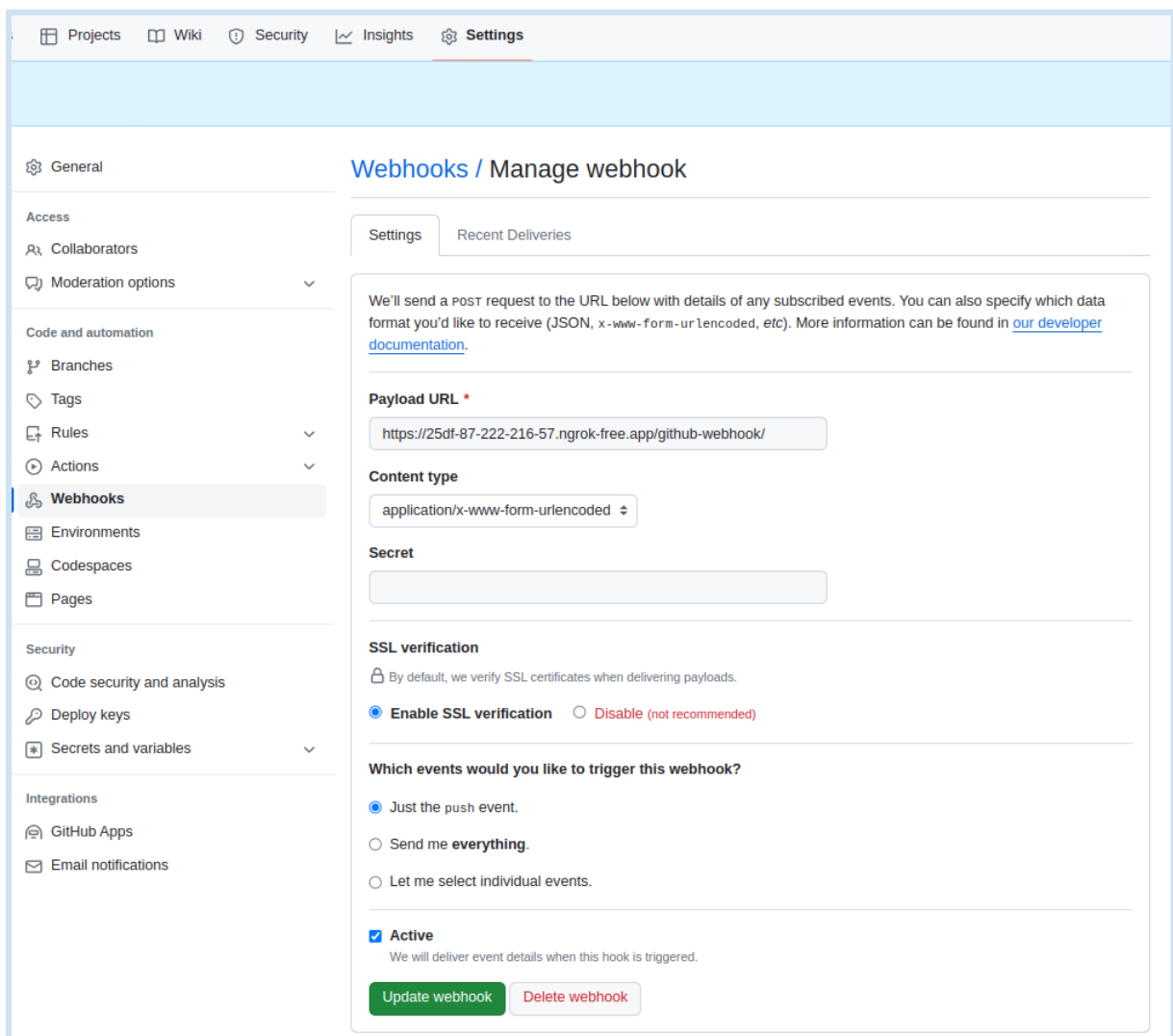
```
sudo systemctl status jenkins
```

4. Utilizar Ngrok para obtener una URL pública y redirigir el tráfico desde el puerto 8080 con el siguiente comando:

```
ngrok http 8080
```

5. Copiar la URL generada por Ngrok y configurarla en la sección de Webhooks del repositorio "App". Esto permitirá recibir notificaciones por cada cambio realizado en el repositorio.

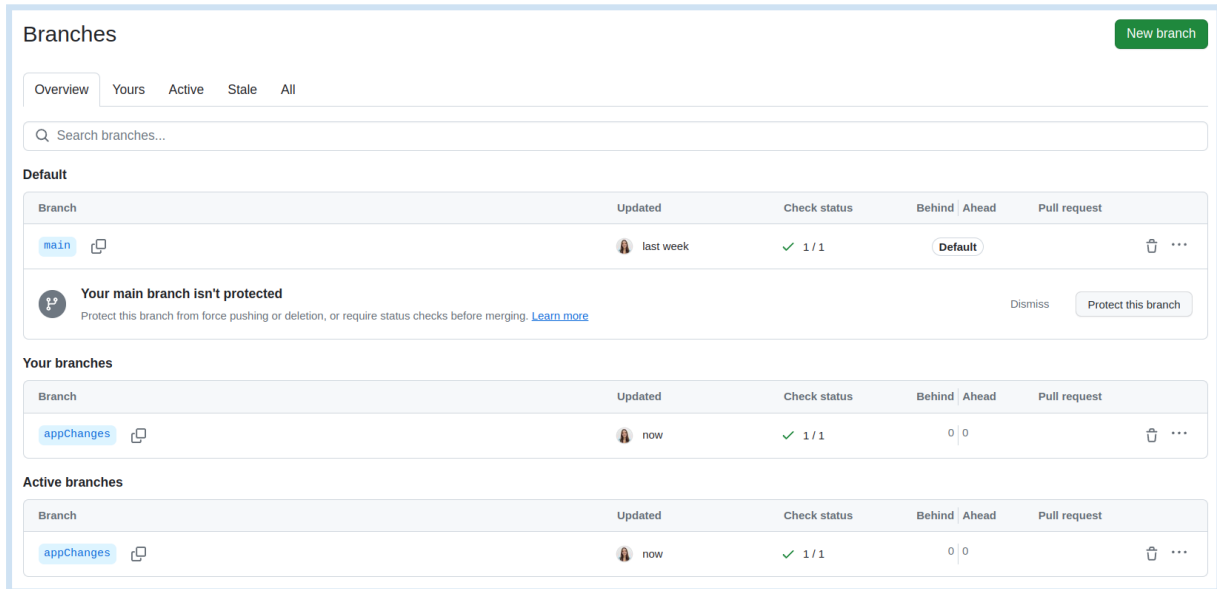
```
https://25df-87-222-216-57.ngrok-free.app/github-webhook/
```



The screenshot displays the GitHub Settings interface for a repository. The 'Settings' tab is selected, and the 'Webhooks / Manage webhook' section is active. The left sidebar shows the 'Webhooks' menu item highlighted. The main content area contains the following configuration options:

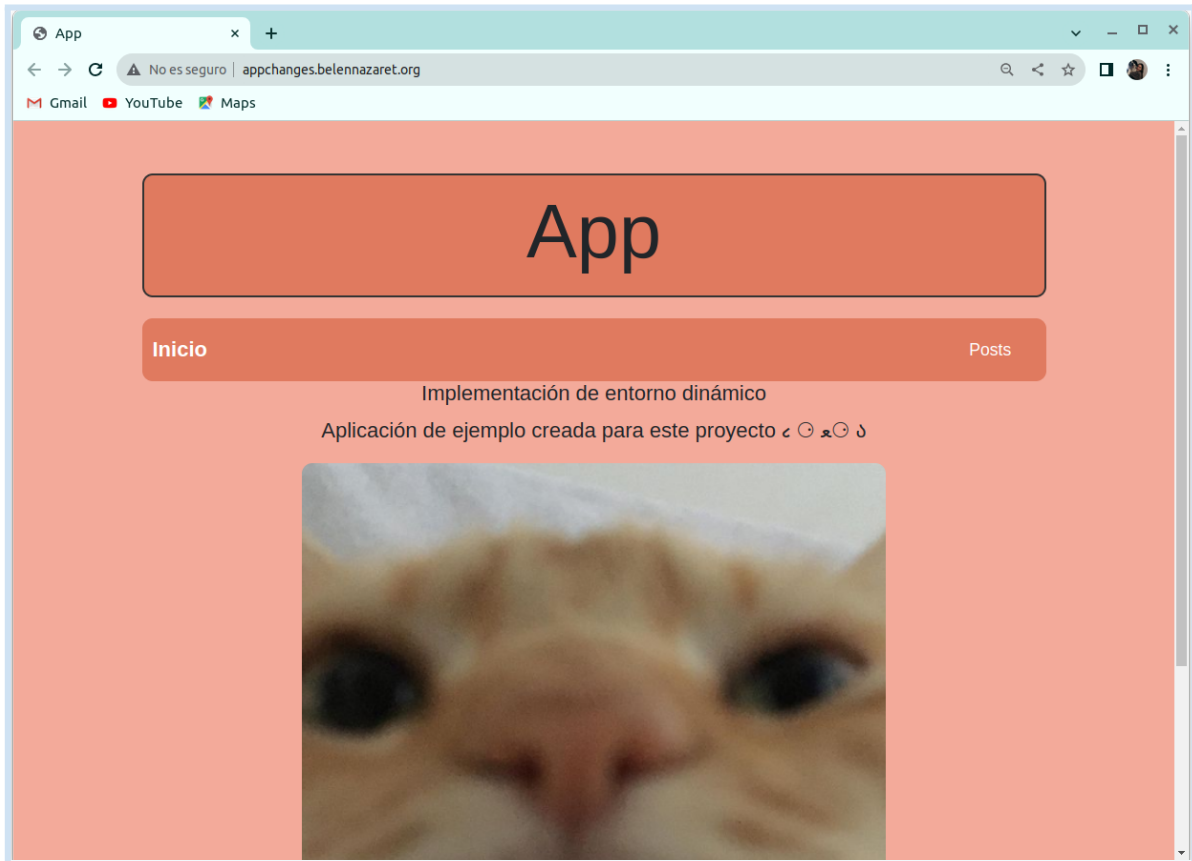
- Settings** (selected) and **Recent Deliveries** tabs.
- General** section: A note stating, "We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#)."
- Payload URL ***: A text input field containing the URL `https://25df-87-222-216-57.ngrok-free.app/github-webhook/`.
- Content type**: A dropdown menu set to `application/x-www-form-urlencoded`.
- Secret**: An empty text input field.
- SSL verification**: A section with a lock icon and the text "By default, we verify SSL certificates when delivering payloads." Below it are two radio buttons: **Enable SSL verification** (selected) and **Disable (not recommended)**.
- Which events would you like to trigger this webhook?**: Three radio buttons: **Just the push event.** (selected), **Send me everything.**, and **Let me select individual events.**
- Active**: A checked checkbox with the text "We will deliver event details when this hook is triggered."
- At the bottom, there are two buttons: **Update webhook** (green) and **Delete webhook** (red).

6. Se crea una rama nueva en el repositorio “app” donde se trabajara en las modificaciones de la aplicación:

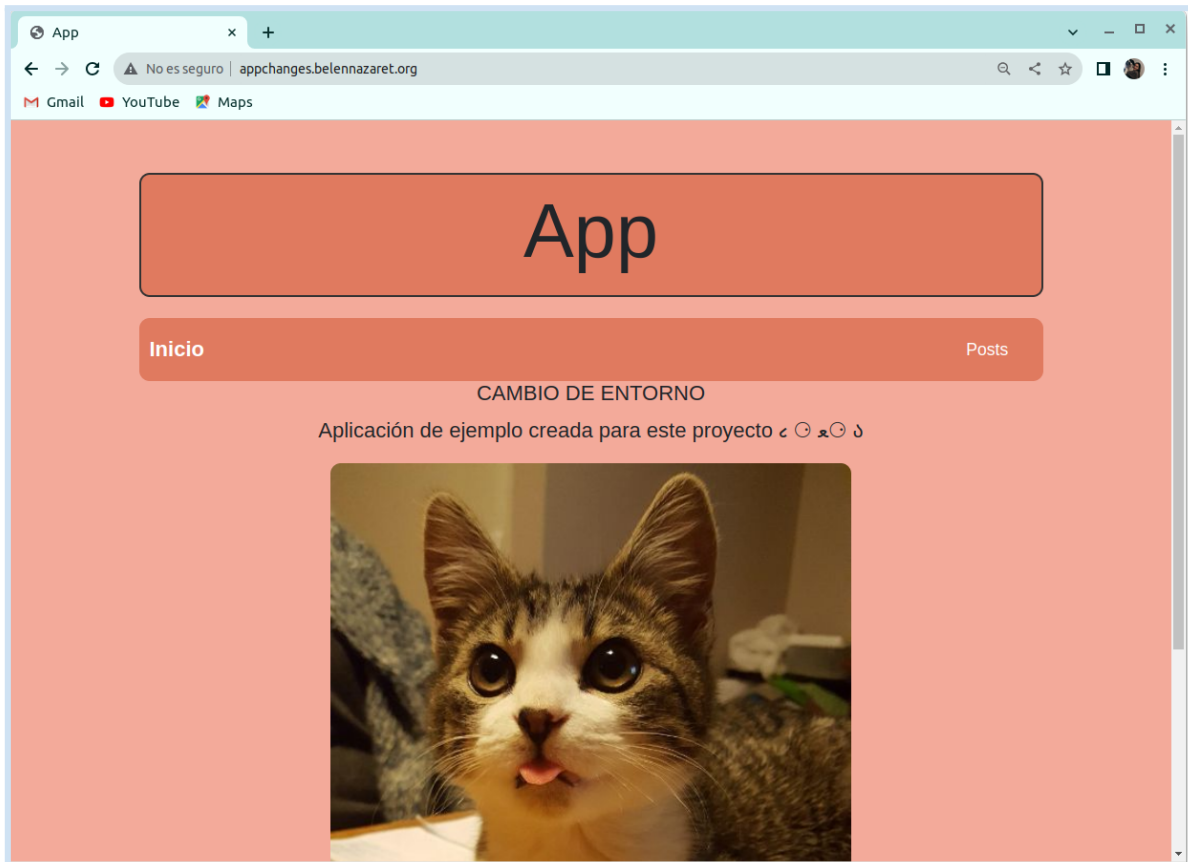


Al crear la rama nueva se genera un nuevo apartado dentro del pipeline multibranch “EntornoDinamico” de Jenkins, se hará un escaneo de la rama en busca de un Jenkinsfile. Automáticamente, si existe, se ejecutará.

7. El Jenkinsfile asociado a la nueva rama realizará varias acciones, incluyendo la transformación de nombres, la creación y carga de la imagen Docker, la implementación en Kubernetes y la adición de la dirección IP y URL al archivo “/etc/hosts” para permitir la ejecución local de la aplicación.



8. Realizar cambios en la aplicación y confirmar que, con cada commit, se vuelva a ejecutar el Jenkinsfile y se apliquen los cambios al escenario.



9. Se elimina la rama creada anteriormente, gracias al cron job de borrado (pipeline de borrado en Jenkins) se eliminará el namespace y todo lo asociado al mismo. Este pipeline, compara las ramas existentes en el repositorio de Github con los namespaces existentes en Kubernetes, si no existe la rama se elimina el namespace y con ello los recursos asociados como los pods.

4.2 Instalaciones y configuraciones

4.2.1 Containerd

Antes de instalar Kubernetes, es necesario configurar el entorno para admitir un runtime de contenedores, en este caso, containerd. Esto implica realizar ajustes en la configuración del sistema para permitir el enrutamiento de paquetes IPv4 desde una ubicación de origen a una ubicación de destino y permitir que iptables supervise el tráfico de red en modo bridge.

Se debe de llevar a cabo los siguientes pasos:

1. Para garantizar un entorno limpio es necesario desinstalar todos los paquetes que puedan resultar conflictivos ejecutando el siguiente comando:

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

2. Se cargan los módulos necesarios para el entorno Kubernetes. Esto se realiza mediante la creación y ejecución de un archivo de configuración llamado k8s.conf en el directorio /etc/modules-load.d/. Los módulos overlay y br_netfilter son esenciales para el funcionamiento de Kubernetes:

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
```

```
sudo sysctl --system
```

3. Si queremos probar que funciona correctamente ejecutamos los siguientes comandos:

```
lsmod | grep br_netfilter  
lsmod | grep overlay
```

4. A continuación, verificamos que todas las variables del sistema se establecen en 1:

```
sysctl net.bridge.bridge-nf-call-iptables  
net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
```

5. Lo siguiente es agregar la clave GPG oficial de Docker para garantizar la autenticidad de los paquetes:

```
sudo apt-get update  
sudo apt-get install ca-certificates curl gnupg  
sudo install -m 0755 -d /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |  
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

6. Añadimos el repositorio de Docker a las fuentes de Apt, lo cual nos permitirá instalar Docker y containerd desde este repositorio:

```
echo "deb [arch=\"$(dpkg --print-architecture)\"  
signed-by=/etc/apt/keyrings/docker.gpg]  
https://download.docker.com/linux/ubuntu \"$(. /etc/os-release  
&& echo \"$VERSION_CODENAME\")\" stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null  
  
sudo apt-get update
```

7. Instalar containerd y cambiar el fichero de configuración de este para que kubernetes pueda usarlo:

```
sudo apt-get install containerd.io  
containerd config default > /etc/containerd/config.toml
```

8. Se edita el archivo /etc/containerd/config.toml incluyendo la siguiente sección para habilitar SystemdCgroup:

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]  
  
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.opti  
ons]  
    SystemdCgroup = true
```

9. Se edita el archivo `/etc/crictl.yaml` y se añade la configuración necesaria para establecer la comunicación con containerd:

```
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint:  unix:///run/containerd/containerd.sock
timeout: 10
debug: true
```

10. Finalmente, reiniciamos containerd para aplicar los cambios realizados en su configuración:

```
sudo systemctl restart containerd
```

4.2.2 Docker

Continuamos con la instalación y configuración de Docker, a continuación, se detallan los pasos a seguir:

1. Instalación de los paquetes necesarios:

```
sudo apt-get install docker-ce docker-ce-cli
docker-buildx-plugin docker-compose
```

2. Después de la instalación, hay que agregar el usuario (propio) al grupo "docker" para poder ejecutar comandos Docker sin necesidad de utilizar "sudo":

```
sudo usermod -aG docker $USER
```

3. Para que los cambios en los grupos se hagan efectivos, se actualiza el grupo actual con el siguiente comando:

```
newgrp docker
```

4. Es necesario activar el bit de forward para permitir que los contenedores se comuniquen con el exterior:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

4.2.3 Helm

Instalación de Helm (Versión 3)

Para integrar Helm en el proyecto, es necesario seguir estos pasos:

1. Instalación de Helm:

```
curl -fsSL -o get_helm.sh  
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-  
3  
chmod 700 get_helm.sh  
./get_helm.sh
```

2. Creación de una estructura básica de Chart con Helm con el nombre especificado. La estructura incluye archivos y directorios esenciales para definir y desplegar aplicaciones:

```
helm create <nombre_chart>
```


- La estructura generada:

```
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

3. Inicio del Chart con Helm:

```
helm upgrade app --install -f values.yaml . -n namespace
--create-namespace --set image.tag=test --set
'ingress.hosts[0].host=belennazareth.org'
```

- upgrade: Actualiza el chart si ya existe.
- --install: Instala el chart si no existe.
- -f values.yaml: Utiliza el archivo “values.yaml” para configurar valores específicos.
- -n namespace: Especifica el nombre del namespace.
- --create-namespace: Crea el namespace si no existe.
- --set image.tag=test: Modifica el tag de la imagen en el chart.
- --set 'ingress.hosts[0].host=belennazareth.org': Configura el host para el Ingress.

Es importante asegurarse de tener Helm versión 3 y no la versión 2, ya que la sintaxis y la estructura han cambiado significativamente entre versiones.

4.2.4 Kubernetes

Una vez que ya contamos con un runtime de contenedores (containerd) y con Docker para construir y administrar contenedores, podemos proceder con la instalación y configuración de kubernetes. Este se encarga de la orquestación y gestión a nivel de clúster en producción.

1. Descargar la clave pública para autenticar los paquetes de Kubernetes:

```
curl -fsSL  
https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key |  
sudo gpg --dearmor -o  
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

2. Introducir el repositorio de Kubernetes al sistema y actualización de la información de los repositorios:

```
echo 'deb  
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]  
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee  
/etc/apt/sources.list.d/kubernetes.list  
sudo apt update
```

3. Instalar paquetes de Kubernetes:

```
sudo apt install -y kubelet kubeadm kubectl
```

4. Marcar versión para evitar que los paquetes de kubernetes se actualicen automáticamente:

```
sudo apt-mark hold kubelet kubeadm kubectl
```

5. Desactivar el espacio de intercambio (Swap) ya que Kubernetes no funciona correctamente con la Swap activa:

```
sudo swapoff -a
```

6. Inicializar el clúster de Kubernetes con kubeadm:

```
sudo kubeadm init --pod-network-cidr=10.1.0.0/16  
--apiserver-advertise-address=192.168.1.144
```

- --pod-network-cidr: Establece la red de pods en 10.1.0.0/16.
- --apiserver-advertise-address: Anuncia la dirección IP del servidor de API en 192.168.1.144.

7. Configurar el entorno del usuario para que pueda usar la herramienta de línea de comandos kubectl para interactuar con el clúster de Kubernetes:

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

8. Verificar la creación del cluster y que esté activo:

```
kubectl get nodes
```

9. Listar los pods en el namespace "kube-system" para comprobar que se despliegan los componentes del sistema de Kubernetes, como los controladores de replicación, servicios de red, y otras utilidades que hacen que el clúster funcione correctamente:

```
kubectl get pods -n kube-system
```

10. Instalar Weave para permitir la comunicación entre los pods:

```
kubectl apply -f  
https://github.com/weaveworks/weave/releases/download/v2.8.1/  
weave-daemonset-k8s.yaml
```

11. Lo último será quitar la restricción que impide la programación de pods en el nodo de control (Control Plane):

```
kubectl taint nodes ubuntu  
node-role.kubernetes.io/control-plane:NoSchedule-
```

Ahora tenemos un clúster, pero no tenemos un punto de entrada para el tráfico ni tenemos un método para enrutar nuestro tráfico. Esto se hace con un controlador de ingreso. Se usa helm para instalar, configurar y administrar el controlador de ingreso. En este caso, usaremos el controlador "ingress-nginx".

Primero hay que configurar helm y decirle dónde está el repositorio “ingress-nginx” para que pueda recuperar los datos que necesita para manejar nuestra implementación.

```
helm repo add nginx  
https://kubernetes.github.io/ingress-nginx
```

Normalmente, tendríamos un balanceador de carga u otro sistema de alta disponibilidad para enrutar el tráfico a nuestro clúster, pero como se trata de un laboratorio local, configuraremos el controlador de ingreso para usar la red del host para permitir un fácil acceso.

Para hacer eso, es necesario realizar la implementación de “ingress-nginx” con el siguiente comando:

```
helm install \  
  --create-namespace \  
  --namespace ingress-nginx \  
  ingress-nginx \  
  nginx/ingress-nginx \  
  --set controller.hostNetwork=true
```

- helm install: Este comando se utiliza para instalar un paquete Helm en un clúster de Kubernetes.
- --create-namespace: Crea el espacio de nombres (namespace) “ingress-nginx” si aún no existe.
- --namespace ingress-nginx: Especifica el espacio de nombres donde se instalará el controlador de Ingress.
- ingress-nginx: Es el nombre que se le dará a la release (instancia) del paquete Helm.

- `nginx/ingress-nginx`: Especifica el nombre del repositorio de Helm (`nginx`) y el nombre del chart (`ingress-nginx`) que se instalará.
- `--set controller.hostNetwork=true`: Establece una configuración específica para el controlador de Ingress para que use la red del host.

4.2.5 Configuración de Jenkins y Desarrollo de la Aplicación

A continuación, se presenta la configuración de Jenkins y el desarrollo básico de la aplicación:

1. Descarga e instalación de la clave pública de Jenkins:

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \  
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
```

2. Configuración del repositorio de Jenkins:

```
echo 'deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \  
https://pkg.jenkins.io/debian-stable binary' | sudo tee \  
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

3. Actualización de paquetes e instalación de Java y Jenkins:

```
sudo apt-get update  
sudo apt-get install default-jre jenkins
```

4. Concesión de permisos a Jenkins:

```
nano /etc/sudoers
└─jenkins ALL=(ALL) NOPASSWD: ALL
```

5. Con el repositorio de la aplicación ya creado, se realiza la instalación e inicialización de npm:

```
sudo apt install npm
npm init -y
```

El comando “npm init -y” inicializa un nuevo proyecto Node.js y, como resultado, crea un archivo llamado “package.json”. Este archivo proporciona una descripción detallada de la configuración del proyecto, facilitando la reproducción del entorno de desarrollo en otros sistemas y asegurando la coherencia entre diferentes entornos de desarrollo. Para instalar las dependencias y herramientas especificadas en el archivo “package.json” se ejecuta “npm install”.

7. Instalar Express, con esto se agrega el framework (Express) a nuestro proyecto para simplificar y organizar el desarrollo de aplicaciones web en Node.js. También se instala EJS para incorporar el motor de plantillas (EJS), facilitando la construcción de vistas web de manera eficiente. En conjunto resultan como una buena optimización del proceso de desarrollo:

```
npm install express
npm install ejs
```

Incluimos “node_modules” en el archivo “.gitignore” para evitar que los módulos de “Node.js” se carguen en el repositorio de GitHub.

8. Creación del archivo “app.js”, archivo principal que contiene la configuración y la lógica central de la aplicación:

```
const express = require('express');
const app = express();
const PORT = 3000;

app.set('view engine', 'ejs');
app.use(express.static('public'));

app.get('/', (req, res) => {
  res.render('index', {
    title: 'App',
    subtitle: 'Implementación de entorno dinámico',
    text: 'Aplicación de ejemplo creada para este proyecto',
    imageUrl: 'https://cataas.com/cat',
    altText: 'Gato aleatorio',
    colorfondo: '#f3aa9a',
    titleBoxColor: '#e07a5f'
  });
});

app.get('/posts', (req, res) => {
  res.render('posts', {
    postTitle: 'Posts',
    postText: 'Aquí estarán todos los posts',
    postFondoColor: '#f3aa9a'
  });
});

app.listen(PORT, () => {
  console.log(`Servidor Express iniciado en http://localhost:${PORT}`);
});
```


9. Creación del directorio de vistas (“views”) y el archivo “index.ejs” como motor de plantillas para poder usar variables:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title><%= title %></title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/
bootstrap.min.css">
  <style>
    body {
      background-color: <%= colorfondo %>;
    }
    .container {
      text-align: center;
      padding: 50px;
    }
    .title-box {
      border: 2px solid #333;
      padding: 10px;
      border-radius: 10px;
      margin-bottom: 20px;
      background-color: <%= titleBoxColor %>;
    }
    img {
      max-width: 100%;
      height: auto;
      border-radius: 10px; /* Añade esquinas redondeadas a la
imagen */
    }
    .navbar {
      background-color: <%= titleBoxColor %>;
      border-radius: 10px;
      padding: 10px;
    }
    .navbar-brand {
      color: #fff;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="title-box">
      <h1><%= title %></h1>
    </div>
    <img alt="Placeholder for an image" data-bbox="125 196 872 892" />
    <div class="navbar">
      <span class="navbar-brand"><%= title %></span>
    </div>
  </div>
</body>
</html>
```

```
    }
    .navbar-nav {
      margin-left: auto;
    }
    .nav-item {
      margin-right: 15px;
    }
    .nav-link {
      color: #fff;
    }
  }
</style>
</head>
<body>
  <div class="container">
    <div class="title-box">
      <h1 class="display-3"><%= title %></h1>
    </div>
    <nav class="navbar navbar-expand-lg">
      <a class="navbar-brand" href="/">Inicio</a>
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link" href="/posts">Posts</a>
        </li>
      </ul>
    </nav>
    <h2 class="lead"><%= subtitle %></h2>
    <p class="lead"><%= text %></p>
    "
class="img-fluid">
  </div>
  <script
src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></scri
pt>
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/u
md/popper.min.js"></script>
  <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bo
otstrap.min.js"></script>
</body>
</html>
```

10. Para poder iniciar la aplicación usando “npm run”, se edita el fichero “package.json” añadiendo una entrada dentro del bloque “scripts”:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node app.js" <---  
},
```

11. Para permitir el acceso externo a Jenkins se instala ngrok, con esto también se consigue ejecutar el pipeline cada vez que se haga un commit en cualquier rama:

```
curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | \  
sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null && \  
echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | \  
\ \  
sudo tee /etc/apt/sources.list.d/ngrok.list && \  
sudo apt update && sudo apt install ngrok
```

12. Configuración del token de autenticación de ngrok entrando en la cuenta y copiando el siguiente comando:

```
ngrok config add-authtoken <TOKEN>
```

13. Inicio de Jenkins y ngrok :

```
sudo systemctl start jenkins  
ngrok http 8080
```

14. Por último, se accede al enlace generado “http://localhost:8080/” y se añade un token generado (previamente) en GitHub para pasar las credenciales a Jenkins. Seguidamente, en Jenkins, se crea una nueva tarea como carpeta llamada “app” y configuramos la adición de una biblioteca:

- Nombre: shared-library
- Versión predeterminada: main
- Seleccionar "modern scm"
- Repositorio del proyecto: Repositorio GitHub de las bibliotecas “shared-library”
- Credenciales: Utilizamos las credenciales de GitHub.

Dentro de la tarea “app”, se agrega un nuevo ítem llamado “EntornoDinamico”:

- Seleccionar la opción de ser multibranch.
- Añadir el repositorio de la aplicación en “branch sources”.
- Añadir un origen Git.
- Repositorio del proyecto: Repositorio GitHub de la aplicación.
- Credenciales: Utilizar las credenciales previamente creadas.
- Comportamientos: Añadir un filtro de nombre (con comodines) para incluir todo excepto la rama “main”.
- Activar la opción de escaneo automático del pipeline multibranch y configurar el intervalo de escaneo a 1 minuto.

Para el borrado, dentro de “app”, se crea un cron job que se ejecuta cada minuto. Es necesario marcar:

- Ejecutar periódicamente: * * * * *
- Definir como SCM e indicar la opción Git.
- Añadir el repositorio que se va a usar, en este caso “shared-library”.
- Añadir las credenciales creadas para Github.
- Especificar la rama “main” para que no sea examinada.
- Indicar la dirección del directorio y el fichero en el que se encuentra el pipeline de borrado, en este caso “/vars/entDestroy.groovy”.

5. Conclusión

En mi opinión, este proyecto ha sido una buena excusa para adentrarme un poco más en el uso de estas tecnologías y poder comprobar el gran potencial que tienen. He podido ampliar mis habilidades en el desarrollo y despliegue de aplicaciones, aunque sea un nivel básico. Aprendí de manera clara cómo herramientas como Jenkins, Docker, Git y otras colaboran para optimizar el ciclo de vida de una aplicación. En resumen, este proyecto me ha proporcionado conocimientos prácticos esenciales y una visión valiosa sobre la implementación de infraestructuras más actuales.

6. Referencias

6.1 Recursos

6.1.1 Aplicación

<https://github.com/belennazareth/app>

6.1.2 Librería compartida (Shared Library)

<https://github.com/belennazareth/shared-library>

6.1.3 Presentación

<https://view.genial.ly/657601a04fa94400150e42ed>

<https://youtu.be/tOoJyKqUbl8?si=6d5CE0u7PjpFrUzp>

6.2 Bibliografía

Shared libraries:

<https://www.jenkins.io/doc/book/pipeline/shared-libraries/#directory-structure>

Creación de imágenes, esquemas y presentación:

<https://genial.ly/es/>

Jenkins:

<https://rootstack.com/es/technology/jenkins>

<https://sentr.io/blog/que-es-jenkins/>

<https://www.jenkins.io/doc/>

<https://ngrok.com/docs/getting-started/?os=linux>

<https://plugins.jenkins.io/docker-plugin/>

<https://blogs.perficient.com/2022/08/23/configuring-jenkins-jobs-by-using-cron/>

Containerd:

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#forwarding-ipv4-and-letting-iptables-see-bridged-traffic>

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#containerd>

<https://github.com/containerd/containerd/blob/main/docs/getting-started.md>

<https://docs.docker.com/engine/install/ubuntu/>

Docker:

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

[https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))

<https://www.redhat.com/es/topics/containers/what-is-docker#ventajas-de-docker>

Kubeadm:

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

<https://keepcoding.io/blog/que-es-kubeadm/>

Kubernetes:

<https://www.smoothnet.org/kubernetes-on-linux-with-kubeadm/>

Helm:

https://helm.sh/es/docs/intro/using_helm/

<https://www.redhat.com/es/topics/devops/what-is-helm>

<https://www.freecodecamp.org/espanol/news/que-es-un-helm-chart-un-tutorial-para-principiantes-en-kubernetes/>

<https://helm.sh/docs/intro/install/>

https://helm.sh/docs/helm/helm_create/

<https://gist.github.com/mudssrali/acfb8b57e5847c7308e1064a739971da>

K8s Lens:

<https://k8slens.dev/>

Docker-Node:

https://hub.docker.com/_/node/

<https://hub.docker.com/layers/library/node/20.10.0/images/sha256-b41c3eccd3c2404464bdb e37f6ee28fc832b2d99b10caeea38b250a9b9d601e8?context=explore>