



# Comparación y Despliegue de Clústeres Locales de Kubernetes: KinD vs. Minikube

## ÍNDICE

1. Objetivo
2. Descripción
3. Introducción del proyecto
  - 3.1 Creación y presentación del escenario de Minikube
  - 3.2 Instalación de Minikube
    - 3.2.1 Hypervisor
    - 3.2.2 Docker y Kubectl
    - 3.2.3 Minikube
  - 3.3 Inicio de Minikube
    - 3.3.1 Que es un POD?
    - 3.3.2 Que es un DEPLOYMENT?
    - 3.3.3 Ejemplo de despliegue de Wordpress + MySQL con servicios y volúmenes persistentes.
    - 3.3.4 VOLUMENES
    - 3.3.5 SERVICIOS
    - 3.3.6 SECRETS
  - 4.1 Creación y presentación del escenario Kind (Kubernetes In Docker)
  - 4.2 Instalación de KinD
  - 4.3 Inicio de KinD
    - 4.3.1 Despliegue de Deployment y Daemonset en varios nodos
    - 4.3.2 ¿Qué es un DaemonSet?
5. Conclusión
6. Medios a utilizar

#### 6.1 Recursos Materiales:

##### Equipo de Cómputo

#### 6.2 Recursos Humanos:

#### 7. Tiempo de Ejecución:

#### 8. Presupuesto

#### 9. Título

#### 10. CONTROL DE VERSIONS:

## 1. Objetivo

**Comparar la Arquitectura:** Investigaremos cómo [KinD](#) y [Minikube](#) implementan los clústeres de Kubernetes y analizaremos las implicaciones de sus respectivas arquitecturas en términos de rendimiento y aislamiento.

**Requisitos y Dependencias:** Examinaremos los requisitos y dependencias necesarios para ejecutar [KinD](#) y [Minikube](#), y discutiremos cómo estos factores afectan la elección de una herramienta en función del entorno y los recursos disponibles.

**Velocidad y Eficiencia:** Realizaremos pruebas de rendimiento para determinar cuál de las dos herramientas es más rápida en la creación y destrucción de clústeres de Kubernetes.

**Despliegue de Ejemplos:** Proporcionaremos ejemplos detallados de cómo desplegar aplicaciones en los clústeres creados por [KinD](#) y [Minikube](#), resaltando las diferencias clave en el proceso de implementación.

**Conclusiones y Recomendaciones:** Resumiremos los hallazgos clave y proporcionaremos recomendaciones sobre cuándo utilizar [KinD](#) o [Minikube](#) en función de diferentes escenarios y necesidades de desarrollo.

## 2. Descripción

Este proyecto tiene como objetivo principal comparar dos herramientas ampliamente utilizadas para la creación de clústeres de Kubernetes en entornos locales: kind (Kubernetes IN Docker) y [Minikube](#). A través de esta comparación detallada, exploraremos las diferencias clave en términos de arquitectura, requisitos, velocidad, nivel de aislamiento y facilidad de uso entre kind y [Minikube](#).

Además de la comparación, este proyecto también incluirá ejemplos de despliegue práctico de clústeres de Kubernetes utilizando ambas herramientas. Esto permitirá a los usuarios comprender cómo implementar aplicaciones y servicios en estos clústeres y experimentar directamente con las diferencias en el proceso de implementación.

En última instancia, este proyecto proporcionará a los estudiantes y profesionales de la administración de sistemas informáticos en red (ASIR) una visión integral de las opciones disponibles para crear

clústeres de Kubernetes en entornos locales y les ayudará a tomar decisiones informadas sobre cuál de las dos herramientas es la más adecuada para sus necesidades específicas.

La idea para este proyecto fue generada a partir de una inspiración proporcionada por el profesor Manuel Corbelle

## 3. Introducción del proyecto

### 3.1 Creación y presentación del escenario de Minikube



El escenario para instalar [Minikube](#) se compone de una máquina virtual linux, concretamente la máquina tendrá instalado Ubuntu 20.04.1 LTS, conectada a una red NAT con el exterior.

- Necesitaremos 2 CPU o más
- 8 GB de memoria libre
- 20 GB de espacio libre en el disco
- Conexión a Internet
- Gestor de máquinas virtuales como Hyper-V, VirtualBox etc...

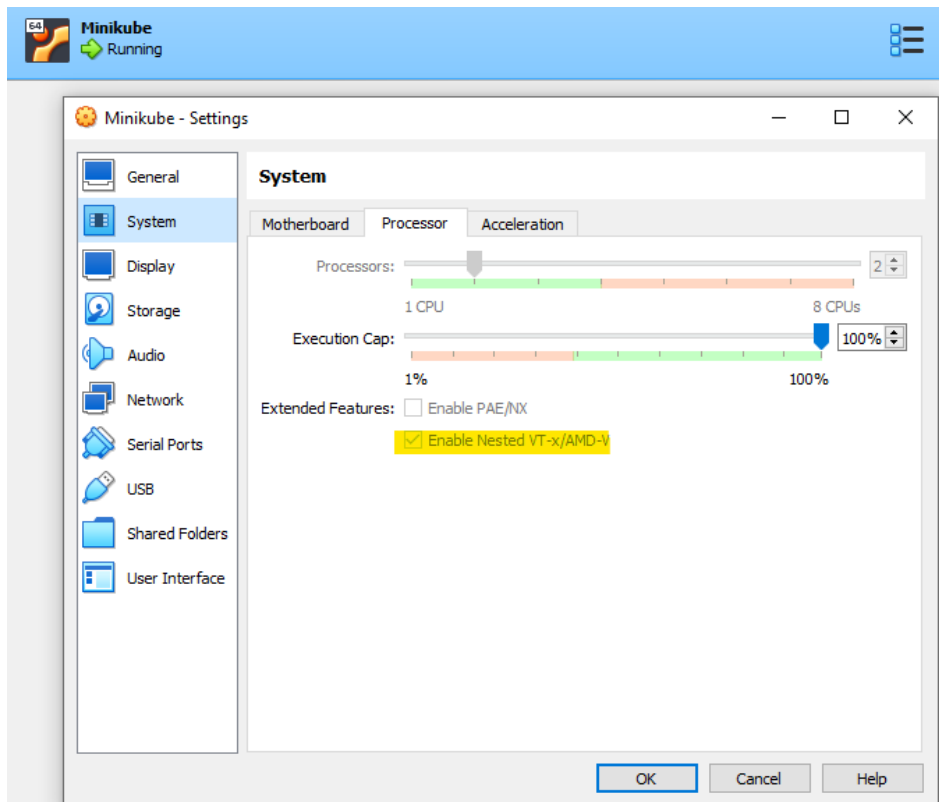
Para instalar [Minikube](#) en nuestra máquina virtual tendremos que activar la opción de virtualización anidada, ya que lo que hace [Minikube](#) es crear una máquina virtual.

Si lo estuviéramos haciendo en nuestra maquina local no habría nada más que instalar [Minikube](#) .

Para activar la opción de la virtualización anidada en nuestra máquina virtual anfitriona abriremos la consola e introducimos el siguiente comando

```
VBoxManage modifyvm Minikube --nested-hw-virt on
```

Luego haremos un reinicio y vemos en la configuración de nuestra máquina virtual que aparece como activada

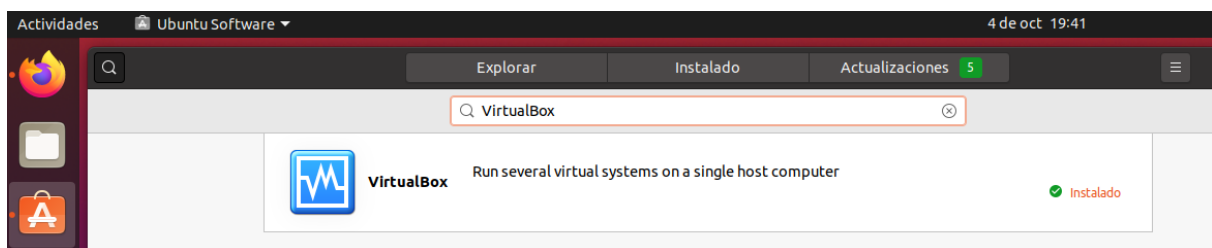


## 3.2 Instalación de Minikube

### 3.2.1 Hypervisor

Al llegar a la página oficial de [Minikube](#) vemos que nos hace falta un hipervisor en nuestra máquina virtual (virtualización anidada).

Instalamos desde el Ubuntu Software



### 3.2.2 Docker y Kubectl

Tenemos que asegurarnos de instalar **Docker** en nuestra máquina virtual ya que actúa como la base tecnológica que permite a Minikube y KinD crear entornos de Kubernetes locales de manera eficiente y simplificada, al ejecutar los nodos de Kubernetes como contenedores Docker en lugar de máquinas virtuales separadas.

**Kubectl** es una herramienta de línea de comandos para interactuar con los clústeres de kubernetes. **Kubectl** es la abreviatura de “Kube control” lo que significa que podremos desplegar aplicaciones, gestionar recursos, obtener información sobre el estado del clúster y mucho más.

**Kubectl** es una herramienta esencial para los administradores de sistemas, desarrolladores y operadores que trabajan con clústeres de Kubernetes. Les permite interactuar con el clúster de manera eficiente y realizar tareas de gestión y monitoreo de aplicaciones en contenedores de manera efectiva.

**Minikube** ya se encarga como parte de su instalación y su configuración, no hace falta instalarlo por separado.

Se usa con el comando minikube kubectl - -“comando de kubectl”

Sin embargo, si deseas interactuar con otros clústeres de Kubernetes (como clústeres en la nube), deberás instalar **Kubectl** por separado y configurarlo según tus necesidades.

Para instalar **Kubectl** en Ubuntu, puedes usar la sugerencia mencionada en el mensaje de error, que es utilizar Snap. Puedes instalar **Kubectl** a través de Snap con el siguiente comando:

```
sudo snap install kubectl --classic
```

```
minikube@minikube-VirtualBox:~$ sudo snap install kubectl --classic
Se ha instalado kubectl 1.28.3 por Canonical✓
```

Una vez que se complete la instalación, verifica la versión de **Kubectl** para asegurarte de que se ha instalado correctamente:

```
kubectl version --client
```

```
minikube@minikube-VirtualBox:~$ kubectl version --client
Client Version: v1.28.3
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

### 3.2.3 Minikube

Para instalar la última versión estable de **Minikube** en x86-64 Linux utilizando el paquete Debian

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
```

```
sudo dpkg -i minikube_latest_amd64.deb
```

### 3.3 Inicio de Minikube

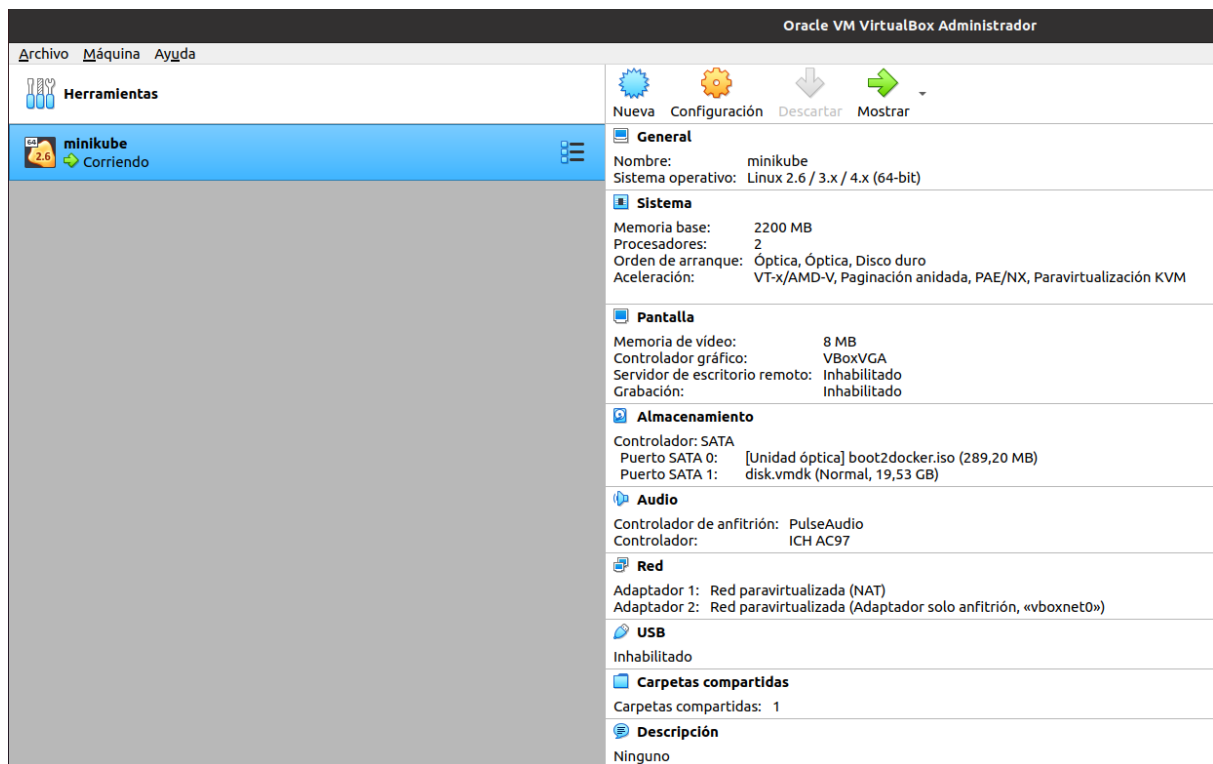
Para iniciar **Minikube** y crear nuestro clúster introducimos el comando

```
minikube start
```

esto buscará en internet una máquina virtual con Kubernetes ya instalado

```
minikube@minikube-VirtualBox:~$ minikube start
🌻 minikube v1.31.2 en Ubuntu 20.04
🔧 Controlador virtualbox seleccionado automáticamente
📥 Descargando la imagen de arranque de la VM
E1004 20:20:09.091409 212440 iso.go:90] Unable to download https://storage.googleapis.com/minikube-b
eapis.com/minikube-builds/iso/16971/minikube-v1.31.0-amd64.iso?checksum=file:https://storage.googleap
iso/amd64/minikube-v1.31.0-amd64.iso.download Pwd: Mode:2 Umask:----- Detectors:[0x3f9c8a8 0x3f9
7f08 tar:0xc000057eb0 tar.bz2:0xc000057ec0 tar.gz:0xc000057ed0 tar.xz:0xc000057ee0 tar.zst:0xc000057
f20 zst:0xc000057f18] Getters:map[file:0xc000d768f0 http:0xc000cf30e0 https:0xc000cf3130] Dir:false P
Error downloading checksum file: bad response code: 404
📥 Descargando la imagen de arranque de la VM
> minikube-v1.31.0-amd64.iso....: 65 B / 65 B [-----] 100.00% ? p/s 0s
> minikube-v1.31.0-amd64.iso: 289.20 MiB / 289.20 MiB 100.00% 44.49 MiB p
🔥 Starting control plane node minikube in cluster minikube
📥 Descargando Kubernetes v1.27.4 ...
> preloaded-images-k8s-v18-v1...: 393.21 MiB / 393.21 MiB 100.00% 30.07 M
🔥 Creando virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🔧 Preparando Kubernetes v1.27.4 en Docker 24.0.4...
  ■ Generando certificados y llaves
  ■ Iniciando plano de control
  ■ Configurando reglas RBAC...
🔗 Configurando CNI bridge CNI ...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔧 Verifying Kubernetes components...
🌟 Complementos habilitados: default-storageclass, storage-provisioner
🔧 kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🎉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Aquí vemos la máquina que nos creó y sus características



para ver los complementos que podemos añadir y sus estados podemos utilizar el comando

```
minikube addons list
```

```
minikube@minikube-VirtualBox:~$ minikube addons list
```

ADDON NAME	PROFILE	STATUS	MAINTAINER
ambassador	minikube	disabled	3rd party (Ambassador)
auto-pause	minikube	disabled	minikube
cloud-spanner	minikube	disabled	Google
csi-hostpath-driver	minikube	disabled	Kubernetes
dashboard	minikube	disabled	Kubernetes
default-storageclass	minikube	enabled ✓	Kubernetes
efk	minikube	disabled	3rd party (Elastic)
freshpod	minikube	disabled	Google
gcp-auth	minikube	disabled	Google
gvisor	minikube	disabled	minikube
headlamp	minikube	disabled	3rd party (kinvolk.io)
helm-tiller	minikube	disabled	3rd party (Helm)
inacel	minikube	disabled	3rd party (InAccel [info@inacel.com])
ingress	minikube	disabled	Kubernetes
ingress-dns	minikube	disabled	minikube
inspektor-gadget	minikube	disabled	3rd party (inspektor-gadget.io)
istio	minikube	disabled	3rd party (Istio)
istio-provisioner	minikube	disabled	3rd party (Istio)
kong	minikube	disabled	3rd party (Kong HQ)
kubevirt	minikube	disabled	3rd party (KubeVirt)
logviewer	minikube	disabled	3rd party (unknown)
metallb	minikube	disabled	3rd party (MetalLB)
metrics-server	minikube	disabled	Kubernetes
nvidia-driver-installer	minikube	disabled	3rd party (Nvidia)
nvidia-gpu-device-plugin	minikube	disabled	3rd party (Nvidia)
olm	minikube	disabled	3rd party (Operator Framework)
pod-security-policy	minikube	disabled	3rd party (unknown)
portainer	minikube	disabled	3rd party (Portainer.io)
registry	minikube	disabled	minikube
registry-aliases	minikube	disabled	3rd party (unknown)
registry-creds	minikube	disabled	3rd party (UPMC Enterprises)
storage-provisioner	minikube	enabled ✓	minikube
storage-provisioner-gluster	minikube	disabled	3rd party (Gluster)
volumesnapshots	minikube	disabled	Kubernetes

Que vengan estos addons cargados es bastante cómodo ya que sólo tienes que activarlos para utilizarlos.

Un addon útil de [Minikube](#) para realizar ejercicios y aprender sobre Kubernetes es dashboard. El addon del panel de control (dashboard) proporciona una interfaz web que facilita la visualización y administración de los recursos de tu clúster.

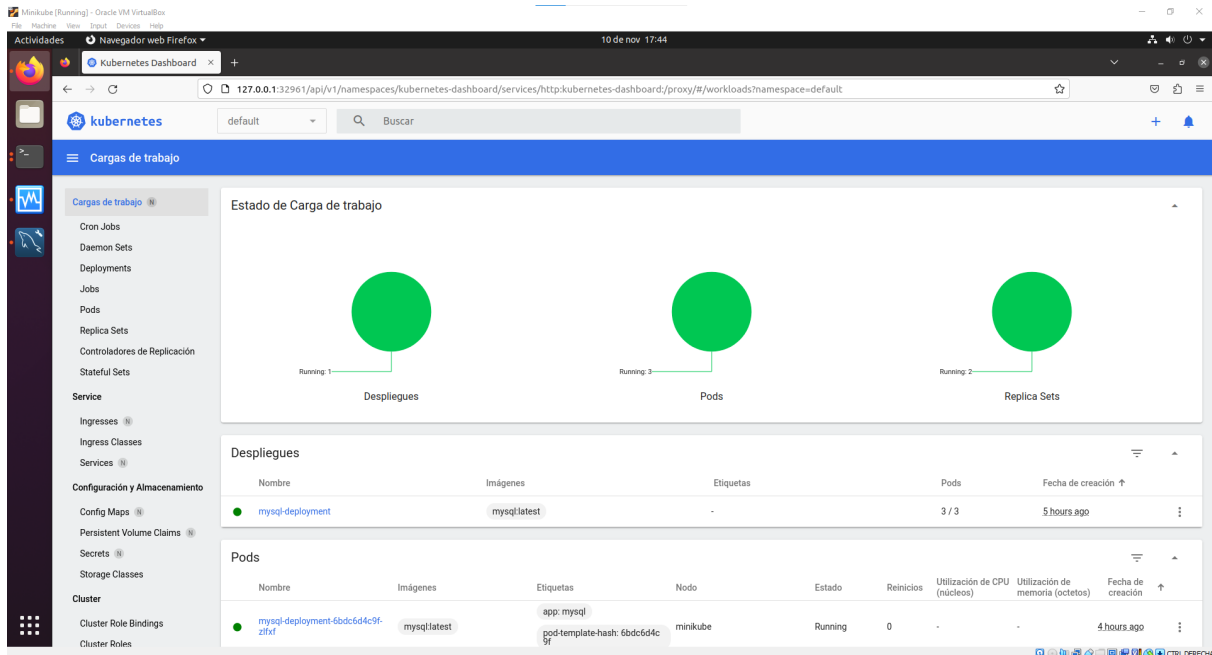
```
minikube dashboard
```



```
minkube@minkube-VirtualBox:~$ minikube dashboard
Habilitando dashboard
■ Using image docker.io/kubernetes/dashboard:v2.7.0
■ Using image docker.io/kubernetes/metrics-scraper:v1.0.8
💡 Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

😄 Verifying dashboard health ...
🔧 Launching proxy ...
> kubectl.sha256: 64 B / 64 B [-----] 100.00% ? p/s 0s
> kubectl: 46.98 MiB / 46.98 MiB [-----] 100.00% 76.02 MiB p/s 800ms
😄 Verifying proxy health ...
🔗 Opening http://127.0.0.1:35605/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```



### 3.3.1 Que es un POD?

Un Pod en Kubernetes es la unidad más pequeña desplegable y ejecutable en la plataforma. Representa un único proceso en un clúster de Kubernetes y puede contener uno o más contenedores. En resumen, un Pod es la envoltura más básica para una o más instancias de contenedores.

Los Pods se utilizan para implementar aplicaciones o servicios en Kubernetes. Un Pod puede contener uno o más contenedores que comparten el mismo espacio de red y almacenamiento. Esto significa que los contenedores en un mismo Pod pueden comunicarse entre sí a través de localhost y acceder a los mismos volúmenes de almacenamiento.

Los Pods son efímeros, lo que significa que pueden ser eliminados y reemplazados en cualquier momento. Esto es importante para la escalabilidad y la alta disponibilidad de las aplicaciones en Kubernetes. Los controladores de nivel superior, como los Despliegues, los **StatefulSets** y los **DemonSets**, se utilizan para administrar y garantizar la disponibilidad de los Pods en un clúster de Kubernetes.

Para crear un pod, por ejemplo, de un servidor web (nginx), con el nombre webserver1 y para ver los pods creados

```
kubectl run webservber1 --image=nginx

kubectl get pod
```

```
minikube@minikube-VirtualBox:~$ kubectl run webservber1 --image=nginx
pod/webservber1 created
minikube@minikube-VirtualBox:~$ kubectl get pod
NAME                                READY   STATUS             RESTARTS   AGE
hello-minikube-59d4768566-l66fd    1/1     Running            0          21h
webservber1                         0/1     ContainerCreating  0          12s
```

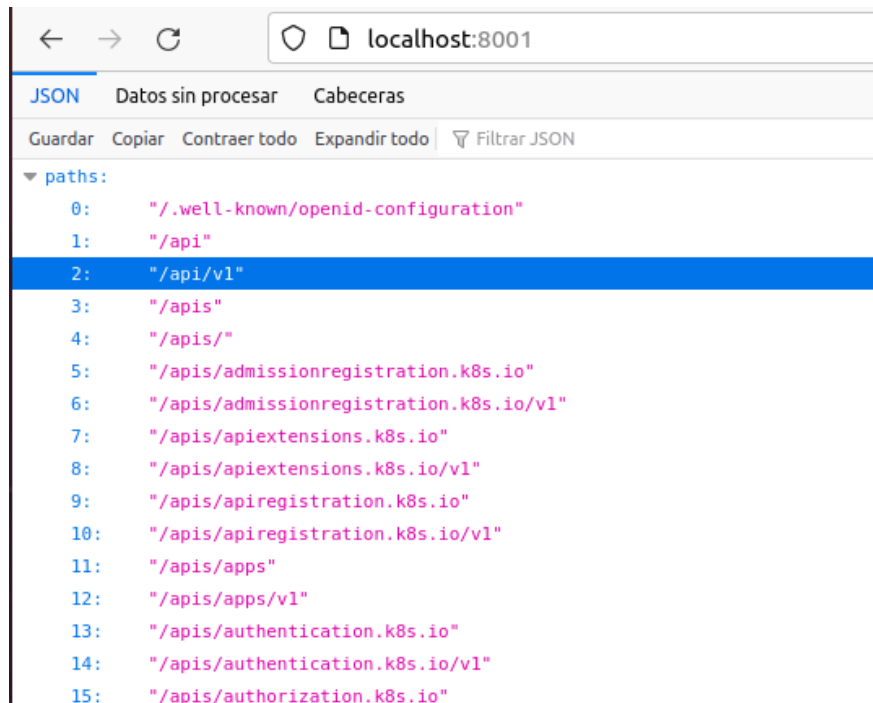
El comando `kubectl proxy` se utiliza para crear un servidor proxy local que permite acceder a los servicios de Kubernetes desde tu máquina local a través de una interfaz web. Esto es útil cuando deseas acceder a la API de Kubernetes, explorar servicios, consultar información sobre Pods, o interactuar con los recursos de Kubernetes desde una interfaz web amigable.

```
kubectl proxy
```

```
minikube@minikube-VirtualBox:~$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Después de ejecutar este comando, el servidor proxy se inicia en el puerto 8001 de tu máquina local. Puedes acceder a la interfaz web de Kubernetes en la siguiente URL:

```
http://localhost:8001/
```



Y con la siguiente URL ya podremos acceder al recurso

```
http://localhost:8001/api/v1/namespaces/default/pods/webserver1/proxy/
```

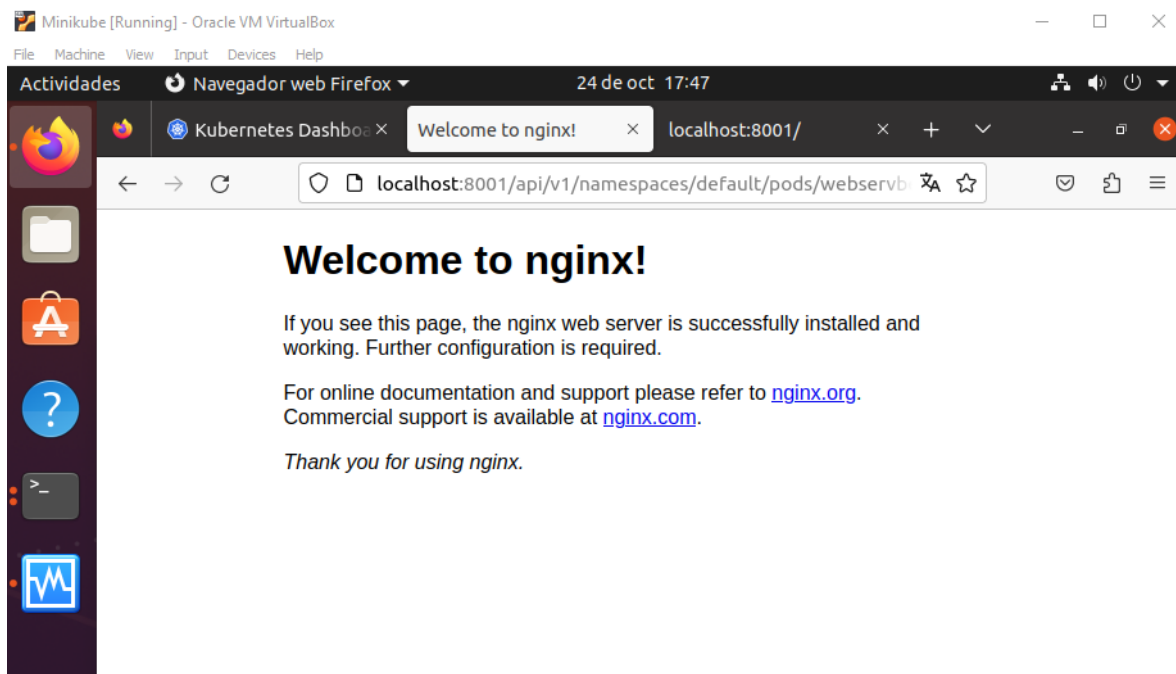
**http://localhost:8001**: Esta es la dirección base de tu servidor proxy local. Al ejecutar `kubectl proxy`, se inicia un servidor proxy en tu máquina local que escucha en el puerto 8001. La URL comienza con **http://localhost:8001** para acceder a este servidor proxy.

**api/v1**: Esta parte de la URL se refiere a la versión de la API de Kubernetes que estás utilizando. `api/v1` se refiere a la API de Kubernetes en su versión 1.

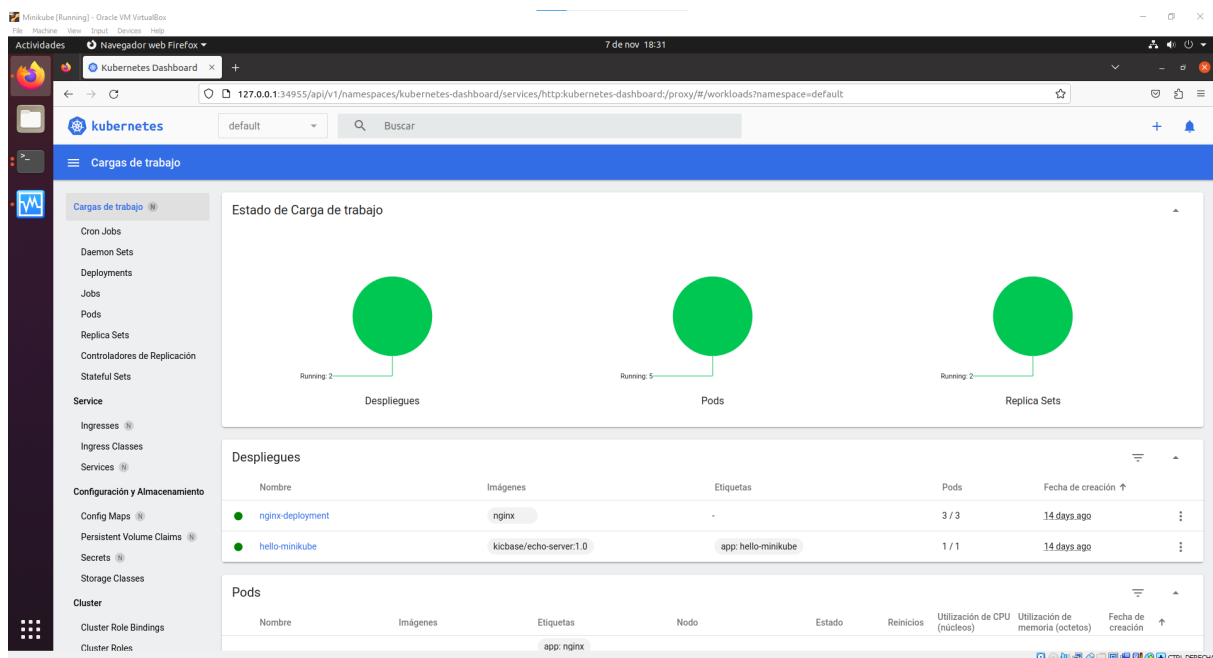
**namespaces/default**: Aquí especificas el espacio de nombres (namespace) al que pertenece el recurso que deseas acceder. En este caso, estás accediendo al espacio de nombres "default", que es el espacio de nombres por defecto en Kubernetes.

**pods/webserver1**: Esta parte de la URL se refiere al tipo de recurso que estás accediendo y al nombre del recurso. En este caso, estás accediendo a un Pod llamado "webserver1". El término "pods" indica que estás trabajando con un recurso de tipo Pod.

**proxy/**: Finalmente, `/proxy/` indica que deseas acceder al Pod "webserver1" a través de un proxy. Esta parte se utiliza para establecer la conexión y permitirte interactuar con el Pod a través de una interfaz web.



Así se vería los pods en el dashboard de **Minikube**



### 3.3.2 Que es un DEPLOYMENT?

Un Deployment en Kubernetes es un objeto que se utiliza para gestionar la implementación de aplicaciones en contenedores de una manera declarativa y escalable. Proporciona las siguientes funcionalidades clave:

**Declaratividad:** Un Deployment se define mediante un archivo YAML que describe el estado deseado de la aplicación, incluyendo la cantidad de réplicas y la imagen del contenedor que se debe usar. Kubernetes se encarga de llevar el estado actual al estado deseado, lo que significa que no necesitas preocuparte por las operaciones específicas para la implementación.

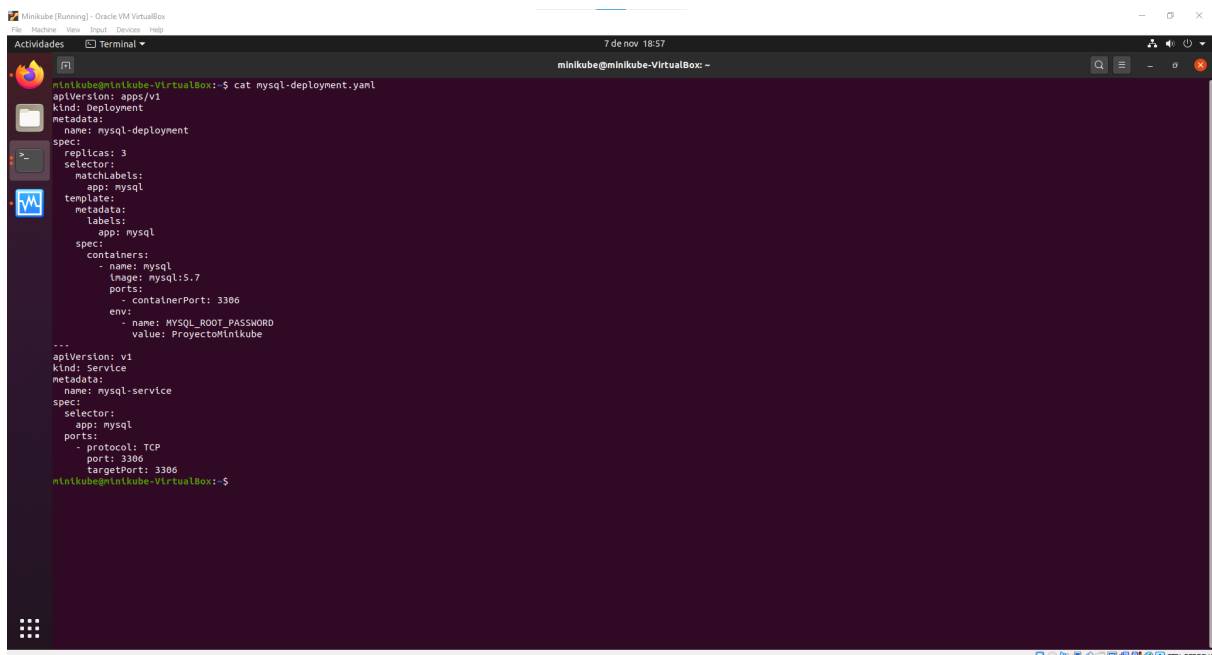
**Escalabilidad:** Puedes aumentar o disminuir el número de réplicas de tu aplicación de manera sencilla y automática. Esto permite escalar horizontalmente para manejar cargas de trabajo cambiantes.

**Actualizaciones y Rollbacks:** Los Deployments permiten realizar actualizaciones de tus aplicaciones de manera controlada. Puedes cambiar la imagen del contenedor o la configuración sin tiempo de inactividad y, si algo sale mal, realizar un rollback a una versión anterior.

**Alta Disponibilidad:** Los Deployments se encargan de mantener el número deseado de réplicas en funcionamiento, incluso en caso de fallos en nodos o contenedores. Si una réplica falla, Kubernetes automáticamente reemplaza la réplica con una nueva.

Para ver como funciona crearemos un clúster de MySQL.

Paso1: Creamos un deployment para desplegar servicios de una base de datos de mysql con 3 réplicas y las siguientes especificaciones.



```
minikube@minikube-VirtualBox:~$ cat mysql-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: ProyectoMinikube
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
minikube@minikube-VirtualBox:~$
```

Paso 2: Aplicar la Configuración a Minikube

Guardamos el archivo YAML en el sistema y luego lo aplicamos a Minikube usando el siguiente comando:

```
kubectl apply -f mysql-deployment.yaml
```

```
targetPort: 3306
minikube@minikube-VirtualBox:~$ kubectl apply -f mysql-deployment.yaml
deployment.apps/mysql-deployment created
service/mysql-service created
```

### Paso 3: Verificar el Estado del Deployment

Podemos verificar el estado del Deployment usando el siguiente comando:

```
kubectl get deployments
```

```
minikube@minikube-VirtualBox:~$ kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
mysql-deployment	0/3	3	0	52s

### 3.3.3 Ejemplo de despliegue de Wordpress + MySQL con servicios y volúmenes persistentes.

Primero vamos a instalar **Kustomize**, que es una herramienta de línea de comandos que permite personalizar y gestionar archivos de configuración de Kubernetes de manera declarativa. Esta herramienta fue desarrollada por Google y se utiliza para simplificar y modularizar la gestión de configuraciones en entornos Kubernetes.

La idea principal detrás de **Kustomize** es separar la definición de recursos de Kubernetes de las configuraciones específicas del entorno, lo que facilita la reutilización de configuraciones en diferentes contextos y entornos. En lugar de tener archivos YAML estáticos con todas las configuraciones incrustadas, **Kustomize** utiliza una estructura de directorios y archivos de configuración que se pueden componer y sobreponer de manera modular.

Descargamos el binario de Kustomize (reemplazamos la versión con la última disponible)

```
wget https://github.com/kubernetes-sigs/kustomize/releases/download/v5.2.1/kustomize_v5.2.1_linux_amd64.tar.gz
```

Extraemos el contenido del archivo tar.gz

```
tar xvf kustomize_v5.2.1_linux_amd64.tar.gz
```

Movemos el ejecutable a un directorio incluido en nuestro PATH (por ejemplo, /usr/local/bin/)

```
sudo mv kustomize /usr/local/bin/
```

Verificamos que Kustomize esté disponible

```
kustomize version
```

```
minikube@minikube-VirtualBox:~$ kustomize version
v5.2.1
```

Creamos los siguientes archivos .yaml dentro de un directorio (Proyecto)

```
mysql-secret.yaml
mysql-volume.yaml
mysql-deployment.yaml
mysql-service.yaml
wordpress-volume.yaml
wordpress-deployment.yaml
wordpress-service.yaml
```

## DEPLOYMENT

wordpress-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
```

```
spec:
  containers:
  - image: wordpress:5.7.1-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_HOST
      value: wordpress-mysql
    - name: WORDPRESS_DB_NAME
      value: wordpress
    - name: WORDPRESS_DB_USER
      value: root
    - name: WORDPRESS_DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql-pass
          key: password
    ports:
    - containerPort: 80
      name: wordpress
    volumeMounts:
    - name: wordpress-persistent-storage
      mountPath: /var/www/html
  volumes:
  - name: wordpress-persistent-storage
    persistentVolumeClaim:
      claimName: wp-pv-claim
```

## mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - args:
        - --default-authentication-plugin=mysql_native_password
        image: mysql:8.0.20
        name: mysql
        env:
        - name: MYSQL_DATABASE
          value: wordpress
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
```



```

    volumeMounts:
      - name: mysql-persistent-storage
        mountPath: /var/lib/mysql
    volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim

```

### 3.3.4 VOLUMENES

#### PersistentVolumeClaim (PVC)

Un PersistentVolumeClaim es un recurso de Kubernetes que permite a las aplicaciones solicitar almacenamiento persistente. Cuando las aplicaciones necesitan almacenar datos de manera persistente, como en bases de datos o sistemas de archivos, utilizan un PersistentVolumeClaim para solicitar el espacio de almacenamiento que necesitan.

En este caso el archivo `wordpress-volume.yaml`, el PVC se utiliza para que la aplicación WordPress tenga acceso a un volumen persistente de al menos 5 gigabytes con permisos de lectura y escritura. Este volumen persistente se utiliza para almacenar datos importantes, como los archivos del sitio web de WordPress y otros datos que necesitan persistir incluso cuando los contenedores se detienen o reinician.

#### `wordpress-volume.yaml`

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi

```

MySQL también necesita almacenamiento persistente para garantizar que los datos de la base de datos no se pierdan cuando los contenedores se reinician o actualizan.

#### `mysql-volume.yaml`

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:

```

```
- ReadWriteOnce
resources:
  requests:
    storage: 5Gi
```

### 3.3.5 SERVICIOS

En Kubernetes, los Servicios (Services) son una abstracción que define un conjunto lógico de pods y una política por la cual acceder a ellos. Los servicios permiten la exposición de aplicaciones y conjuntos de pods de manera uniforme, independientemente de la ubicación física de los pods en el clúster. El tipo de LoadBalancer es una de las formas en que se puede exponer un servicio para que sea accesible desde fuera del clúster

En Kubernetes, existen principalmente tres tipos de servicios:

#### **ClusterIP (Predeterminado):**

Accesible solo dentro del clúster.

Se utiliza para exponer servicios internos que solo deben ser accesibles desde otros pods dentro del clúster.

#### **NodePort:**

Expone el servicio en un puerto fijo en cada nodo del clúster.

Accesible desde fuera del clúster usando la IP del nodo y el puerto especificado.

#### **LoadBalancer:**

Proporciona una dirección IP externa (o nombre de dominio) y distribuye el tráfico entre los pods del servicio.

Utilizado para exponer servicios de manera escalable y accesible desde fuera del clúster.

Estos tipos permiten adaptar la exposición de servicios según las necesidades de conectividad y acceso requeridas por las aplicaciones desplegadas en Kubernetes.

wordpress-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

El servicio de MySQL se define como un ClusterIP en lugar de un LoadBalancer por razones de seguridad, eficiencia y alcance interno. Al limitar la accesibilidad a nivel interno del clúster, se reduce el riesgo de exposición no deseada y se optimizan los recursos. La conectividad eficiente entre aplicaciones y bases de datos se facilita internamente en el clúster, y cambiar a un servicio LoadBalancer es una opción si se requiere una exposición externa en el futuro. En resumen, el uso de un servicio ClusterIP para MySQL es una práctica común para aplicaciones que no necesitan ser accesibles directamente desde fuera del clúster.

mysql-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
```

### 3.3.6 SECRETS

En Kubernetes, los Secrets son objetos que se utilizan para almacenar información sensible, como contraseñas, claves de API o certificados. Están diseñados para permitir que esta información sensible se gestione de manera más segura que si se almacenara directamente en archivos de configuración o en variables de entorno.

mysql-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-pass
type: Opaque
stringData:
  password: omv
```

Todos estos .yaml los metemos en kustomization.yaml

```
resources:
  - mysql-secret.yaml
  - mysql-volume.yaml
```

- mysql-deployment.yaml
- mysql-service.yaml
- wordpress-volume.yaml
- wordpress-deployment.yaml
- wordpress-service.yaml

Y estando dentro del directorio de PROYECTO con un solo comando podremos ejecutar todos los archivos .yaml

```
kubectl apply -k ./
```

```
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl apply -k ./
secret/mysql-pass created
service/wordpress created
service/wordpress-mysql created
persistentvolumeclaim/mysql-pv-claim created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created
deployment.apps/wordpress-mysql created
```

Con un kubectl get all veremos lo que nos ha creado

```
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/wordpress-64d9d47969-495wx	1/1	Running	1 (3m10s ago)	6d23h
pod/wordpress-mysql-85966d8d8-kbb17	1/1	Running	5 (39s ago)	4d1h
pod/wordpress-mysql-85966d8d8-nj6c6	1/1	Running	2 (38s ago)	6d23h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	7d1h
service/wordpress	LoadBalancer	10.103.29.102	<pending>	80:32502/TCP	7d
service/wordpress-mysql	ClusterIP	None	<none>	3306/TCP	7d

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/wordpress	1/1	1	1	7d
deployment.apps/wordpress-mysql	2/2	2	2	6d23h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/wordpress-64d9d47969	1	1	1	7d
replicaset.apps/wordpress-mysql-85966d8d8	2	2	2	6d23h

El clúster Kubernetes actual consta de:

Un pod de WordPress en ejecución.

Dos pods relacionados con MySQL en ejecución.

Servicio kubernetes (ClusterIP)

Servicio wordpress (LoadBalancer)

Servicio wordpress-mysql (ClusterIP)

Despliegues y ReplicaSets asociados con WordPress y MySQL.

Con el comando `kubectl get pvc` veremos los volúmenes que no ha creado

```
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl get pvc
```

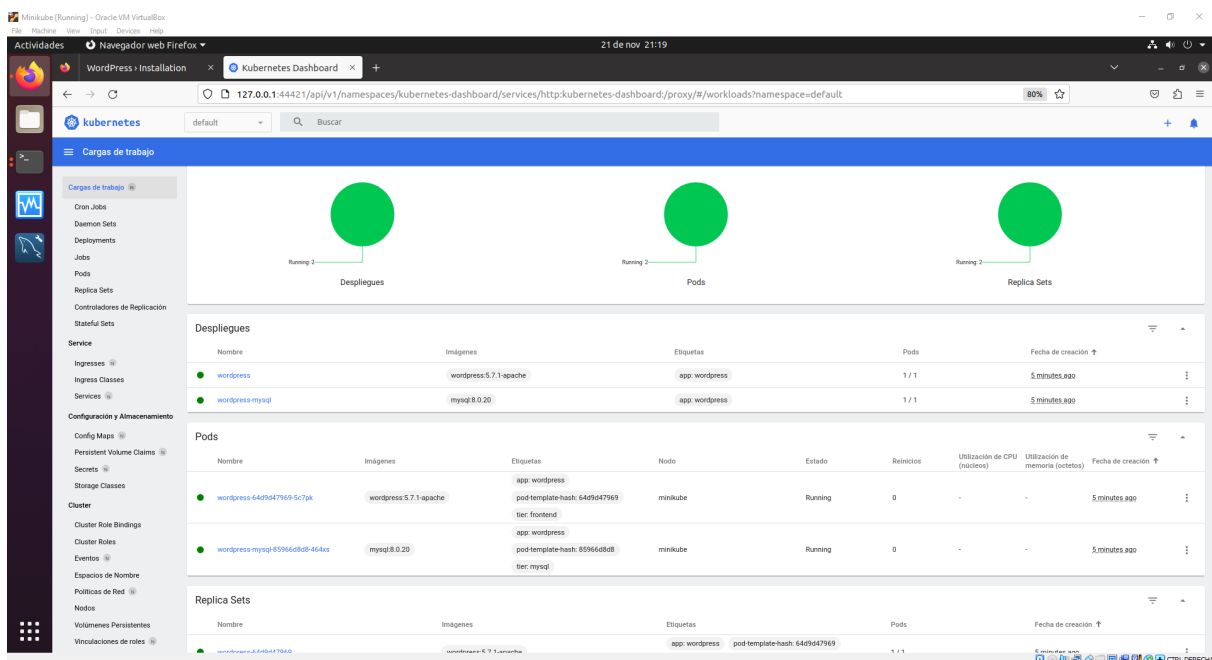
NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
mysql-pv-claim	Bound	pvc-426d1287-9496-4b43-927d-5c806e24d1d8	5Gi	RWO	standard	7d
wp-pv-claim	Bound	pvc-71bbfcc3-df47-4a27-a522-950799806838	5Gi	RWO	standard	7d

Y con `kubectl get secret`

```
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl get secret
```

NAME	TYPE	DATA	AGE
mysql-pass	Opaque	1	7d

Podemos ver con el complemento de Minikube dashboard más gráficamente

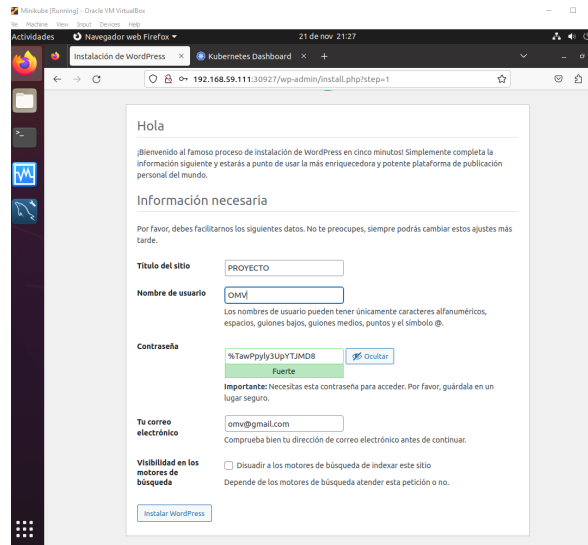
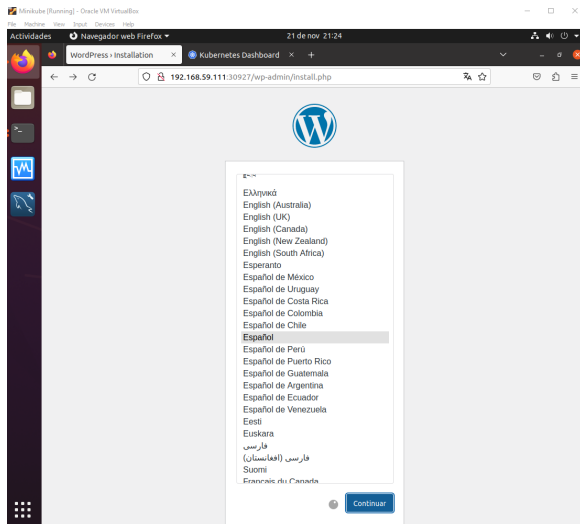


Para ver nuestro Wordpress en el navegador podemos pedirle a minikube que nos de la URL

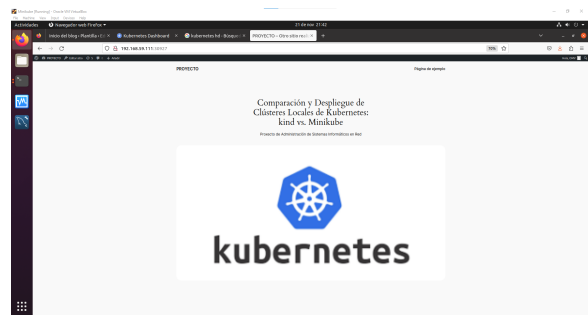
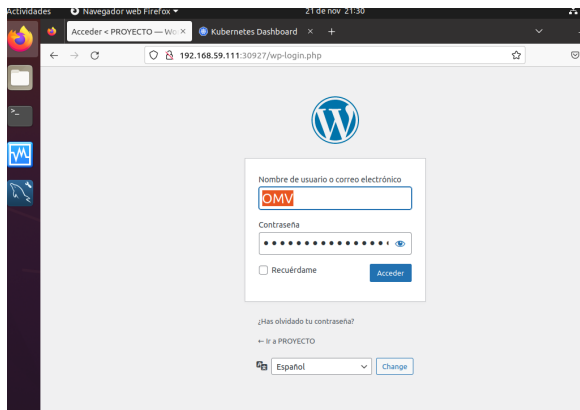
```
minikube@minikube-VirtualBox:~/PROYECTO$ minikube service wordpress --url
```

http://192.168.59.110:32502

Lo instalamos



Accedemos y creamos una pequeña pagina de portada



Ahora podemos probar a eliminar nuestro pod de MySQL.

Cuando eliminas un pod de un despliegue en Kubernetes, el controlador de replicación (ReplicaSet) asociado al despliegue detecta que el número de réplicas especificado no se cumple y toma medidas para restaurar el número deseado de pods. Esto es parte del comportamiento normal de Kubernetes para garantizar la disponibilidad y la capacidad de escalabilidad.

También por eso los nombres de los pods aquí no son importantes ya que pueden cambiar si existe alguna modificación.

```
minikube@minikube-VirtualBox:~$ kubectl delete pod/wordpress-mysql-85966d8d8-464xs
pod "wordpress-mysql-85966d8d8-464xs" deleted
minikube@minikube-VirtualBox:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/wordpress-64d9d47969-5c7pk	1/1	Running	0	18h
pod/wordpress-mysql-85966d8d8-vfmqr	1/1	Running	0	17s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	18h
service/wordpress	LoadBalancer	10.110.111.254	<pending>	80:30927/TCP	18h
service/wordpress-mysql	ClusterIP	None	<none>	3306/TCP	18h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/wordpress	1/1	1	1	18h
deployment.apps/wordpress-mysql	1/1	1	1	18h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/wordpress-64d9d47969	1	1	1	18h
replicaset.apps/wordpress-mysql-85966d8d8	1	1	1	18h

Ahora vamos a borrar el deployment de MySQL y vamos a nuestra página a ver que pasa:

```
minikube@minikube-VirtualBox:~$ kubectl delete deployment.apps/wordpress-mysql
deployment.apps "wordpress-mysql" deleted
minikube@minikube-VirtualBox:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/wordpress-64d9d47969-5c7pk	1/1	Running	0	18h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	18h
service/wordpress	LoadBalancer	10.110.111.254	<pending>	80:30927/TCP	18h
service/wordpress-mysql	ClusterIP	None	<none>	3306/TCP	18h

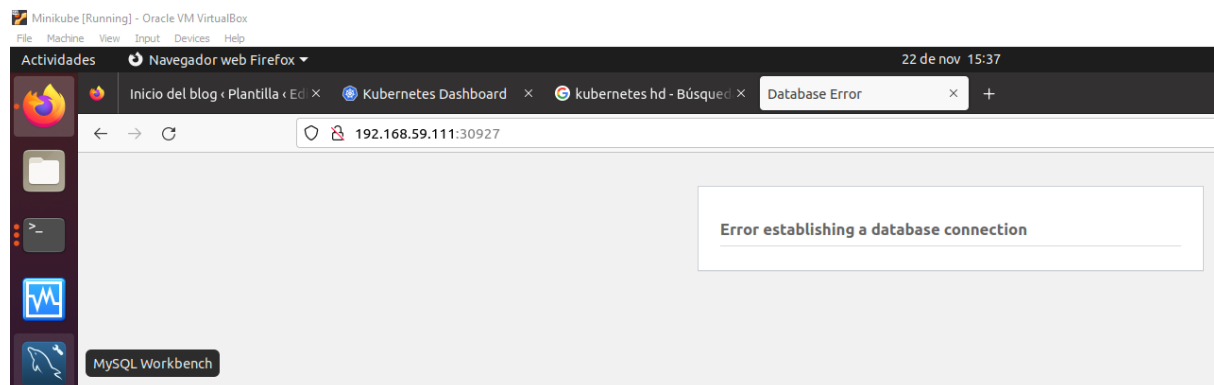
  

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/wordpress	1/1	1	1	18h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/wordpress-64d9d47969	1	1	1	18h

Vemos que nos da un error



Ahora sólo debemos aplicar el despliegue que teníamos de MySQL, y una vez que se crea podemos ir a nuestra página web y vemos que la portada esta igual que la creamos gracias a los volúmenes persistentes.

```
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl apply -f mysql-deployment.yaml
deployment.apps/wordpress-mysql created
minikube@minikube-VirtualBox:~/PROYECTO$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/wordpress-64d9d47969-5c7pk	1/1	Running	0	18h
pod/wordpress-mysql-85966d8d8-289mc	0/1	ContainerCreating	0	4s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	18h
service/wordpress	LoadBalancer	10.110.111.254	<pending>	80:30927/TCP	18h
service/wordpress-mysql	ClusterIP	None	<none>	3306/TCP	18h

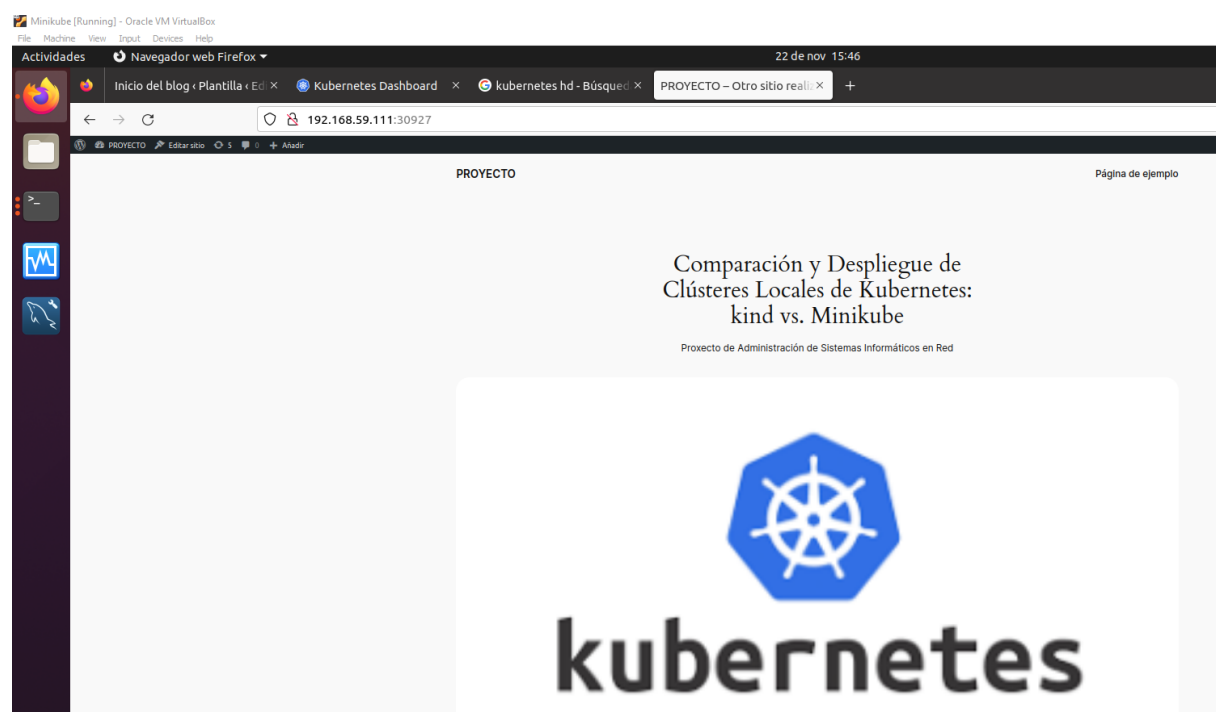
  

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/wordpress	1/1	1	1	18h
deployment.apps/wordpress-mysql	0/1	1	0	4s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/wordpress-64d9d47969	1	1	1	18h
replicaset.apps/wordpress-mysql-85966d8d8	1	1	0	4s

```
minikube@minikube-VirtualBox:~/PROYECTO$
```



Podemos también entrar en MySQL con la contraseña que pusimos en el SECRET y ver las base de datos que nos ha creado y las tablas y si aparece la imagen de la portada que hemos creado anteriormente.



```
kubectl exec -it wordpress-mysql-85966d8d8-7fr6v -- mysql -h localhost -u root -p 3306
```

```
minikube@minikube-VirtualBox:~$ kubectl exec -it wordpress-mysql-85966d8d8-289mc -- mysql -h localhost -u root -p -P 3306
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.20 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| wordpress |
+-----+
5 rows in set (0.06 sec)

mysql> use wordpress;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

```
mysql> SELECT * FROM wp_postmeta WHERE post_id = 13;
+-----+-----+-----+-----+
| meta_id | post_id | meta_key | meta_value |
+-----+-----+-----+-----+
| 16 | 13 | _wp_attached_file | 2023/11/Kubernetes.png |
+-----+-----+-----+-----+
```

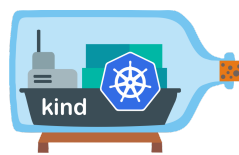
## 4.1 Creación y presentación del escenario Kind (Kubernetes In Docker)



El escenario para instalar **Kind** se compone de una máquina virtual linux, concretamente la máquina tendrá instalado Ubuntu 20.04.1 LTS, conectada a una red NAT con el exterior.

- Necesitaremos 2 CPU o más
- 8 GB de memoria libre
- 20 GB de espacio libre en el disco
- Conexión a Internet

## 4.2 Instalación de Kind



Igual que en el escenario de Minikube vamos a necesitar la instalación de **Kubectl** y **Docker**.

Para descargar **Kind** podemos utilizar el siguiente comando en la terminal, este comando utiliza curl para descargar el ejecutable de Kind desde el enlace proporcionado (<https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64>) y lo guarda localmente en el directorio actual con el nombre **Kind**.

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
```

```
oliver@oliver-VirtualBox:~$ curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100  97  100  97    0    0   563      0  --:--:-- --:--:-- --:--:--   563
0    0    0    0    0    0     0      0  --:--:-- --:--:-- --:--:--     0
100 6304k  100 6304k    0    0 6808k      0  --:--:-- --:--:-- --:--:--  28.1M
```

**Kind** utiliza contenedores Docker como nodos de clúster, eliminando la necesidad de máquinas virtuales. Cada nodo se ejecuta en su propio contenedor, lo que hace que la creación y destrucción de clústeres sea más liviana y rápida.

## 4.3 Inicio de Kind

Crear un clúster Kubernetes con **Kind** es un proceso sencillo y rápido, simplemente debemos ejecutar la siguiente sentencia:

```
kind create cluster --name Proyecto
```

```
omvkind@omvkind-VirtualBox:~/Proyecto$ kind create cluster --name proyecto
Creating cluster "proyecto" ...
 ✓ Ensuring node image (kindest/node:v1.21.1) 📁
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📄
 ✓ Starting control-plane 🚦
 ✓ Installing CNI 🖱️
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-proyecto"
You can now use your cluster with:

kubectl cluster-info --context kind-proyecto

Have a nice day! 🌻
omvkind@omvkind-VirtualBox:~/Proyecto$ kubectl cluster-info --context kind-proyecto
Kubernetes control plane is running at https://127.0.0.1:43485
CoreDNS is running at https://127.0.0.1:43485/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Pero podemos crear un archivo .yaml al que podemos configurar algunos aspectos que queremos que tenga nuestro clúster, por ejemplo cuantos nodos queremos que tenga.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: control-plane
- role: control-plane
- role: worker
- role: worker
```

y ejecutarlo con la siguiente sentencia:

```
kind create cluster --config kind-config.yaml --name proyecto2
```

```
omvkind@omvkind-VirtualBox:~/Proyecto$ kubectl get nodes -o wide
NAME                                STATUS    ROLES                  AGE      VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
proyecto2-control-plane             Ready     control-plane,master   8m15s    v1.21.1    172.18.0.6     <none>         Ubuntu 21.04         5.15.0-89-generic containerd://1.5.2
proyecto2-control-plane2            Ready     control-plane,master   7m44s    v1.21.1    172.18.0.5     <none>         Ubuntu 21.04         5.15.0-89-generic containerd://1.5.2
proyecto2-control-plane3            Ready     control-plane,master   6m48s    v1.21.1    172.18.0.7     <none>         Ubuntu 21.04         5.15.0-89-generic containerd://1.5.2
proyecto2-worker                    Ready     <none>                 6m14s    v1.21.1    172.18.0.4     <none>         Ubuntu 21.04         5.15.0-89-generic containerd://1.5.2
proyecto2-worker2                   Ready     <none>                 6m14s    v1.21.1    172.18.0.8     <none>         Ubuntu 21.04         5.15.0-89-generic containerd://1.5.2
```

### 4.3.1 Despliegue de Deployment y Daemonset en varios nodos

Ahora que tenemos ya el clúster con 5 nodos (3 masters y 2 workers), vamos a desplegar como en el ejercicio anterior un deployment, en este caso del servidor web nginx.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 6
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

y con su servicio en este caso de tipo NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
```

```
kubectl apply -f nginx-deployment.yaml nginx-deployment.yaml
```

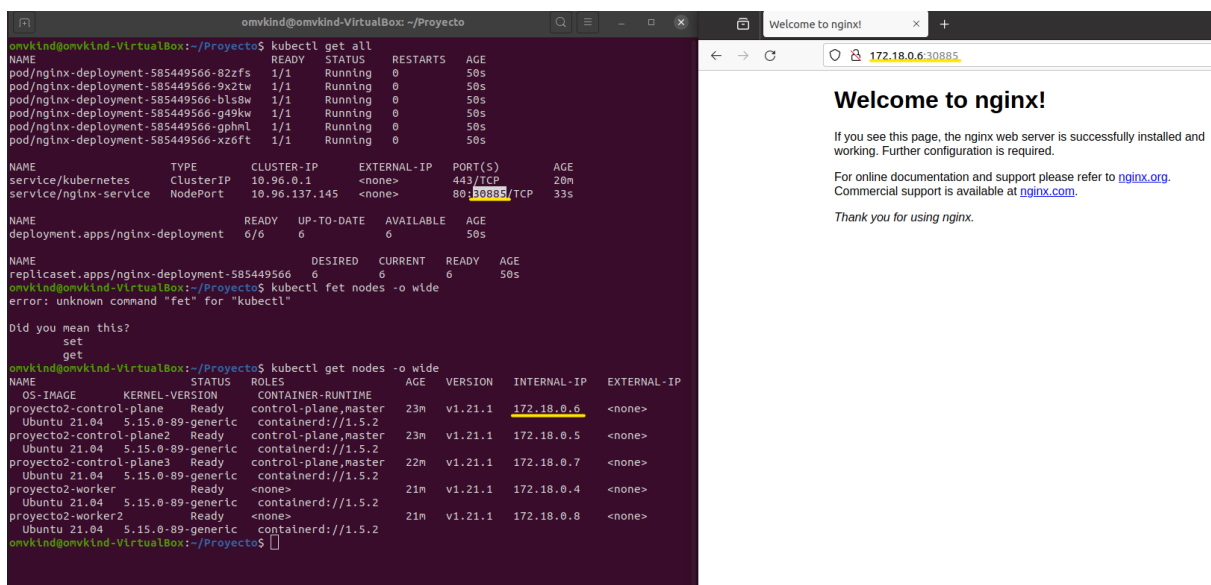
```
omvkind@omvkind-VirtualBox:~/Projecto$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/nginx-deployment-585449566-82zfs 1/1      Running   0           50s
pod/nginx-deployment-585449566-9x2tw 1/1      Running   0           50s
pod/nginx-deployment-585449566-bls8w 1/1      Running   0           50s
pod/nginx-deployment-585449566-g49kw 1/1      Running   0           50s
pod/nginx-deployment-585449566-gphml 1/1      Running   0           50s
pod/nginx-deployment-585449566-xz6ft 1/1      Running   0           50s

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/kubernetes                  ClusterIP     10.96.0.1     <none>         443/TCP          20m
service/nginx-service              NodePort      10.96.137.145 <none>         80:30885/TCP     33s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/nginx-deployment    6/6      6              6            50s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/nginx-deployment-585449566 6          6          6        50s
```

Vemos que creó las 6 réplicas y el servicio de nginx, ahora podemos ver si es accesible buscando en el explorador con la ip del nodo y el puerto del servicio de nginx



```
omvkind@omvkind-VirtualBox:~/Projecto$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/nginx-deployment-585449566-82zfs 1/1      Running   0           50s
pod/nginx-deployment-585449566-9x2tw 1/1      Running   0           50s
pod/nginx-deployment-585449566-bls8w 1/1      Running   0           50s
pod/nginx-deployment-585449566-g49kw 1/1      Running   0           50s
pod/nginx-deployment-585449566-gphml 1/1      Running   0           50s
pod/nginx-deployment-585449566-xz6ft 1/1      Running   0           50s

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/kubernetes                  ClusterIP     10.96.0.1     <none>         443/TCP          20m
service/nginx-service              NodePort      10.96.137.145 <none>         80:30885/TCP     33s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/nginx-deployment    6/6      6              6            50s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/nginx-deployment-585449566 6          6          6        50s

omvkind@omvkind-VirtualBox:~/Projecto$ kubectl get nodes -o wide
error: unknown command "fet" for "kubectl"

Did you mean this?
set
get

omvkind@omvkind-VirtualBox:~/Projecto$ kubectl get nodes -o wide
NAME                                STATUS    ROLES    AGE    VERSION    INTERNAL-IP    EXTERNAL-IP
projecto2-control-plane            Ready    control-plane,master 23m    v1.21.1    172.18.0.6     <none>
projecto2-control-plane2           Ready    control-plane,master 23m    v1.21.1    172.18.0.5     <none>
projecto2-control-plane3           Ready    control-plane,master 22m    v1.21.1    172.18.0.7     <none>
projecto2-worker                    Ready    <none>      21m    v1.21.1    172.18.0.4     <none>
projecto2-worker2                   Ready    <none>      21m    v1.21.1    172.18.0.8     <none>
```

### 4.3.2 ¿Qué es un DaemonSet?

Un DaemonSet es un tipo de controlador de replicación en Kubernetes que garantiza que una copia de un pod se ejecute en cada nodo del clúster. Cada nodo del clúster tiene su propia instancia del pod gestionado por el DaemonSet. La principal característica de un DaemonSet es que garantiza que siempre haya una instancia activa del pod en cada nodo, y se encarga automáticamente de crear o eliminar pods cuando se añaden o eliminan nodos en el clúster.

En el mismo clúster vamos a desplegar un Daemonset de MongoDB

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mongodb-daemonset
spec:
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule

      containers:
        - name: mongodb
          image: mongo:latest
          ports:
            - containerPort: 27017
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              value: admin
            - name: MONGO_INITDB_ROOT_PASSWORD
              value: password
```

y vemos con “kubectl get all” el resultado:

```
omvkind@omvkind-VirtualBox:~/Proyecto$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/mongodb-daemonset-7z9hj         1/1     Running   0           4m17s
pod/mongodb-daemonset-rjwkh         1/1     Running   0           4m17s
pod/mongodb-daemonset-vftgl         1/1     Running   0           4m17s
pod/mongodb-daemonset-xmdgd         1/1     Running   0           4m17s
pod/mongodb-daemonset-zqcdh         1/1     Running   0           4m17s
pod/nginx-deployment-585449566-82zfs 1/1     Running   0           26m
pod/nginx-deployment-585449566-9x2tw 1/1     Running   0           26m
pod/nginx-deployment-585449566-bls8w 1/1     Running   0           26m
pod/nginx-deployment-585449566-g49kw 1/1     Running   0           26m
pod/nginx-deployment-585449566-gphml 1/1     Running   0           26m
pod/nginx-deployment-585449566-xz6ft 1/1     Running   0           26m

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
service/kubernetes                  ClusterIP     10.96.0.1     <none>        443/TCP          46m
service/nginx-service               NodePort      10.96.137.145 <none>        80:30885/TCP     25m

NAME                                DESIRED    CURRENT    READY    UP-TO-DATE    AVAILABLE    NODE SELECTOR    AGE
daemonset.apps/mongodb-daemonset    5          5          0        5             0            <none>           4m19s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/nginx-deployment    6/6      6             6           26m

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/nginx-deployment-585449566 6          6          6        26m
```

Con el comando “kubectl get pods -o wide -l app=mongodb” podemos ver en que nodos están desplegados.

```
omvkind@omvkind-VirtualBox:~/Proyecto$ kubectl get pods -o wide -l app=mongodb
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
mongodb-daemonset-7z9hj	1/1	Running	0	9m34s	10.244.3.5	proyecto2-worker	<none>		<none>	
mongodb-daemonset-rjwkh	1/1	Running	0	9m34s	10.244.2.2	proyecto2-control-plane3	<none>		<none>	
mongodb-daemonset-vftgl	1/1	Running	0	9m34s	10.244.0.4	proyecto2-control-plane	<none>		<none>	
mongodb-daemonset-xmdgd	1/1	Running	0	9m34s	10.244.4.5	proyecto2-worker2	<none>		<none>	
mongodb-daemonset-zqcdh	1/1	Running	0	9m34s	10.244.1.3	proyecto2-control-plane2	<none>		<none>	

```
omvkind@omvkind-VirtualBox:~/Proyecto$
```

## 5. Conclusión

En conclusión, para propósitos educativos, Minikube se presenta como una herramienta valiosa, facilitando la enseñanza y aprendizaje de Kubernetes en entornos de desarrollo locales. Sus addons integrados ofrecen funcionalidades adicionales de manera sencilla, enriqueciendo la experiencia didáctica. Sin embargo, su limitación como clúster mononodo puede limitar la representación de escenarios más complejos que se asemejen a entornos de producción.

Por otro lado, Kubernetes in Docker (KinD) sobresale por su configurabilidad, permitiendo la creación de clústeres con múltiples nodos. Esta flexibilidad proporciona un entorno más parecido a un entorno de producción, ofreciendo una experiencia de aprendizaje más avanzada y realista. Aunque la configuración puede ser más detallada, la capacidad de simular un clúster con varios nodos en KinD aporta un valor significativo para comprender mejor la complejidad y dinámicas de un clúster Kubernetes en situaciones del mundo real. En última instancia, la elección entre Minikube y KinD dependerá de los objetivos específicos del aprendizaje y la representación deseada del entorno Kubernetes en el ámbito educativo.

## 6. Medios a utilizar

### 6.1 Recursos Materiales:

#### Equipo de Cómputo

**Procesador:** AMD Ryzen 3 4100 4-Core Processor 3.80 GHz

**Ram instalada:** 16,0 GB (15,9 GB utilizable)

**Gráfica:** NVIDIA GeForce GTX 750 Ti

**Sistema Operativo:** Windows 10 Pro 64bits

**Software de Virtualización:** VirtualBox Graphical User Interface Version 7.0.6 r155176 (Qt5.15.2)

**Entorno de Desarrollo:** Notion web, Notion app

**Conexión a Internet:** Para descargar software y acceder a recursos en línea relacionados con kind y Minikube.

## 6.2 Recursos Humanos:

Oliver Manteiga Vázquez

## 7. Tiempo de Ejecución:

Diagrama de Gantt

## 8. Presupuesto

En este apartado, se presenta el presupuesto estimado para la ejecución del proyecto, teniendo en cuenta el valioso recurso que es el tiempo. Cabe destacar que, al tratarse de un trabajo docente, se busca maximizar la eficiencia y minimizar cualquier gasto innecesario.

Detalles del Presupuesto:

Tiempo:

Se estima que el proyecto requerirá una inversión significativa de tiempo, recurso invaluable que, lamentablemente, no siempre es reconocido como se debería. La dedicación de horas a la planificación, ejecución y evaluación del trabajo es un aspecto crucial que se reflejará en la calidad del proyecto.

Recursos Didácticos:

Se destinará una parte del presupuesto a la adquisición de recursos didácticos que enriquezcan la experiencia de aprendizaje. Aunque se prioriza la eficiencia en la gestión del tiempo, se reconocen las inversiones necesarias para garantizar un entorno educativo enriquecedor.

Material de Apoyo:

<https://www.youtube.com/@PeladoNerd>

<https://kind.sigs.k8s.io/>

<https://minikube.sigs.k8s.io/>

## 9. Título

**Comparación y Despliegue de Clústeres Locales de Kubernetes: kind vs. Minikube**



## 10. CONTROL DE VERSIONS:

Versión	Data	Observacións	
1.0	20/09/2023	Idea del proyecto y objetivos a cumplir.	
2.0	27/09/2023	Proyecto aprobado. Inicio de la instalación de las máquinas virtuales, y sus correspondientes programas a comparar en el proyecto.	
3.0	08/12/2023	Terminar de documentar la memoria, creación de la exposición y del video demostrativo	

/8/Centro educativo

Código	Centro	Concello	Ano académico
15005397	I.E.S. Fernando Wirtz Suárez	A Coruña	2023-2024

Ciclo formativo

Código da familia profesional	Familia profesional	Código do ciclo formativo	Ciclo formativo	Grao	Réxime
FP16	Informática e comunicacións	CSIFC01	Administración de Sistemas Informáticos en Rede	Superior	Adultos

Módulo profesional e unidades formativas de menor duración (\*)

Código MP/UF	Nome
MP0374	Proxecto de Administración de Sistemas Informáticos en RedeEquivalencia en créditos ECTS: 5.

Profesorado responsable

Tutor	María Angeles Gómez Mosquera
-------	------------------------------

Alumno

Alumno	Oliver Manteiga Vázquez
--------	-------------------------

Datos do Proxecto

<b>Título</b>	Comparación y Despliegue de Clústeres Locales de Kubernetes: kind vs. Minikube
---------------	--