Actividad 0: Verificación del funcionamiento de Docker, Docker, Desktop, Minikube, Kind

.

Objetivos

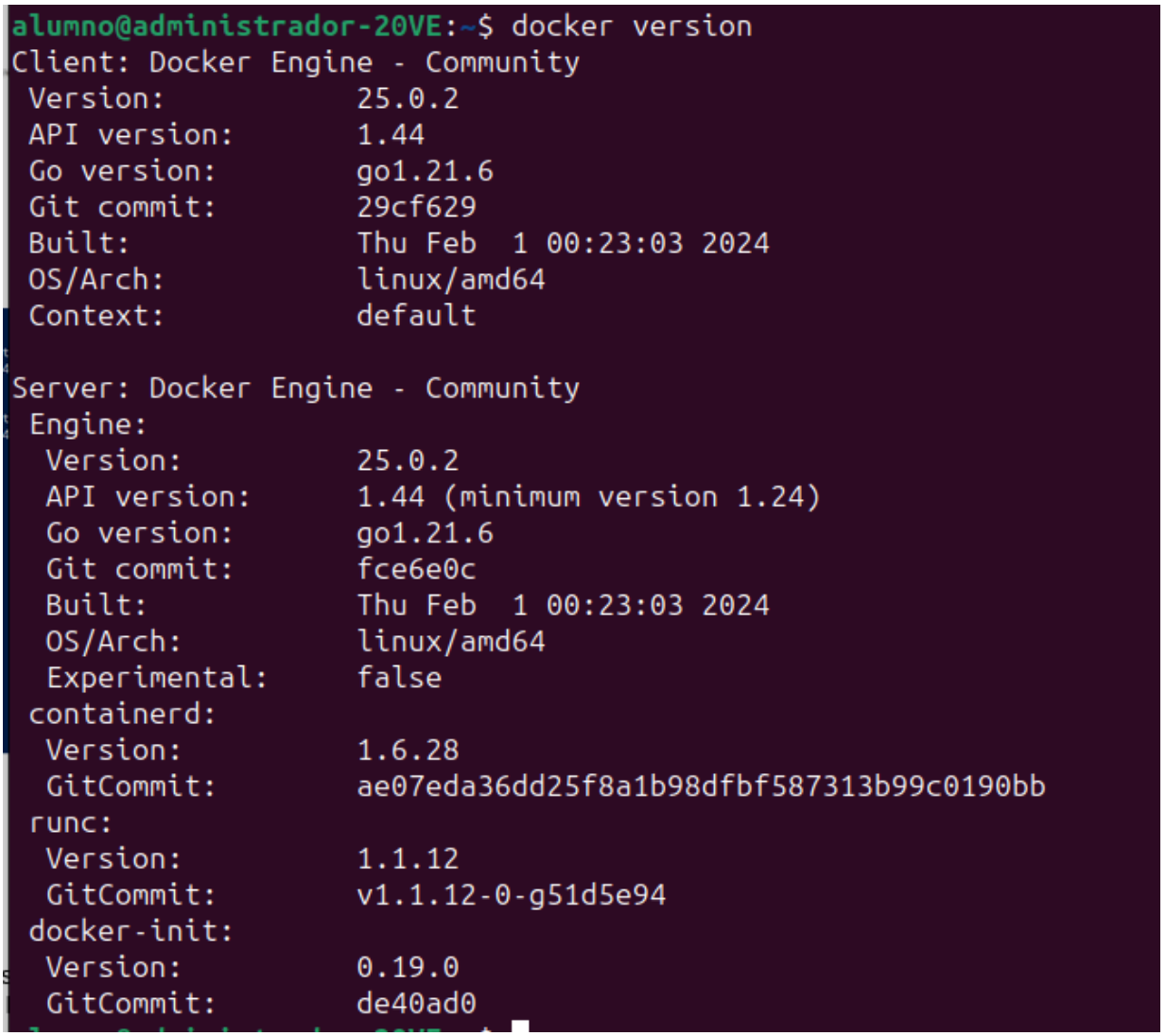
1. Comprender la virtualización y la contenerización: Entender las diferencias fundamentales entre la virtualización tradicional y la contenerización, y cómo Docker revoluciona el desarrollo y la implementación de aplicaciones mediante la creación de entornos ligeros y portátiles.
2. Familiarización con Docker y Docker Desktop: Aprender a instalar y configurar Docker y Docker Desktop en Linux y cómo pueden facilitar el desarrollo y la distribución de aplicaciones.
3. Explorar Kubernetes con Minikube y Kind: Entender los conceptos básicos de Kubernetes como sistema de orquestación de contenedores, y aprender a utilizar Minikube y Kind para crear y gestionar clústeres de Kubernetes en un entorno local. Esto incluye comprender la arquitectura de Kubernetes, los objetos de Kubernetes (pods, servicios, despliegues, etc.) y cómo se pueden manejar a través de estas herramientas.

Prueba de Docker Engine

1. Descarga e instale Docker Desktop: https://docs.docker.com/desktop/install/ubuntu/
2. Ahora que ha instalado Docker Desktop con éxito, probémoslo. Comenzaremos ejecutando un contenedor Docker simple directamente desde la línea de comando. Abra una ventana de Terminal y ejecute el siguiente comando:

$ docker version

```
C:\Users\alexandra>docker version
Client:
 Cloud integration: v1.0.35+desktop.11
 Version:           25.0.3
 API version:       1.44
 Go version:        go1.21.6
 Git commit:        4debf41
 Built:             Tue Feb  6 21:13:02 2024
 OS/Arch:           windows/amd64
 Context:           default

Server: Docker Desktop 4.28.0 (139021)
 Engine:
  Version:          25.0.3
  API version:      1.44 (minimum version 1.24)
  Go version:       go1.21.6
  Git commit:       f417435
  Built:            Tue Feb  6 21:14:25 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.28
  GitCommit:        ae07eda36dd25f8a1b98dfbf587313b99c0190bb
 runc:
  Version:          1.1.12
  GitCommit:        v1.1.12-0-g51d5e94
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```



3. Para ver si puede ejecutar contenedores, ingrese el siguiente comando en la ventana de Terminal y presione Enter:

   $ docker container run hello-world

```
C:\Users\alexandra>docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53641cd209a4fecfc68e21a99871ce8c6920b2e7502df0a20671c6fccc73a7c6
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

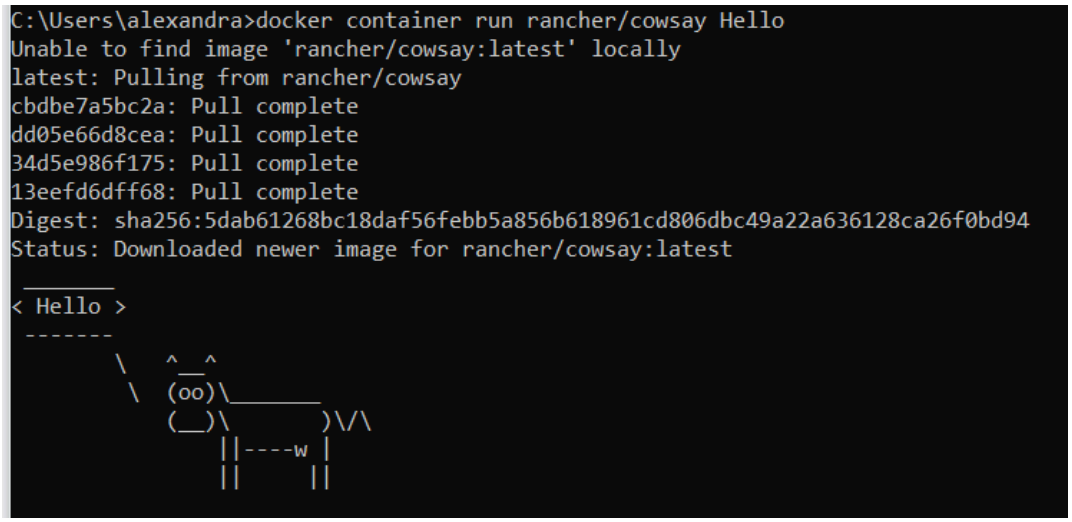
```
alumno@administrador-20VE:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53641cd209a4fecfc68e21a99871ce8c6920b2e7502df0a20671c6fccc73a7c6
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
```

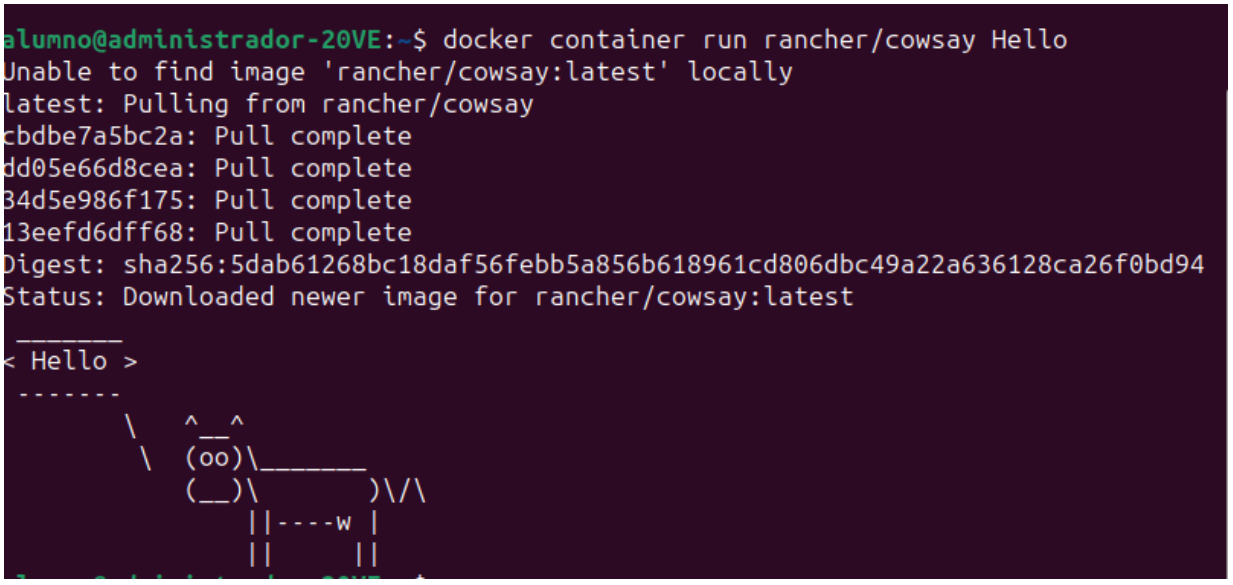
Si lee atentamente el resultado anterior, habrá notado que Docker no encontró una imagen llamada hello-world:latest y, por lo tanto, decidió descargarla desde un registro de imágenes de Docker. Una vez descargado, Docker Engine creó un contenedor a partir de la imagen y lo ejecutó. La aplicación se ejecuta dentro del contenedor y luego genera todo el texto, comenzando con Hello from Docker!

Esta es una prueba de que Docker está instalado y funcionando correctamente en su máquina.

4. Probemos con otra imagen de prueba divertida que normalmente se usa para verificar la instalación de Docker. Ejecute el siguiente comando:

$ docker container run rancher/cowsay Hello





Genial: hemos confirmado que Docker Engine funciona en nuestra computadora local. Ahora, asegurémonos de que lo mismo ocurre con Docker Desktop.

Observación : para crear el grupo de Docker y agregar su usuario se requiere los siguientes pasos:

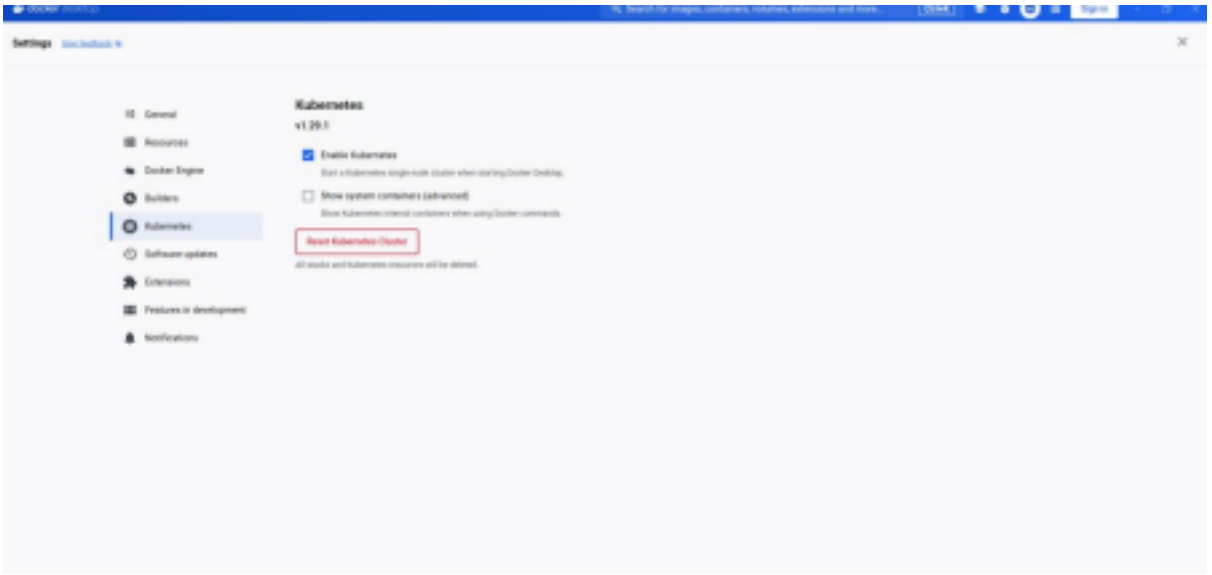
1. Crear el grupo docker: sudo groupadd docker
2. Agregar tu usuario a tu grupo docker: sudo usermod -aG docker $USER También revisar: https://docs.docker.com/engine/security/rootless/
3. Cierre sesión y vuelva a iniciarla para que se reevalúe su permanencia en el grupo.
4. También puede ejecutar el siguiente comando para activar los cambios en los grupos: newgrp docker

5. Verifica que puedes correr comandos docker sin usar sudo.

Habilitar Kubernetes en Docker Desktop

Instalación de Docker Desktop: https://docs.docker.com/desktop/install/linux-install/

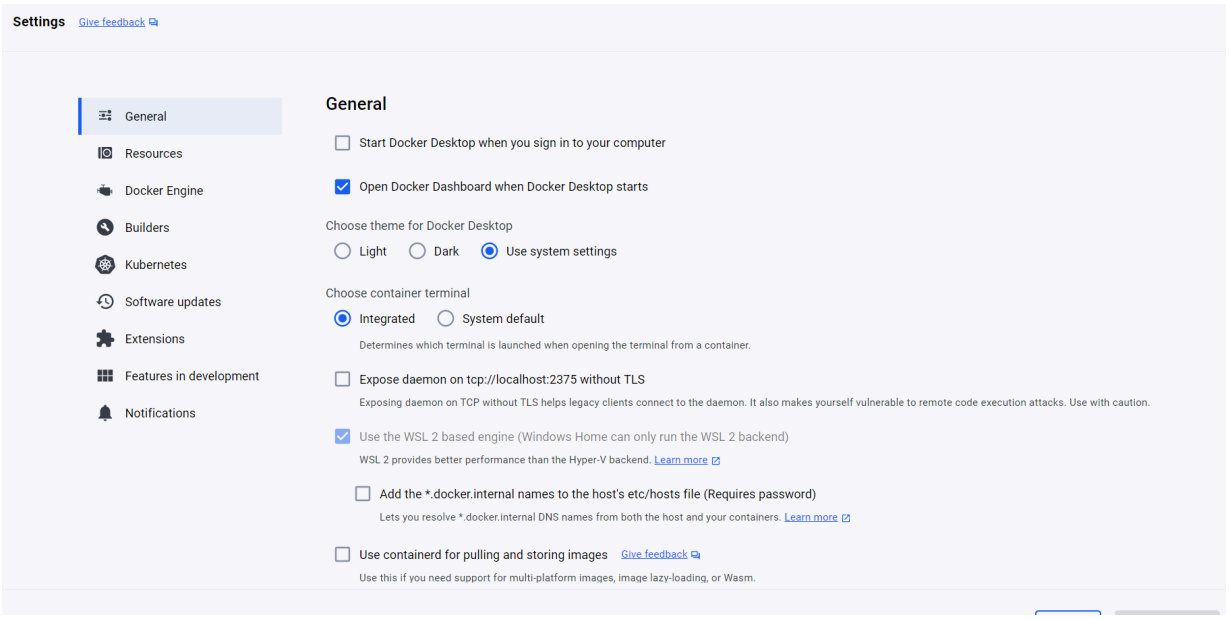
Docker Desktop viene con soporte integrado para Kubernetes.



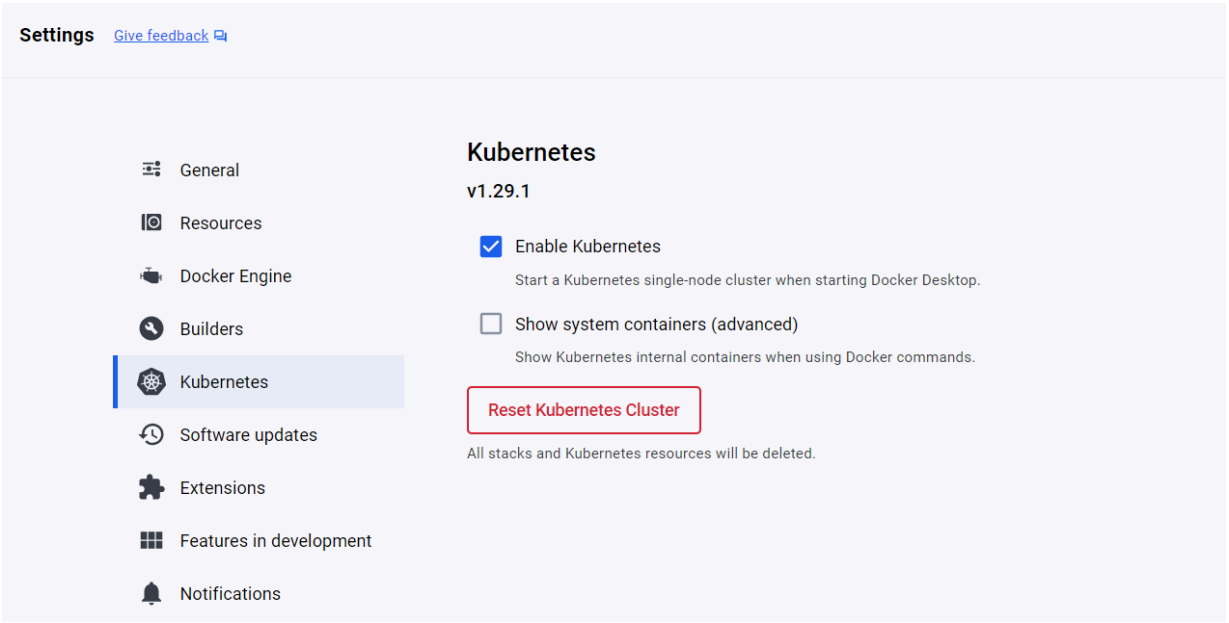
¿Qué es Kubernetes?
Kubernetes es una poderosa plataforma para automatizar la implementación, el escalado y la gestión de aplicaciones en contenedores. Ya seas desarrollador, ingeniero de DevOps o administrador de sistemas, Kubernetes proporciona las herramientas y abstracciones que necesita para administrar sus contenedores y aplicaciones de manera escalable y eficiente. Este soporte está desactivado de forma predeterminada. Pero no te preocupes: es muy fácil de activar:

1. Abra el panel de **Docker Desktop.**

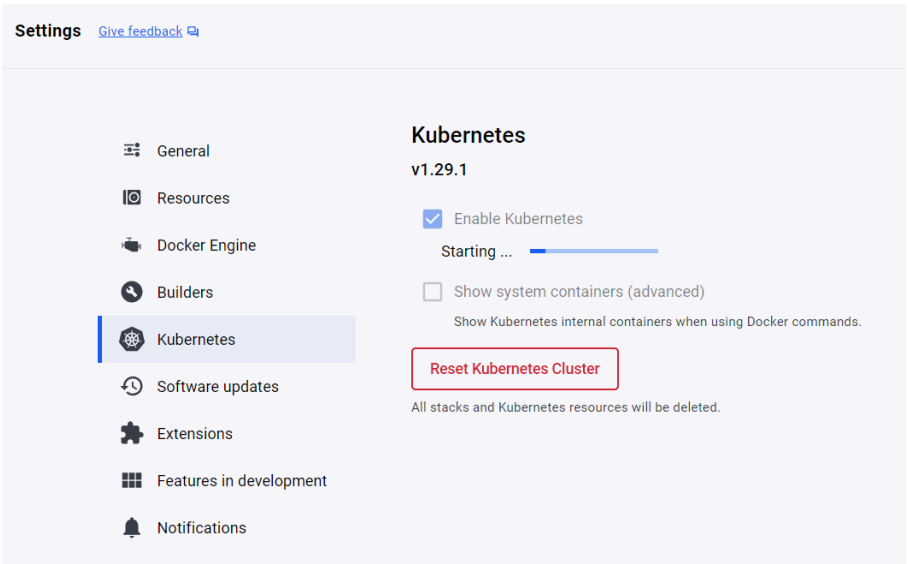
2. En la esquina superior izquierda, seleccione el ícono de la rueda dentada. Esto abrirá la página de configuración (setting).



3. En el lado izquierdo, seleccione la pestaña Kubernetes y luego marque la casilla Enable Kubernetes



4. Haga clic en el botón Apply & restart.



Ahora, deberá tener paciencia ya que Docker está descargando toda la infraestructura de soporte y luego inicia Kubernetes.
Una vez que Docker se haya reiniciado, estará listo para usar Kubernetes.

Probando minikube y kubectl

Descarga e instala y minikube https://minikube.sigs.k8s.io/docs/start/ y kubectl https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/ y la opción Install using native package management

Ahora escribe minikube start para iniciar el cluster por defecto . La primera vez que haga esto, llevará un tiempo ya que minikube necesita descargar todos los binarios de Kubernetes. Cuando termine, la última línea del resultado en su pantalla debería ser algo como esto:

```
>> minikube start --driver=docker
>>
W0401 06:57:15.664026    6248 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\alexandra\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f41
33a33f0688f\meta.json: El sistema no puede encontrar la ruta especificada.
* Removed all traces of the "minikube" cluster.
W0401 06:57:19.881778    2568 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\alexandra\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f41
33a33f0688f\meta.json: El sistema no puede encontrar la ruta especificada.
* minikube v1.32.0 en Microsoft Windows 10 Home Single Language 10.0.19045.4170 Build 19045.4170
* Using the docker driver based on user configuration
* Using Docker Desktop driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
    > gcr.io/k8s-minikube/kicbase...:  453.90 MiB / 453.90 MiB  100.00% 2.44 Mi
* Creating docker container (CPUs=2, Memory=2200MB) ...
! Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 2.462088s
* Restarting the docker service may improve performance.
* Preparando Kubernetes v1.28.3 en Docker 24.0.7...
  - Generando certificados y llaves
  - Iniciando plano de control
  - Configurando reglas RBAC...
* Configurando CNI bridge CNI ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

```
alumno@administrador-20VE:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 89.3M  100 89.3M    0     0  7631k      0  0:00:11  0:00:11 --:--:--  9.8M
[sudo] contraseña para alumno:
alumno@administrador-20VE:~$ minikube start
😄  minikube v1.32.0 en Ubuntu 23.04
✨  Controlador docker seleccionado automáticamente. Otras opciones: none, ssh
📌  Using Docker driver with root privileges
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
💾  Descargando Kubernetes v1.28.3 ...
    > preloaded-images-k8s-v18-v1...:  403.35 MiB / 403.35 MiB  100.00% 6.53 Mi
    > gcr.io/k8s-minikube/kicbase...:  453.90 MiB / 453.90 MiB  100.00% 2.35 Mi
🔥  Creating docker container (CPUs=2, Memory=3900MB) ...
🐳  Preparando Kubernetes v1.28.3 en Docker 24.0.7...
    ▪ 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: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Empecemos. Siga estos pasos cuidadosamente:

1. Intentemos acceder a nuestro clúster usando kubectl. Primero, debemos asegurarnos de tener seleccionado el contexto correcto para kubectl. Si anteriormente instaló Docker Desktop y ahora minikube, puede usar el siguiente comando:

$ kubectl config get-contexts

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-CB286$ kubectl config get-contexts
CURRENT   NAME             CLUSTER          AUTHINFO         NAMESPACE
          docker-desktop   docker-desktop   docker-desktop
*         minikube         minikube         minikube         default
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-CB286$
```

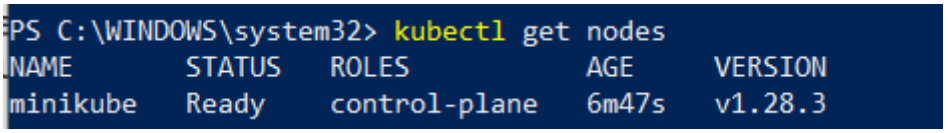
```
PS C:\WINDOWS\system32>  kubectl config get-contexts
CURRENT   NAME             CLUSTER          AUTHINFO         NAMESPACE
          docker-desktop   docker-desktop   docker-desktop
*         minikube         minikube         minikube         default
```

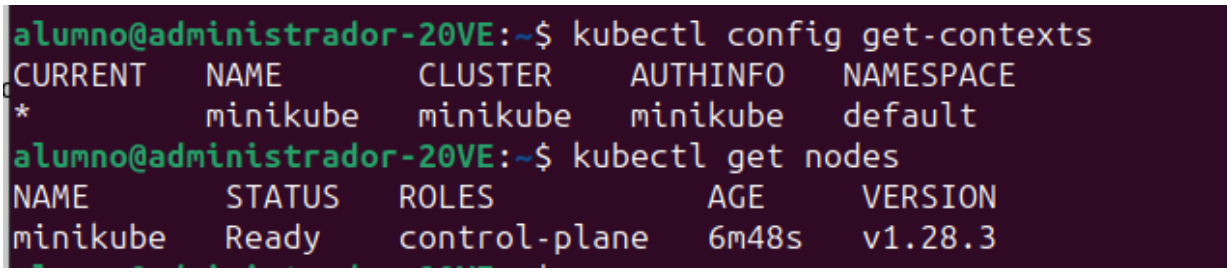
```
alumno@administrador-20VE:~$ kubectl config get-contexts
CURRENT   NAME       CLUSTER    AUTHINFO   NAMESPACE
*         minikube   minikube   minikube   default
```

El asterisco al lado del contexto llamado minikube nos dice que este es el contexto actual. Así, al usar kubectl, trabajaremos con el nuevo cluster creado por minikube.

2. Ahora veamos cuántos nodos tiene nuestro cluster con este comando:

$kubectl get nodes





Deberías obtener algo similar a esto. Tenga en cuenta que la versión mostrada podría diferir en su caso:

Aquí tenemos un clúster de un solo nodo. El papel del nodo es el del plano de control, lo que significa que es un nodo maestro. Un clúster de Kubernetes típico consta de unos pocos nodos maestros y muchos nodos trabajadores. La versión de Kubernetes con la que estamos trabajando aquí es la v1.28.3.

3. Ahora, intentemos ejecutar algo en este clúster. Usaremos Nginx, un servidor web popular para esto. Utiliza el archivo .yaml, que acompaña a la actividad que vamos a utilizar para esta prueba:

Abra una nueva ventana de Terminal y crea un pod que ejecute Nginx con el siguiente comando:

$ kubectl apply -f nginx.yaml

kubectl apply -f /home/alumno/Descargas/nginx.yaml

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kubectl apply -f nginx.yaml
pod/nginx created
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

Deberías ver este resultado:

pod/nginx created

```
alumno@administrador-20VE:~$ kubectl apply -f /home/alumno/Descargas/nginx.yaml
pod/nginx created
alumno@administrador-20VE:~$
```

4. Podemos verificar si el pod se está ejecutando con kubectl:

$ kubectl get pods

Deberíamos ver esto:

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          2m9s
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

```
alumno@administrador-20VE:~$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          2m4s
alumno@administrador-20VE:~$
```

Esto indica que tenemos 1 pod con Nginx ejecutándose y que se ha reiniciado 0 veces.

5. Para acceder al servidor Nginx, necesitamos exponer la aplicación que se ejecuta en el pod con el siguiente comando:

$ kubectl expose pod nginx --type=NodePort --port=80

```
service/nginx exposed
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

Esta es la única forma en que podemos acceder a Nginx desde nuestra computadora portátil, por ejemplo, a través de un navegador. Con el comando anterior, estamos creando un servicio de Kubernetes, como se indica en el resultado generado para el comando:

service/nginx exposed

```
alumno@administrador-20VE:~$ kubectl expose pod nginx --type=NodePort --port=80
service/nginx exposed
alumno@administrador-20VE:~$
```

6. Podemos usar kubectl para enumerar todos los servicios definidos en nuestro clúster: $ kubectl get services

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        18h
nginx        NodePort    10.108.34.206   <none>        80:30432/TCP   99s
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

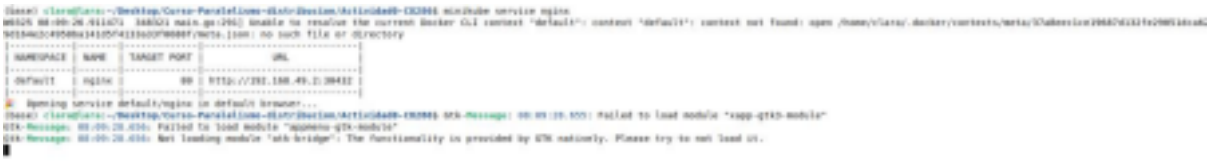
RESULTADO MIO

```
alumno@administrador-20VE:~$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        78m
nginx        NodePort    10.106.68.62    <none>        80:31219/TCP   106s
alumno@administrador-20VE:~$
```

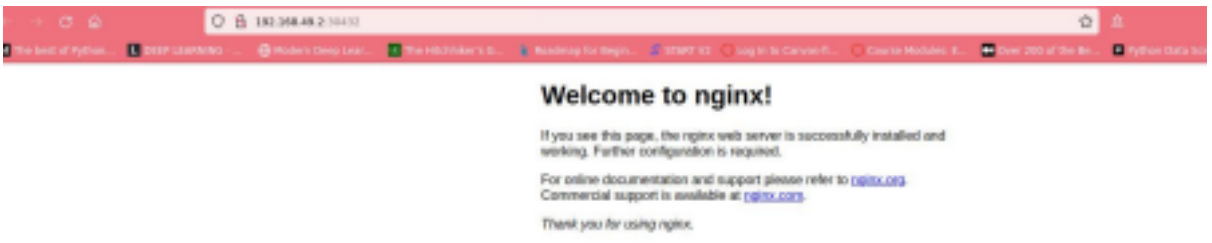
En el resultado anterior, podemos ver el segundo servicio llamado Nginx, que acabamos de crear. El servicio es del tipo NodePort; El puerto 80 del pod se había asignado al puerto 30432 del nodo del clúster de nuestro clúster de Kubernetes en minikube.

7. Ahora, podemos usar minikube para crear un túnel hacia nuestro clúster y abrir un navegador con la URL correcta para acceder al servidor web Nginx. Utilice este comando:

$ minikube service nginx

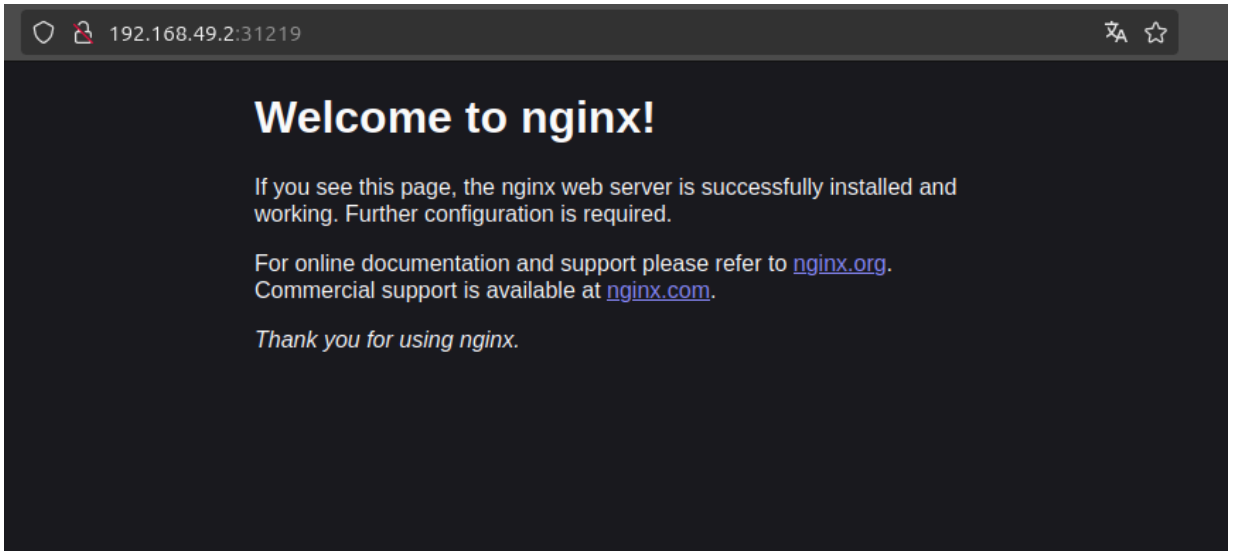


El resultado en su ventana de Terminal será el siguiente:



RESULTADO MIO

```
alumno@administrador-20VE:~$ minikube service nginx
|-----------|-------|-------------|--------------------------|
| NAMESPACE | NAME  | TARGET PORT |            URL           |
|-----------|-------|-------------|--------------------------|
| default   | nginx |          80 | http://192.168.49.2:31219 |
|-----------|-------|-------------|--------------------------|
🎉  Opening service default/nginx in default browser...
alumno@administrador-20VE:~$ update.go:85: cannot change mount namespace accordi
ng to change mount (/run/user/1001/doc/by-app/snap.firefox /run/user/1001/doc no
ne bind,rw,x-snapd.ignore-missing 0 0): cannot inspect "/run/user/1001/doc": lst
at /run/user/1001/doc: permission denied
Gtk-Message: 13:19:13.766: Not loading module "atk-bridge": The functionality is
 provided by GTK natively. Please try to not load it.
[104611, Main Thread] WARNING: Failed to read portal settings: GDBus.Error:org.f
reedesktop.DBus.Error.AccessDenied: Portal operation not allowed: Unable to open
 /proc/104611/root: 'glib warning', file /build/firefox/parts/firefox/build/tool
kit/xre/nsSigHandlers.cpp:187

(firefox:104611): Gdk-WARNING **: 13:19:13.769: Failed to read portal settings:
GDBus.Error:org.freedesktop.DBus.Error.AccessDenied: Portal operation not allowe
d: Unable to open /proc/104611/root
```



El resultado anterior muestra que minikube creó un túnel para el servicio nginx que escucha en el puerto del nodo 30432 que está en nuestra computadora portátil.

Hemos ejecutado y accedido con éxito a un servidor web Nginx en nuestro pequeño clúster de Kubernetes de un solo nodo en minikube! Una vez que hayas terminado de jugar, es hora de limpiar:

- Detenga el túnel hacia el clúster presionando Ctrl + C dentro de la ventana de Terminal.
- Elimine el servicio nginx y el pod en el clúster: $ kubectl delete service nginx $ kubectl delete pod nginx
- Detenga el clúster con el siguiente comando: minikube stop
- Deberías ver esto:

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ minikube stop
W0325 08:15:48.445689  372293 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open /home/clara/.docker/contexts/meta/37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0680f/meta.json: no such file or directory
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
```

RESULTADO MIO

```
alumno@administrador-20VE:~$ kubectl delete pod nginx
pod "nginx" deleted
alumno@administrador-20VE:~$ minikube stop
✋  Stopping node "minikube"  ...
🛑  Apagando "minikube" mediante SSH...
```

Ejercicios

A veces, probar con un clúster de un solo nodo no es suficiente. minikube lo resuelve. Siga estas instrucciones para crear un verdadero clúster de Kubernetes de múltiples nodos en minikube:

1. Si queremos trabajar con un clúster que consta de varios nodos en minikube, podemos usar este comando:

$ minikube start --nodes 3 –p demo

RESULTADO MIO

```
alumno@administrador-20VE:~$ minikube start --nodes 3 -p demo
😄  minikube v1.32.0 en Ubuntu 23.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparando Kubernetes v1.28.3 en Docker 24.0.7...
🔗  Configurando CNI bridge CNI ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Complementos habilitados: storage-provisioner, default-storageclass
❗  The cluster minikube already exists which means the --nodes parameter will b
e ignored. Use "minikube node add" to add nodes to an existing cluster.
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" name
space by default
alumno@administrador-20VE:~$
```

El comando anterior crea un clúster con tres nodos y lo llama demo.

```
alumno@administrador-20VE:~$ minikube node add
😄  Agregando el nodo m02 al cluster minikube.
❗  Cluster was created without any CNI, adding a node to it might cause broken
networking.
👍  Starting worker node minikube-m02 in cluster minikube
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparando Kubernetes v1.28.3 en Docker 24.0.7...
🔎  Verifying Kubernetes components...
🏄  Successfully added m02 to minikube!
alumno@administrador-20VE:~$ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
minikube       Ready    control-plane   97m   v1.28.3
minikube-m02   Ready    <none>          44s   v1.28.3
alumno@administrador-20VE:~$ minikube node add
😄  Agregando el nodo m03 al cluster minikube.
👍  Starting worker node minikube-m03 in cluster minikube
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparando Kubernetes v1.28.3 en Docker 24.0.7...
🔎  Verifying Kubernetes components...
🏄  Successfully added m03 to minikube!
alumno@administrador-20VE:~$ kubectl get nodes
NAME           STATUS   ROLES           AGE     VERSION
minikube       Ready    control-plane   99m     v1.28.3
minikube-m02   Ready    <none>          2m48s   v1.28.3
minikube-m03   Ready    <none>          23s     v1.28.3
```

2. Utilice kubectl para enumerar todos los nodos de su clúster:

$ kubectl get

```
alumno@administrador-20VE:~$ kubectl get nodes
NAME           STATUS   ROLES           AGE     VERSION
minikube       Ready    control-plane   99m     v1.28.3
minikube-m02   Ready    <none>          2m48s   v1.28.3
minikube-m03   Ready    <none>          23s     v1.28.3
alumno@administrador-20VE:~$
```

Tenemos un clúster de 3 nodos donde el nodo **demo** es un nodo maestro y los dos nodos restantes son nodos de trabajo.

3. No vamos a continuar con este ejemplo aquí, así que use el siguiente comando para detener el clúster:

$ minikube stop -p demo

```
alumno@administrador-20VE:~$ minikube stop -p demo
🤷  Profile "demo" not found. Run "minikube profile list" to view all profiles.
👉  To start a cluster, run: "minikube start -p demo"
```

4. Elimine todos los clústeres de su sistema con este comando:

$ minikube delete --all

```
alumno@administrador-20VE:~$ minikube delete --all
🔥  Eliminando "minikube" en docker...
🔥  Eliminando /home/alumno/.minikube/machines/minikube...
🔥  Eliminando /home/alumno/.minikube/machines/minikube-m02...
🔥  Eliminando /home/alumno/.minikube/machines/minikube-m03...
💀  Removed all traces of the "minikube" cluster.
🔥  Successfully deleted all profiles
```

Esto eliminará el clúster predeterminado (llamado minikube) y el clúster **demo** en nuestro caso.

Con esto, pasaremos a la siguiente herramienta interesante y útil a la hora de trabajar con contenedores y Kubernetes. Debería tenerlo instalado y disponible en la computadora de su trabajo.

Kind

Kind (https://kind.sigs.k8s.io/docs/user/quick-start) es otra herramienta popular que se puede utilizar para ejecutar un clúster de Kubernetes de múltiples nodos localmente en su máquina. Es muy fácil de instalar y usar.

Vamos:

1. En una máquina Linux, puedes usar el siguiente script para instalar Kind desde sus

archivos binarios:

curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.22.0/kind-linux-amd64

```
alumno@administrador-20VE:~$ curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.22.0
/kind-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    167      0 --:--:-- --:--:-- --:--:--   168
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6245k  100 6245k    0     0  4912k      0  0:00:01  0:00:01 --:--:-- 21.7M
```

$chmod +x ./kind

$sudo mv ./kind /usr/local/bin/kind

```
alumno@administrador-20VE:~$ chmod +x ./kind
alumno@administrador-20VE:~$ sudo mv ./kind /usr/local/bin/kind
[sudo] contraseña para alumno:
Lo siento, pruebe otra vez.
[sudo] contraseña para alumno:
alumno@administrador-20VE:~$
```

2. Una vez instalado Kind, pruébelo con el siguiente comando:

$ kind version

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kind version
kind v0.22.0 go1.20.13 linux/amd64
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

RESULTADO MIO

```
alumno@administrador-20VE:~$ kind version
kind v0.22.0 go1.20.13 linux/amd64
```

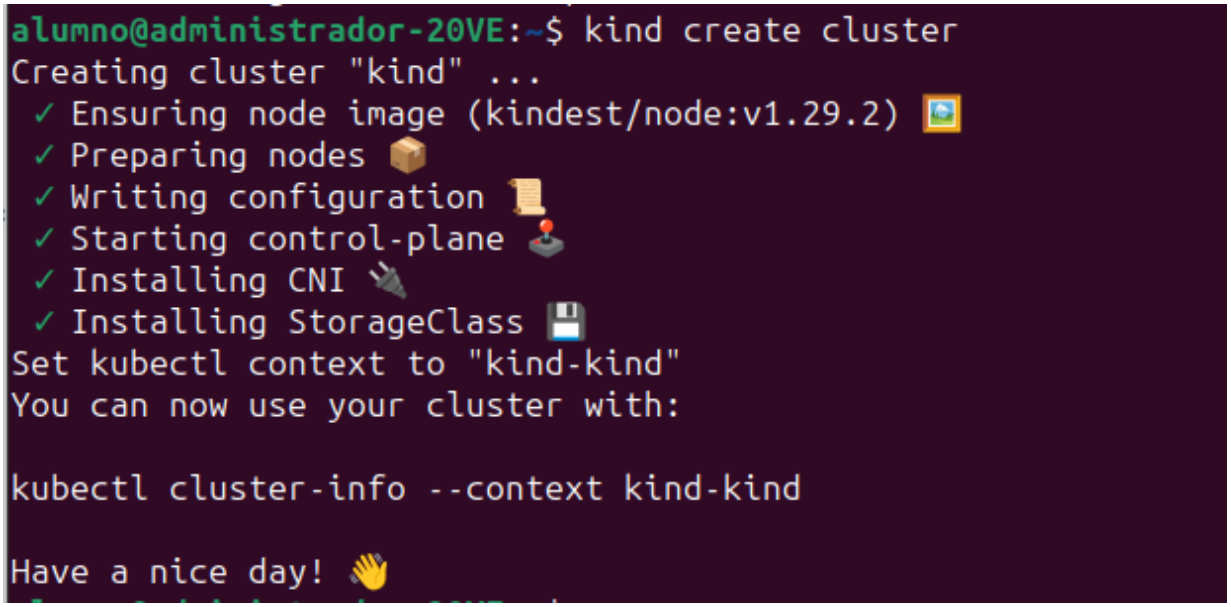
3. Ahora, intente crear un clúster de Kubernetes simple que consta de un nodo maestro y dos nodos trabajadores. Utilice este comando para lograr esto:

$ kind create cluster

Después de un tiempo, deberías ver este resultado:

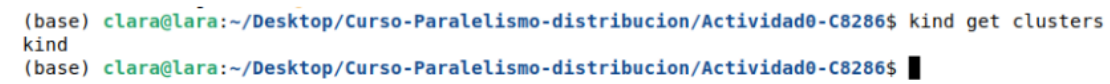
```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kind create cluster
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.29.2) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Thanks for using kind! 😊
```

RESULTADO MIO

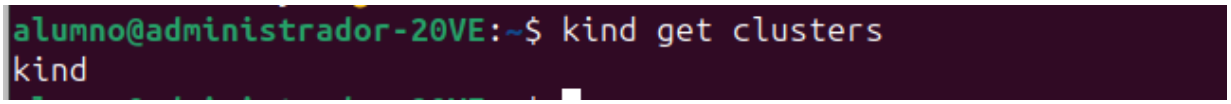


4. Para verificar que se ha creado un clúster, utilice este comando:
$ kind get clusters

El resultado anterior muestra que hay exactamente un clúster llamado **kind,** que es el nombre predeterminado.



RESULTADO MIO



5. Podemos crear un clúster adicional con un nombre diferente usando el parámetro -- **name,** así:

$ kind create cluster --name demo

RESULTADO MIO

```
alumno@administrador-20VE:~$ kind create cluster --name demo
Creating cluster "demo" ...
 ✓ Ensuring node image (kindest/node:v1.29.2) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-demo"
You can now use your cluster with:

kubectl cluster-info --context kind-demo

Have a nice day! 👋
alumno@administrador-20VE:~$
```

6. Al enumerar los clústeres se mostrará esto:

$ kind get clusters

```
alumno@administrador-20VE:~$ kind get clusters
demo
kind
alumno@administrador-20VE:~$
```

Y esto funciona como se esperaba.

Ahora que hemos usado kind para crear dos clústeres de muestra, usemos kubectl para jugar con uno de los clústeres y ejecutar la primera aplicación en él. Usaremos Nginx para esto, similar a lo que hicimos con minikube:

Ahora podemos usar kubectl para acceder y trabajar con los clústeres que acabamos de crear. Mientras creaba un clúster, Kind también actualizó el archivo de configuración de nuestro kubectl. Podemos verificar esto con el siguiente comando:

$ kubectl config get-contexts

Debería producir el siguiente resultado:

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kubectl config get-contexts
CURRENT   NAME             CLUSTER          AUTHINFO         NAMESPACE
          docker-desktop   docker-desktop   docker-desktop
*         kind-demo        kind-demo        kind-demo
          kind-kind        kind-kind        kind-kind
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

RESULTADO MIO

```
alumno@administrador-20VE:~$ kubectl config get-contexts
CURRENT   NAME         CLUSTER      AUTHINFO    NAMESPACE
*         kind-demo    kind-demo    kind-demo
          kind-kind    kind-kind    kind-kind
alumno@administrador-20VE:~$
```

Puede ver que los clústeres **kind** y de **demo** son parte de la lista de clústeres conocidos y que el clúster de demo es el contexto actual para kubectl.

2. Utilice el siguiente comando para convertir el clúster de demo en su clúster actual si el asterisco indica que hay otro clúster actual:

$ kubectl config use-context kind-demo

```
alumno@administrador-20VE:~$ kubectl config use-context kind-demo
Switched to context "kind-demo".
alumno@administrador-20VE:~$
```

3. Enumeremos todos los nodos del clúster de muestra:

$ kubectl get nodes

La salida debería ser así:

```
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
demo-control-plane   Ready    control-plane   11m   v1.29.2
(base) clara@lara:~/Desktop/Curso-Paralelismo-distribucion/Actividad0-C8286$
```

RESULTADO MIO

```
alumno@administrador-20VE:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE     VERSION
demo-control-plane   Ready    control-plane   4m46s   v1.29.2
alumno@administrador-20VE:~$
```

4. Ahora, intentemos ejecutar el primer contenedor en este clúster. Usaremos nuestro servidor web Nginx de confianza, como hicimos antes. Utilice el siguiente comando para ejecutarlo:

$ kubectl apply -f nginx.yaml

```
alumno@administrador-20VE:~$ kubectl apply -f /home/alumno/Descargas/nginx.yaml
pod/nginx created
alumno@administrador-20VE:~$
```

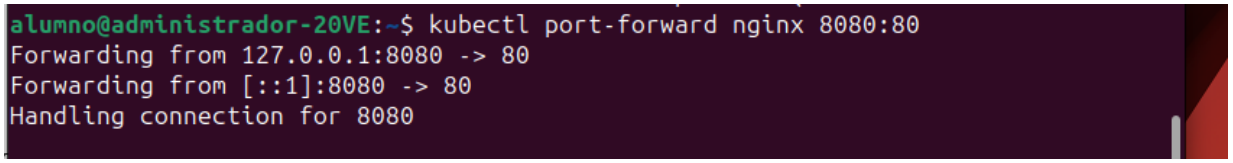
El resultado debería ser el siguiente:

pod/nginx created

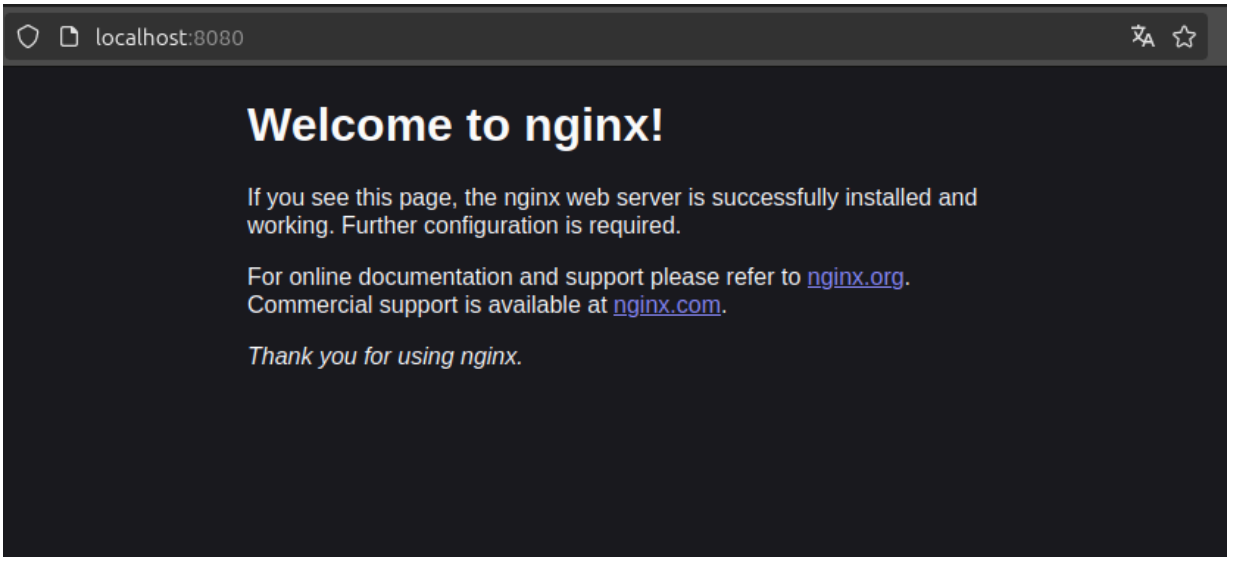
5. Para acceder al servidor Nginx, necesitamos realizar el reenvío de puertos usando kubectl. Utilice este comando para hacerlo:

Revisa: kubectl describe pod nginx, https://www.kristhecodingunicorn.com/post/kubernetes port-forwarding-cleanup-of-orphaned-ports/

$ kubectl port-forward nginx 8080 80 (puedes usar otros puertos)



Abra una nueva pestaña del navegador y navegue hasta http://localhost:8080; Deberías ver la pantalla de bienvenida de Nginx



Una vez que hayas terminado de jugar con Nginx, usa este comando para eliminar el pod del clúster:

$ kubectl delete -f nginx.yaml

kubectl delete -f /home/alumno/Descargas/nginx.yaml

```
alumno@administrador-20VE:~$ kubectl delete -f /home/alumno/Descargas/nginx.yaml

pod "nginx" deleted
```

Antes de continuar, limpiemos y eliminemos los dos clústeres que acabamos de crear:

$ kind delete cluster --name kind

$ kind delete cluster --name demo

```
alumno@administrador-20VE:~$ kind delete cluster --name kind
kind delete cluster --name demo
Deleting cluster "kind" ...
Deleted nodes: ["kind-control-plane"]
Deleting cluster "demo" ...
Deleted nodes: ["demo-control-plane"]
alumno@administrador-20VE:~$
```

Con esto, hemos instalado todas las herramientas que necesitaremos para trabajar exitosamente con contenedores en nuestra máquina local.

Preguntas

Con base en lo que se cubrió en esta actividad, responda las siguientes preguntas:

1. En tus propias palabras, usando analogías, explica qué es un contenedor.

Un contenedor es como una caja fuerte que contiene todo lo que un programa necesita para funcionar, como archivos, librerías y configuraciones, pero aislado del resto del sistema, como un envase que guarda todos los ingredientes de una comida por separado, listos para ser usados cuando se necesiten.

2. ¿Por qué se considera que los contenedores cambian las reglas del juego en IT? Mencione tres o cuatro razones.

Los contenedores cambian las reglas del juego en IT porque son más rápidos, portátiles y eficientes. Permiten un desarrollo más ágil, facilitan la gestión de aplicaciones y hacen que la infraestructura sea más escalable.

3. ¿Qué significa cuando afirmamos que, si un contenedor se ejecuta en una plataforma determinada, entonces se ejecutará en cualquier lugar? Mencione dos o tres razones por las que esto es cierto.

Cuando decimos que un contenedor se ejecuta en una plataforma determinada, significa que funcionará en cualquier lugar debido a su portabilidad y consistencia. Esto es cierto porque los contenedores encapsulan todo lo que necesitan para ejecutarse, independientemente del entorno.

4. ¿Es verdadera o falsa la siguiente afirmación: los contenedores Docker solo son útiles para aplicaciones modernas y totalmente nuevas basadas en microservicios? Por favor justifique su respuesta.

Falsa. Los contenedores Docker no solo son útiles para aplicaciones modernas y nuevas basadas en microservicios, también pueden beneficiar a aplicaciones existentes al facilitar su despliegue y gestión. Son flexibles y pueden adaptarse a diferentes tipos de aplicaciones

5. ¿Por qué nos importaría instalar y usar un administrador de paquetes en nuestra computadora local?

Nos importaría instalar y usar un administrador de paquetes en nuestra computadora local porque nos permite gestionar y actualizar fácilmente software, librerías y herramientas, simplificando la instalación y mantenimiento de programas.

6. ¿Con Docker Desktop, puede desarrollar y ejecutar contenedores de Linux?

Sí, con Docker Desktop puedes desarrollar y ejecutar contenedores de Linux en sistemas Windows y macOS.

7. ¿Por qué son esenciales buenas habilidades de programación (como Bash o PowerShell) para el uso productivo de los contenedores?

Las buenas habilidades de programación son esenciales para el uso productivo de los contenedores porque permiten automatizar tareas, crear scripts y realizar operaciones avanzadas en el entorno de contenedores.

8. Nombra tres o cuatro distribuciones de Linux en las que Docker esté certificado para ejecutarse.

Algunas distribuciones de Linux en las que Docker está certificado para ejecutarse son Ubuntu, CentOS, Red Hat Enterprise Linux y SUSE Linux Enterprise

9. Instalaste minikube en tu sistema. ¿Para qué tipo de tareas utilizarás esta herramienta?

Al instalar Minikube, podré utilizar esta herramienta para crear y gestionar entornos de Kubernetes locales, lo que me permitirá probar y desarrollar aplicaciones en un entorno similar al de producción, pero en mi propia computadora.