

INFORME DE LA ACTIVIDAD 1: INSTALACIÓN DE SISTEMAS OPERATIVOS

1. Objetivo de la Actividad

El objetivo de esta actividad es instalar y verificar la funcionalidad de dos sistemas operativos basados en Linux: **Kubuntu (con interfaz gráfica)** y **Ubuntu Server (solo línea de comandos)**. Esto demuestra la capacidad de configurar máquinas virtuales y manejar diferentes entornos operativos.

2. Metodología

Ambas instalaciones se realizaron utilizando el software de virtualización Oracle VirtualBox, asignando los recursos de hardware necesarios para cada entorno.

PARTE 1.1: INSTALACIÓN DE KUBUNTU (Entorno Gráfico)

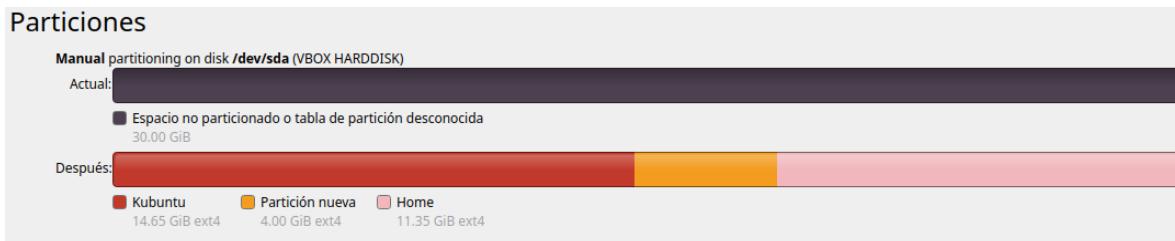
A. Configuración de Particiones

La instalación de Kubuntu requirió la creación de un esquema de particionamiento personalizado para asegurar una distribución organizada del sistema operativo, el área de intercambio y los datos de usuario.

Partición	Tamaño Asignado	Sistema de Archivos	Propósito
Raíz (/)	15 GB	Ext4	Contiene todos los archivos del sistema operativo, programas instalados y configuración general.
Swap	4 GB	Área de Intercambio	Utilizada como memoria virtual cuando la RAM física es insuficiente.

Partición	Tamaño Asignado	Sistema de Archivos	Propósito
Home (/home)	Restante (Aprox. 11 GB)	Ext4	Almacena los archivos personales, documentos y configuraciones de los usuarios.

B. Evidencia de Instalación



PARTE 1.2: INSTALACIÓN DE UBUNTU SERVER (Línea de Comandos)

C. Configuración de Particiones y Usuario

La instalación de Ubuntu Server se realizó sin una interfaz gráfica. El proceso fue asistido por texto, siguiendo la misma estructura de particionamiento, lo cual es fundamental en un entorno de servidor. Se instaló el servidor OpenSSH para futura administración remota.

Partición	Tamaño Asignado	Sistema de Archivos	Propósito
Raíz (/)	15 GB	Ext4	Archivos del sistema operativo y binarios del servidor.

Partición	Tamaño Asignado	Sistema de Archivos	Propósito
Swap	4 GB	Área de Intercambio	Memoria virtual.
Home (/home)	Restante (Aprox. 11 GB)	Ext4	Almacenamiento de datos y scripts de usuarios administradores.

Storage configuration

RESUMEN DEL SISTEMA DE ARCHIVOS

PUNTO DE MONTAJE	TAMAÑO	TIPO	TIPO DE DISPOSITIVO
/	15.000G	new ext4	new partition of disco local ▶]
/home	11.827G	new ext4	new partition of disco local ▶]
SWAP	4.000G	new swap	new partition of disco local ▶]

D. Evidencia Final de Funcionalidad

Una vez completada la instalación y reiniciado el servidor, se accedió al sistema con las credenciales creadas La prueba final verificó la funcionalidad de red del nuevo sistema.

1. Se inició sesión en el servidor (solo texto).
2. Se ejecutó el comando `ip a` para verificar las interfaces de red.

```
Jiner@Jinercito:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:e5:9c:0e brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 85675sec preferred_lft 85675sec
    inet6 fd17:625c:f037:2:a00:27ff:fee5:9c0e/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 86087sec preferred_lft 140087sec
    inet6 fe80::a00:27ff:fee5:9c0e/64 scope link
        valid_lft forever preferred_lft forever
Jiner@Jinercito:"$
```

INFORME DE LA ACTIVIDAD 2: INSTALACIÓN Y PERSONALIZACIÓN DE EDITORES

1. Objetivo de la Actividad

El propósito de esta actividad es demostrar la capacidad de instalar, utilizar y personalizar dos tipos fundamentales de editores de texto en un entorno Linux (Kubuntu): un editor de interfaz gráfica (GUI) y un editor de línea de comandos (CLI).

2. Metodología Utilizada

La actividad se realizó en la máquina virtual Kubuntu.

Se empleó el gestor de paquetes Snap para la instalación del editor gráfico y la edición directa de un archivo de configuración (`.nanorc`) para personalizar el editor de consola.

3. PARTE A: EDITOR GRÁFICO (Visual Studio Code)

Se seleccionó **Visual Studio Code (VS Code)** como el editor con interfaz gráfica debido a su versatilidad y amplio uso. La personalización se centró en la verificación de la **visualización de los números de línea**.

3.1. Instalación

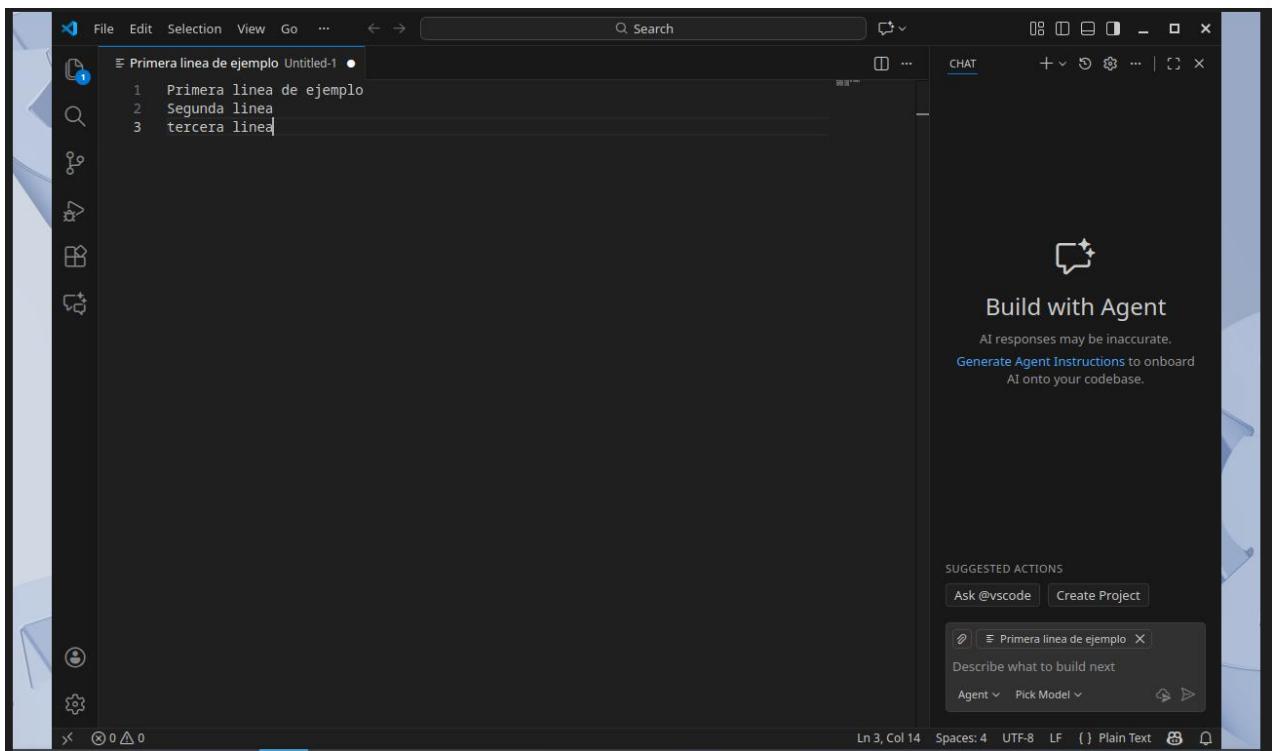
La instalación de Visual Studio Code se llevó a cabo utilizando el gestor de paquetes Snap en la Terminal:

Bash
`sudo snap install code --classic`

3.2. Personalización y Evidencia

La personalización requerida fue la activación de la numeración de línea para facilitar la lectura y el desarrollo.

1. Se accedió a las **Preferencias (Settings)** de Visual Studio Code.
2. Se verificó que la opción **Editor: Line Numbers** estuviera activa.
3. Se creó un archivo de texto de prueba para validar que los números de línea se visualizaban correctamente en el margen izquierdo.



4. PARTE B: EDITOR DE CONSOLA (Nano)

Se seleccionó **Nano** como el editor de texto de línea de comandos (CLI) debido a su simplicidad. La personalización se realizó de manera persistente, configurando el editor para que muestre los **números de línea por defecto** sin intervención manual en cada uso.

4.1. Configuración y Personalización

Para lograr una personalización persistente, se modificó el archivo de configuración de Nano específico del usuario, `.nanorc`, ubicado en el directorio Home.

1. Se abrió el archivo de configuración en la Terminal:

Bash

```
nano ~/.nanorc
```

2. Se insertó la directiva `set linenumbers` para activar la numeración de línea al iniciar Nano:

Bash

```
set linenumbers
```

3. Se guardaron los cambios (**Ctrl + O**) y se salió del editor (**Ctrl + X**).

4.2. Evidencia de la Personalización

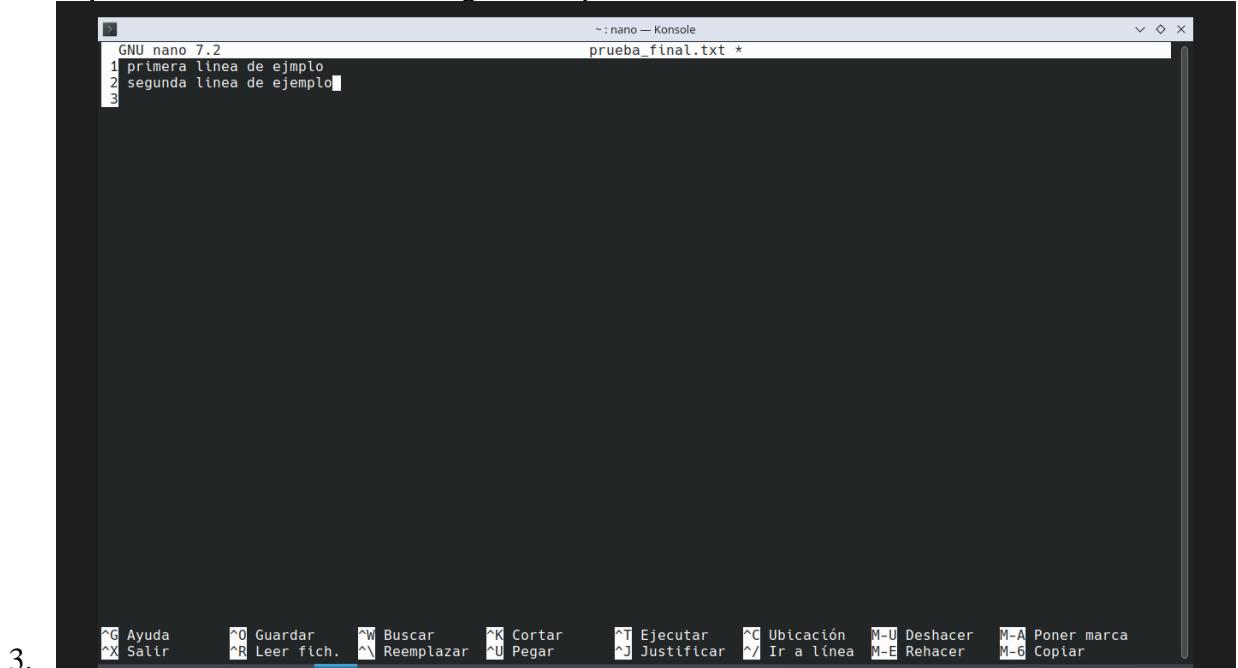
Para comprobar que la personalización fue exitosa, se abrió un archivo de prueba. El resultado validó que los números de línea se activaban automáticamente al inicio.

1. Se ejecutó el comando de prueba:

Bash

```
nano prueba_final.txt
```

2. Se verificó la aparición automática de la numeración de línea en el margen izquierdo, confirmando la configuración persistente.



INFORME – ACTIVIDAD 4: Instalación y configuración de Docker Engine (Debian)

4.1 INSTALACIÓN DE DOCKER ENGINE EN LINUX (DEBIAN)

En esta sección se documenta el proceso de instalación del Docker Engine nativo en Debian, cumpliendo los requerimientos de la guía.

1.1 Actualizar el sistema

```
vboxuser@DEBIAN:~$ sudo apt update  
[sudo] contraseña para vboxuser:  
Obj:1 http://deb.debian.org/debian trixie InRelease  
Obj:2 http://deb.debian.org/debian trixie-updates InRelease  
Obj:3 https://download.docker.com/linux/debian trixie InRelease  
Obj:4 http://security.debian.org/debian-security trixie-security InRelease  
Todos los paquetes están actualizados.  
vboxuser@DEBIAN:~$ sudo apt upgrade -y  
Summary:  
Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0  
vboxuser@DEBIAN:~$
```

1.2 Instalar dependencias necesarias

- ca-certificates: Permite validar certificados HTTPS.
- curl: Descarga archivos desde la terminal.
- gnupg: Gestiona claves GPG necesarias para firmar repositorios

```
vboxuser@DEBIAN:~$ sudo apt install -y ca-certificates curl gnupg lsb-release  
ca-certificates ya está en su versión más reciente (20250419).  
curl ya está en su versión más reciente (8.14.1-2+deb13u2).  
gnupg ya está en su versión más reciente (2.4.7-21).  
lsb-release ya está en su versión más reciente (12.1-1).  
Summary:  
Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

1.3 Agregar el repositorio oficial de Docker

- Crea el directorio donde se almacenarán las claves GPG del repositorio.
- Descarga la clave oficial de Docker y la convierte al formato que usa APT.
- Asigna permisos para que APT pueda leer la clave.

```
vboxuser@DEBIAN:~$ sudo install -m 0755 -d /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
sudo chmod a+r /etc/apt/keyrings/docker.gpg  
El fichero '/etc/apt/keyrings/docker.gpg' ya existe. [Sobreescribir? (s/N) s
```

1.4 Agregar el repositorio

- Añade el repositorio estable de Docker según la versión de Debian instalada.

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

1.5 Actualizar nuevamente

```
vboxuser@DEBIAN: $ sudo apt update
Obj:1 http://deb.debian.org/debian trixie InRelease
Obj:2 http://deb.debian.org/debian trixie-updates InRelease
Obj:3 http://security.debian.org/debian-security trixie-security InRelease
Obj:4 https://download.docker.com/linux/debian trixie InRelease
Todos los paquetes están actualizados.
```

1.6 Instalar Docker y sus componentes

- docker-ce: Motor de Docker.
- docker-ce-cli: Herramientas de comando.
- containerd.io: Motor de contenedores en segundo plano.
- buildx: Permite construir imágenes avanzadas.
- docker compose plugin: Permite usar docker compose.

```
vboxuser@DEBIAN: $ sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
docker-ce ya está en su versión más reciente (5:29.1.2-1~debian.13~trixie).
docker-ce-cli ya está en su versión más reciente (5:29.1.2-1~debian.13~trixie).
containerd.io ya está en su versión más reciente (2.2.0-2~debian.13~trixie).
docker-buildx-plugin ya está en su versión más reciente (0.30.1-1~debian.13~trixie).
docker-compose-plugin ya está en su versión más reciente (5.0.0-1~debian.13~trixie).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

```
vboxuser@DEBIAN: $ sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
    Active: active (running) since Thu 2025-12-11 18:06:31 -05; 4min 36s ago
      Invocation: fb12d3f5732c4581a447d6aa455d9626
TriggeredBy: ● docker.socket
  Docs: https://docs.docker.com
 Main PID: 6781 (dockerd)
   Tasks: 11
     Memory: 28.1M (peak: 28.9M)
       CPU: 1.775s
      CGroup: /system.slice/docker.service
              └─6781 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

dic 11 18:06:29 DEBIAN dockerd[6781]: time="2025-12-11T18:06:29.718445142-05:00" level=info msg="Restoring containers: start."
dic 11 18:06:29 DEBIAN dockerd[6781]: time="2025-12-11T18:06:29.876010664-05:00" level=info msg="Deleting nftables IPv4 rules" error="exit"
dic 11 18:06:29 DEBIAN dockerd[6781]: time="2025-12-11T18:06:29.918073424-05:00" level=info msg="Deleting nftables IPv6 rules" error="exit"
dic 11 18:06:29 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.371476923-05:00" level=info msg="Loading containers: done."
dic 11 18:06:31 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.402967284-05:00" level=info msg="Docker daemon" commit=d045c2a containerd-s>
dic 11 18:06:31 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.403198949-05:00" level=info msg="Initializing buildkit"
dic 11 18:06:31 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.542845816-05:00" level=info msg="Completed buildkit initialization"
dic 11 18:06:31 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.578332738-05:00" level=info msg="Daemon has completed initialization"
dic 11 18:06:31 DEBIAN systemd[1]: Started docker.service - Docker Application Container Engine.
dic 11 18:06:31 DEBIAN dockerd[6781]: time="2025-12-11T18:06:31.578864905-05:00" level=info msg="API listen on /run/docker.sock"
```

1.7 Verificar instalación

- Ejecuta una imagen de prueba para confirmar que Docker funciona correctamente.

```
vboxuser@DEBIAN:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
ea52d2000f90: Download complete
Digest: sha256:d4aab6242e0cace87e2ec17a2ed3d779d18fbfd03042ea58f2995626396a274
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/
```

4.2 CREACIÓN DE IMAGEN DOCKER PERSONALIZADA (DOCKERFILE)

1. Crear directorio del proyecto

- Crea la carpeta donde estará tu Dockerfile.
- Entra en dicha carpeta.

```
vboxuser@DEBIAN:~$ mkdir miapp
vboxuser@DEBIAN:~$ cd miapp
vboxuser@DEBIAN:~/miapp$ █
```

2. Crear el Dockerfile

- Abre el editor nano para crear el archivo Dockerfile.

```
vboxuser@DEBIAN:~/miapp$ nano Dockerfile█
```

3. Contenido del Dockerfile (explicado brevemente)

- Usa Debian como imagen base.

- Instala herramientas de compilación (build-essential).
- Crea un usuario sin privilegios.
- Define que el contenedor se ejecute con este usuario.
- Establece el directorio de trabajo.
- Ejecuta una terminal Bash al iniciar el contenedor.

```
GNU nano 8.4                                            Dockerfile *
FROM debian:latest
RUN apt update && apt install -y gcc g++ make
RUN useradd -ms /bin/bash desarrollador
USER desarrollador
WORKDIR /home/desarrollador
CMD ["bash"]
```

4. Construir la imagen

- Construye la imagen con nombre miimagen y versión 1.0
- El punto (.) significa: usa el Dockerfile en este directorio.

```
vboxuser@DEBIAN:~/miapp$ sudo docker build -t miimagen:1.0 .
[sudo] contraseña para vboxuser:
[+] Building 1.8s (8/8) FINISHED
--> [internal] load build definition from Dockerfile
--> >> transferring dockerfile: 204B
--> [internal] load metadata for docker.io/library/debian:latest
--> [internal] load .dockerignore
--> >> transferring context: 2B
--> [1/4] FROM docker.io/library/debian:latest@sha256:0d01188e8dd0ac63bf155900fad49279131a876alea7fac917c62e87ccb2732d
--> >> resolve docker.io/library/debian:latest@sha256:0d01188e8dd0ac63bf155900fad49279131a876alea7fac917c62e87ccb2732d
--> CACHED [2/4] RUN apt update && apt install -y gcc g++ make
--> CACHED [3/4] RUN useradd -ms /bin/bash desarrollador
--> CACHED [4/4] WORKDIR /home/desarrollador
--> exporting to image
--> >> exporting layers
--> >> exporting manifest sha256:b8e5b08f19c20dc92af43a606eeb16a4df45ea124ef72547ae0700c76929e3e0
--> >> exporting config sha256:lcbe5804ebd4cb81e92363fd70b292ac0747d2bc7b486b06968ac6ae9c9c3d
--> >> exporting attestation manifest sha256:08ded7b4441620bdb8734f51396e47bb8b7f4814040dc9c4286226c10c928d6
--> >> exporting manifest list sha256:1ed5a9b4aae525ac21a5e159297ff0d2b49f2ccal5b19711a74c9bceee25bd8d
--> >> naming to docker.io/library/miimagen:1.0
--> >> unpacking to docker.io/library/miimagen:1.0
vboxuser@DEBIAN:~/miapp$
```

4.3 CREACIÓN Y GESTIÓN DE CONTENEDORES — (VERSIÓN DEBIAN CON DESCRIPCIONES)

1. Crear y ejecutar un contenedor interactivo

- docker run → crea y ejecuta un contenedor.
- -i → modo interactivo (recibe entrada de teclado).
- -t → asigna una terminal tipo TTY.
- --name → asigna un nombre personalizado.
- miimagen:1.0 → imagen base que tú creaste.
- **Ctrl + P + Q** → desconecta la terminal, pero el contenedor sigue en ejecución.
- **exit** → cierra la sesión y detiene el contenedor.

```
vboxuser@DEBIAN:~$ sudo docker run -it --name contenedor_prueba miimagen:1.0 bash  
[sudo] contraseña para vboxuser:  
desarrollador@a5a80aafdal1c:~$ vboxuser@DEBIAN:~$
```

2. Listar contenedores en ejecución

- Muestra solo los contenedores que están activos.

```
desarrollador@a1d7e5decf2a:~$ vboxuser@DEBIAN:~$ sudo docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
a1d7e5decf2a miimagen:1.0 "bash" 14 seconds ago Up 13 seconds  
a5a80aafdal1c miimagen:1.0 "bash" About a minute ago Up About a minute  
f6b0509e3731 hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
```

3. Volver a entrar al contenedor

- docker exec → ejecuta un comando dentro de un contenedor ya creado.
- -it → te permite entrar como si fuera una terminal real.

```
vboxuser@DEBIAN:~$ sudo docker exec -it contenedor_prueba bash  
desarrollador@a5a80aafdal1c:~$ read escape sequence
```

4. Listar contenedores en ejecución

- Muestra solo los contenedores que están activos.

```
vboxuser@DEBIAN:~$ sudo docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
a1d7e5decf2a miimagen:1.0 "bash" About a minute ago Up About a minute  
a5a80aafdal1c miimagen:1.0 "bash" 2 minutes ago Up 2 minutes  
f6b0509e3731 hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
```

5. Detener un contenedor

- Apaga correctamente un contenedor en ejecución.

```
vboxuser@DEBIAN:~$ sudo docker stop contenedor_prueba  
contenedor_prueba
```

6. Listar contenedores en ejecución

- Muestra solo los contenedores que están activos

```
vboxuser@DEBIAN:~$ sudo docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
a1d7e5decf2a miimagen:1.0 "bash" 3 minutes ago Up 3 minutes  
a5a80aafdal1c miimagen:1.0 "bash" 4 minutes ago Exited (137) 16 seconds ago  
f6b0509e3731 hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
```

7. Iniciar un contenedor detenido

- Arranca un contenedor que se encontraba detenido.

```
vboxuser@DEBIAN:~$ sudo docker start contenedor_prueba  
contenedor_prueba
```

8. Listar contenedores en ejecución

- Muestra solo los contenedores que están activos.

```
vboxuser@DEBIAN: $ sudo docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a1d7e5decf2a miiimagen:1.0 "bash" 4 minutes ago Up 4 minutes
a5a80aafda1c miiimagen:1.0 "bash" 5 minutes ago Up 16 seconds
f6b0509e3731 hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
contenedor_pruebal
contenedor_prueba
great_dhawan
```

9. Detener un contenedor

- Apaga correctamente un contenedor en ejecución.

```
vboxuser@DEBIAN:~$ sudo docker stop contenedor_pruebal
contenedor_pruebal
```

10. Eliminar un contenedor

- docker rm → borra un contenedor.
- Solo funciona si está detenido.

```
vboxuser@DEBIAN:~$ sudo docker rm contenedor_pruebal
contenedor_pruebal
```

11. Listar contenedores en ejecución

- Muestra solo los contenedores que están activos.

```
vboxuser@DEBIAN: $ sudo docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a5a80aafda1c miiimagen:1.0 "bash" 8 minutes ago Up 3 minutes
f6b0509e3731 hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
contenedor_prueba
great_dhawan
```

12. Listar imágenes disponibles

- Muestra todas las imágenes instaladas en tu sistema.

```
vboxuser@DEBIAN: $ sudo docker images


| IMAGE              | ID           | DISK USAGE | CONTENT SIZE | EXTRA |
|--------------------|--------------|------------|--------------|-------|
| hello-world:latest | d4aaab6242e0 | 25.9kB     | 9.52kB       | U     |
| miiimagen:1.0      | 792561e5975c | 620MB      | 171MB        | U     |
| miiimagen:1.0      | 1ed5a9b4aae5 | 620MB      | 171MB        | U     |


vboxuser@DEBIAN: $ sudo docker images
```

5. INFORME — Creación y Publicación de Imagen Docker Mejorada (Versión 2.0)

1. Crear una nueva carpeta del proyecto

Se crea el directorio donde se almacenará la versión mejorada de la imagen Docker.

Comandos:

```
vboxuser@DEBIAN:~$ mkdir proyecto_v2
vboxuser@DEBIAN:~$ cd proyecto_v2
vboxuser@DEBIAN:~/proyecto_v2$
```

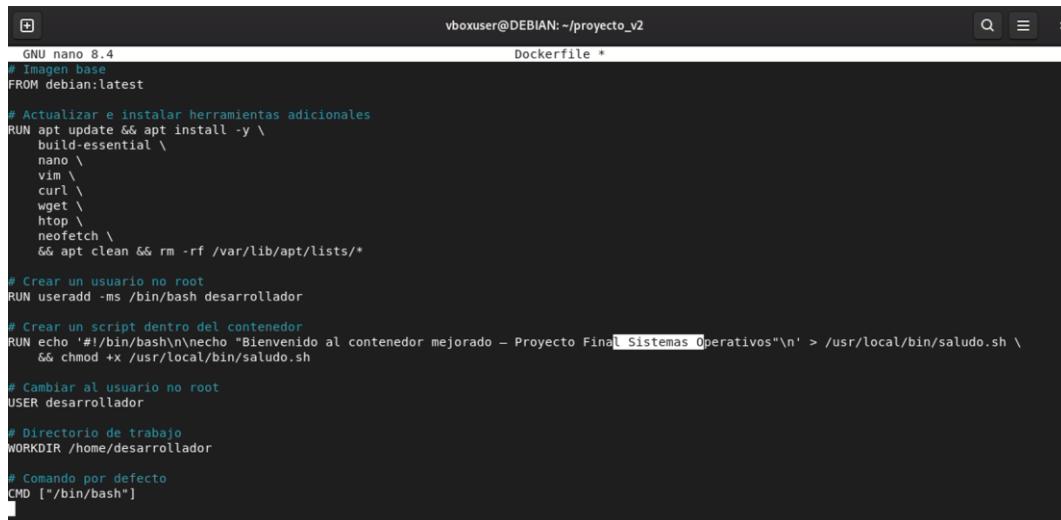
2. Crear el Dockerfile mejorado

Dentro del proyecto abrimos un Dockerfile con nano.

Comando:

```
vboxuser@DEBIAN:~$ mkdir proyecto_v2
vboxuser@DEBIAN:~$ cd proyecto_v2
vboxuser@DEBIAN:~/proyecto_v2$ nano Dockerfile
```

Contenido completo del Dockerfile:



```
GNU nano 8.4
# Imagen base
FROM debian:latest

# Actualizar e instalar herramientas adicionales
RUN apt update && apt install -y \
    build-essential \
    nano \
    vim \
    curl \
    wget \
    htop \
    neofetch \
    && apt clean && rm -rf /var/lib/apt/lists/*

# Crear un usuario no root
RUN useradd -ms /bin/bash desarrollador

# Crear un script dentro del contenedor
RUN echo "#!/bin/bash\n\nnecho \"Bienvenido al contenedor mejorado - Proyecto Final Sistemas Operativos\"\n" > /usr/local/bin/saludo.sh \
    && chmod +x /usr/local/bin/saludo.sh

# Cambiar al usuario no root
USER desarrollador

# Directorio de trabajo
WORKDIR /home/desarrollador

# Comando por defecto
CMD ["/bin/bash"]
```

Descripción breve del Dockerfile:

- Usa **Debian** como base.
- Instala herramientas esenciales: compiladores, editores y utilidades.
- Crea un usuario sin privilegios llamado *desarrollador*.
- Genera el script saludo.sh accesible globalmente desde /usr/local/bin.
- Define el directorio de trabajo y el intérprete Bash como comando inicial.

3. Construcción de la imagen mejorada

- Desde la carpeta del proyecto se ejecutó:

```
vboxuser@DEBIAN:/proyecto_v2$ sudo docker build -t miimage:2.0 .
[sudo] contraseña para vboxuser:
[+] Building 3.0s (4/8)
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 726B
--> [internal] load metadata for docker.io/library/debian:latest
--> [internal] load .dockerignore
--> => transferring context: 2B
--> CACHED [/S1] FROM docker.io/library/debian:latest@sha256:0d01188e8dd0ac63bf1f155900fad49279131a876a1ea7fac917c62e87ccb2732d
--> => resolve docker.io/library/debian:latest@sha256:0d01188e8dd0ac63bf1f155900fad49279131a876a1ea7fac917c62e87ccb2732d
--> [2/5] RUN apt update && apt install -y build-essential nano vim curl wget htop neofetch && apt clean
--> => # WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
--> => # Get:1 http://deb.debian.org/debian trixie InRelease [140 kB]
--> => # Get:2 http://deb.debian.org/debian trixie-updates InRelease [47.3 kB]
--> => # Get:3 http://deb.debian.org/debian-security trixie-security InRelease [43.4 kB]
--> => # Get:4 http://deb.debian.org/debian trixie/main amd64 Packages [9670 kB]
```

- `sudo docker build -t miimage:2.0`.
- El sistema descargó paquetes, configuró usuarios y generó la imagen final.
- La línea final del proceso muestra:

```
-----
Dockerfile:5
-----
4 |      # Actualizar e instalar herramientas adicionales
5 | >>> RUN apt update && apt install -y \
6 | >>>      build-essential \
7 | >>>      nano \
8 | >>>      vim \
9 | >>>      curl \
10 | >>>      wget \
11 | >>>      htop \
12 | >>>      neofetch \
13 | >>>      && apt clean && rm -rf /var/lib/apt/lists/*
14 |
```

4. Comparación de tamaños

Se listaron las imágenes instaladas y se obtuvo una captura donde se observan los tamaños de cada una. Esto evidencia que la nueva versión incluye más herramientas y archivos:

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
hello-world:latest	d4aaab6242e0	25.9kB	9.52kB	U
miimage:1.0	792561e5975c	620MB	171MB	U
miimage:2.0	1b659656105c	3.96GB	1.1GB	
miimagen:1.0	1ed5a9b4aae5	620MB	171MB	U

5. Ejecución de la imagen mejorada

Para iniciar el contenedor se ejecutó. Esto abre una sesión interactiva dentro del contenedor :

```
vboxuser@DEBIAN:~/proyecto_v2$ sudo docker run -it --name contenedor_mejorado miimage:2.0 /bin/bash  
desarrollador@c55a50c7c242:~$
```

Para validar el Script:

```
desarrollador@c55a50c7c242:~$ saludo.sh  
Bienvenido al contenedor mejorado – Proyecto Final Sistemas Operativos  
desarrollador@c55a50c7c242:~$
```

CONCLUSIÓN

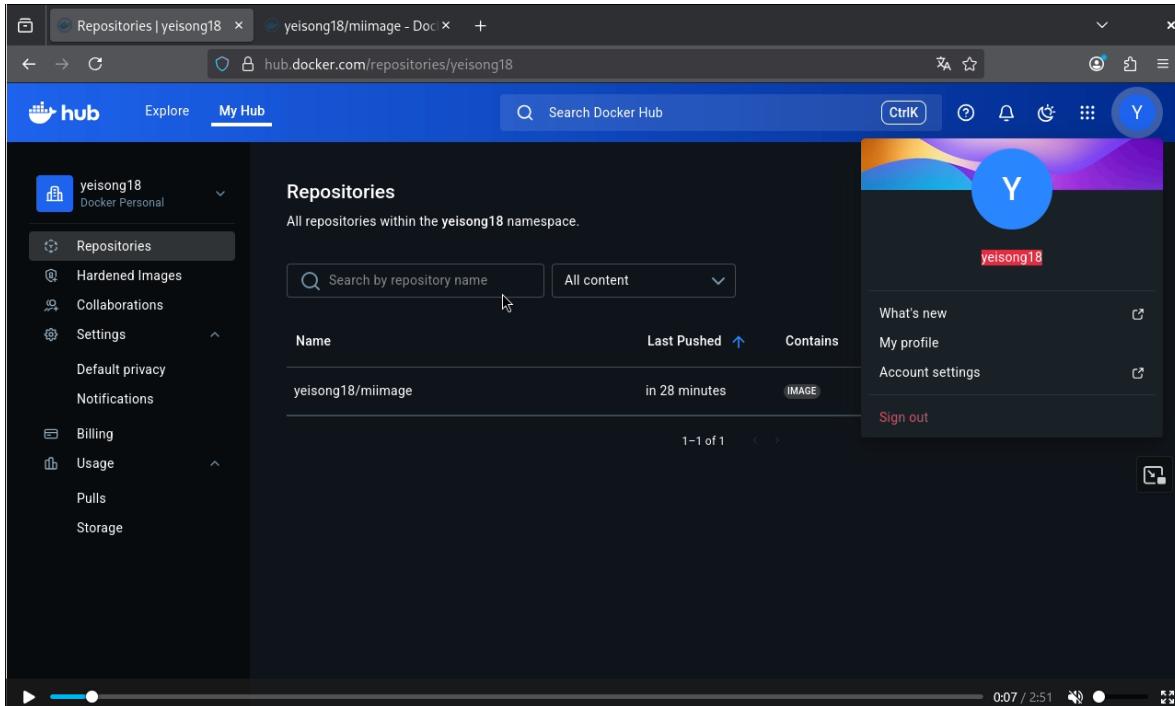
En esta actividad se modificó la imagen Docker previamente creada en el proyecto para agregar funcionalidades adicionales, optimizar su utilidad y posteriormente publicar la versión mejorada en Docker Hub.

El objetivo es demostrar el proceso completo: creación de una imagen avanzada, ejecución de scripts dentro del contenedor, edición de un Dockerfile mejorado y publicación final en un repositorio público.

La nueva imagen miimage:2.0 incluye herramientas de administración, edición, compilación y un script automatizado accesible desde cualquier parte del contenedor.

5.2 INFORME — Descripción detallada de cada paso y comando - Publicación en Docker Hub

1. Tener una cuenta en Docker Hun



2. Iniciar sesión en Docker Hub

Comando

```
vboxuser@DEBIAN: $ sudo docker login
USING WEB-BASED LOGIN
Info → To sign in with credentials on the command line, use 'docker login -u <username>'

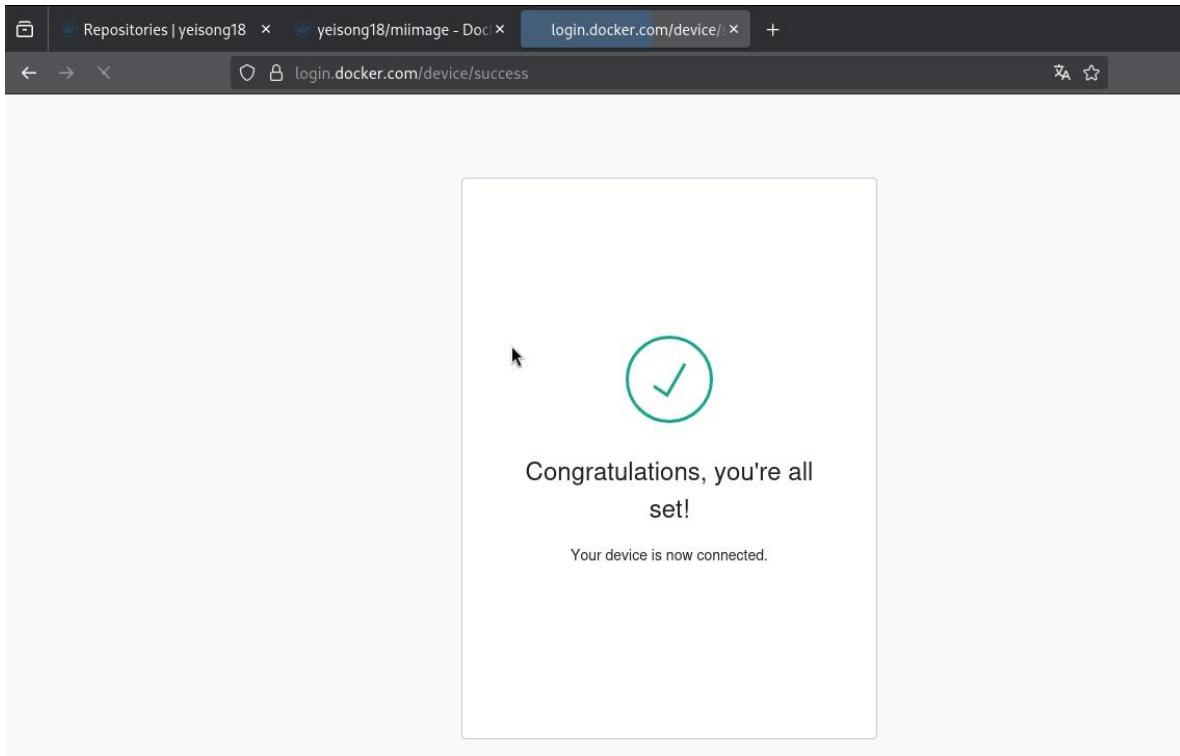
Your one-time device confirmation code is: XCFX-LXVQ
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate
Waiting for authentication in the browser...
```

Descripción

Este comando abre una sesión desde tu computador hacia Docker Hub.

- Permite autenticarte usando:
- Usuario: yeisong18
- Contraseña: la de tu cuenta de Docker Hub

Sin esta autenticación, no puedes subir imágenes al repositorio.



3. Subir la imagen al repositorio en Docker Hub

Comando:

```
vboxuser@DEBIAN:~$ sudo docker tag miimage:2.0 yeisong18/miimage:2.0
vboxuser@DEBIAN:~$
```

Descripción

Este comando renombra (etiqueta) la imagen local para que tenga el formato correcto requerido por Docker Hub:

usuario/imagen:tag

En mi caso queda:

- Usuario: yeisong18
- Imagen: miimage
- Etiqueta: 2.0

Esta etiqueta indica la versión que vamos a publicar.

4. Subir la imagen al repositorio en Docker Hub

Comando:

```
vboxuser@DEBIAN:~$ sudo docker push yeisong18/miimage:2.0
The push refers to repository [docker.io/yeisong18/miimage]
85636c73425a: Already exists
4f4fb700ef54: Layer already exists
61de855609c5: Layer already exists
4f58d5466508: Layer already exists
2981f7e8980b: Layer already exists
5da71eb08337: Layer already exists
2.0: digest: sha256:1b659656105c6d597fd5d8f9bffe950c3c5579f58f110a3cac53424784466702 size: 856
vboxuser@DEBIAN:~$
```

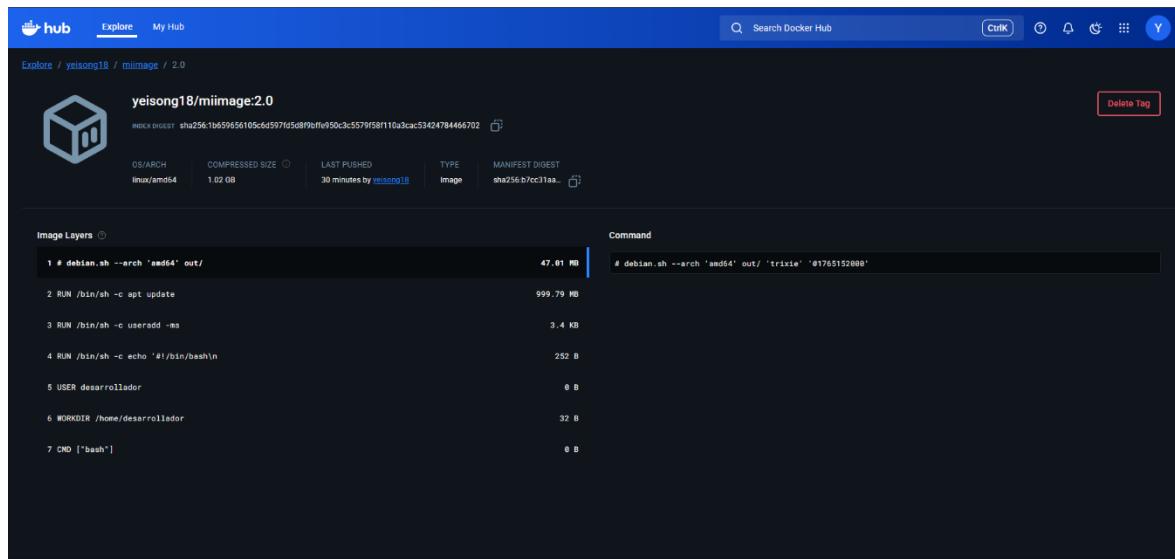
Descripción

- Este comando envía (sube) la imagen desde tu máquina hacia tu repositorio público en Docker Hub.
- Durante la subida verás que Docker envía varias capas (layers) comprimidas.
- Cuando finaliza, tu imagen queda disponible públicamente.

5. Verificar la subida en Docker Hub

URL

👉 <https://hub.docker.com/r/yeisong18/miimage>



Descripción

Al abrir esa página deberías ver:

- La imagen miimage

- La versión publicada 2.0
- Las capas que contiene
- El tamaño total
- El botón de "Pull" para descargarla

ACTIVIDAD 6 – Kubernetes con Minikube (Debian)

6.1 Instalación y configuración de Minikube y kubectl

- Paso 1: Actualizar el sistema

```
sudo apt update
```

```
sudo apt upgrade -y
```

```
vboxuser@DEBIAN:~$ sudo apt update
sudo apt install -y kubectl
[sudo] contraseña para vboxuser:
Obj:1 http://security.debian.org/debian-security trixie-security InRelease
Obj:2 http://deb.debian.org/debian trixie InRelease
Obj:3 http://deb.debian.org/debian trixie-updates InRelease
Obj:4 https://download.docker.com/linux/debian trixie InRelease
Todos los paquetes están actualizados.
kubectl ya está en su versión más reciente (1.32.3+ds-2).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
vboxuser@DEBIAN:~$ █
```

Descripción:

Actualiza la lista de paquetes y el sistema antes de instalar nuevas herramientas.

- Paso 2: Instalar kubectl

```
sudo apt install -y kubectl
```

```
vboxuser@DEBIAN:~$ sudo apt update
sudo apt install -y kubectl
[sudo] contraseña para vboxuser:
Obj:1 http://security.debian.org/debian-security trixie-security InRelease
Obj:2 http://deb.debian.org/debian trixie InRelease
Obj:3 http://deb.debian.org/debian trixie-updates InRelease
Obj:4 https://download.docker.com/linux/debian trixie InRelease
Todos los paquetes están actualizados.
kubectl ya está en su versión más reciente (1.32.3+ds-2).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
vboxuser@DEBIAN:~$ █
```

Descripción:

Instala kubectl, el cliente oficial para administrar clústeres de Kubernetes.

Verificar instalación:

kubectl version --client

```
vboxuser@DEBIAN:~$ kubectl version --client
Client Version: v1.32.3
Kustomize Version: v5.5.0
```

✓ Paso 3: Instalar Minikube

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

```
vboxuser@DEBIAN: $ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total   Spent   Left  Speed
100 133M 100 133M    0      0  27.1M      0  0:00:04  0:00:04  --- 30.3M
```

Dar permisos y mover el binario:

sudo install minikube-linux-amd64 /usr/local/bin/minikube

```
vboxuser@DEBIAN:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
vboxuser@DEBIAN:~$
```

Verificar instalación:

minikube version

```
vboxuser@DEBIAN:~$ minikube version
minikube version: v1.37.0
```

Paso 4: Iniciar el clúster con Docker (driver)

```
minikube start --driver=docker
```

```
vboxuser@DEBIAN: $ minikube start
└─ minikube v1.37.0 en Debian 13.2 (vbox/amd64)
  └─ Using the docker driver based on user configuration
    └─ Using Docker driver with root privileges
      └─ Starting "minikube" primary control-plane node in "minikube" cluster
        └─ Pulling base image v0.0.48 ...
          └─ Descargando Kubernetes v1.34.0 ...
            └─ > preloaded-images-k8s-v18-v1...: 337.07 MiB / 337.07 MiB 100.00% 2.70 Mi
              └─ > gcr.io/k8s-minikube/kicbase...: 488.51 MiB / 488.52 MiB 100.00% 3.30 Mi
                └─ Creating docker container (CPUs=2, Memory=3072MB) ...

          └─ Docker is nearly out of disk space, which may cause deployments to fail! (91% of capacity). You can pass '--force' to skip this check.
            └─ Suggestion:
              Try one or more of the following to free up space on the device:
              1. Run "docker system prune" to remove unused Docker data (optionally with "-a")
              2. Increase the storage allocated to Docker for Desktop by clicking on:
                  Docker icon > Preferences > Resources > Disk Image Size
              3. Run "minikube ssh -- docker system prune" if using the Docker container runtime
              └─ Related issue: https://github.com/kubernetes/minikube/issues/9024

        └─ Preparando Kubernetes v1.34.0 en Docker 28.4.0...
          └─ Configurando CNI bridge CNI ...
            └─ Verifying Kubernetes components...
              └─ Using image gcr.io/k8s-minikube/storage-provisioner:v5
                └─ Complementos habilitados: storage-provisioner, default-storageclass

          └─ /usr/bin/kubectl is version 1.32.3, which may have incompatibilities with Kubernetes 1.34.0.
            └─ • Want kubectl v1.34.0? Try 'minikube kubectl -- get pods -A'
              └─ Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Descripción:

Crea un clúster Kubernetes local usando Docker como motor.

Paso 5: Verificar el clúster

```
kubectl get nodes
```

```
vboxuser@DEBIAN:~$ kubectl get nodes
NAME      STATUS   ROLES      AGE      VERSION
minikube  Ready    control-plane  4m17s   v1.34.0
```

Descripción:

Muestra los nodos activos del clúster.

Debe aparecer un nodo en estado Ready.

 Paso 6: Información del clúster

kubectl cluster-info

```
vboxuser@DEBIAN:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
vboxuser@DEBIAN:~$ █
```

Descripción:

Muestra la información básica del clúster Kubernetes.