

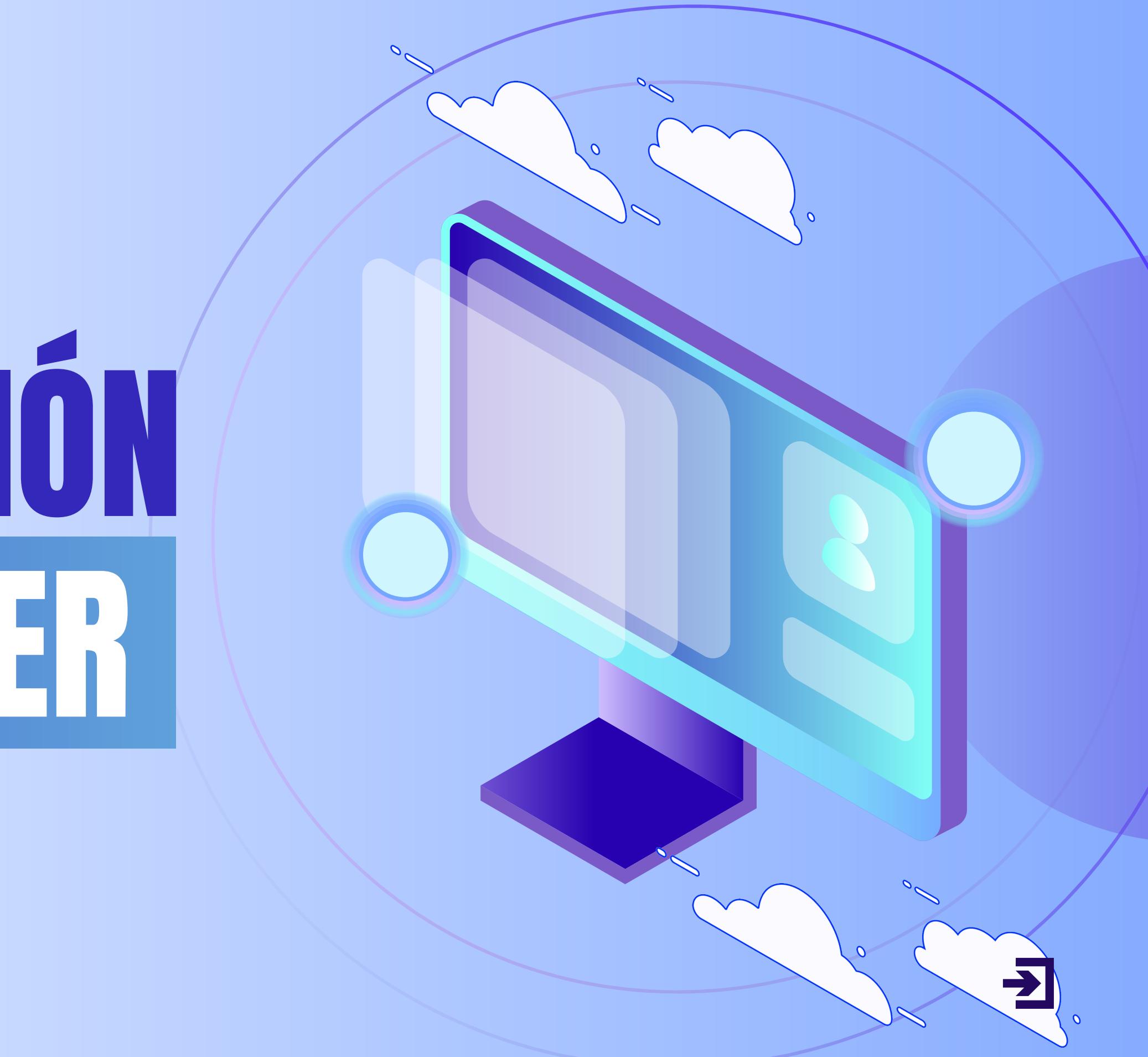


Arquitectura y
Sistemas Operativos

VIRTUALIZACIÓN

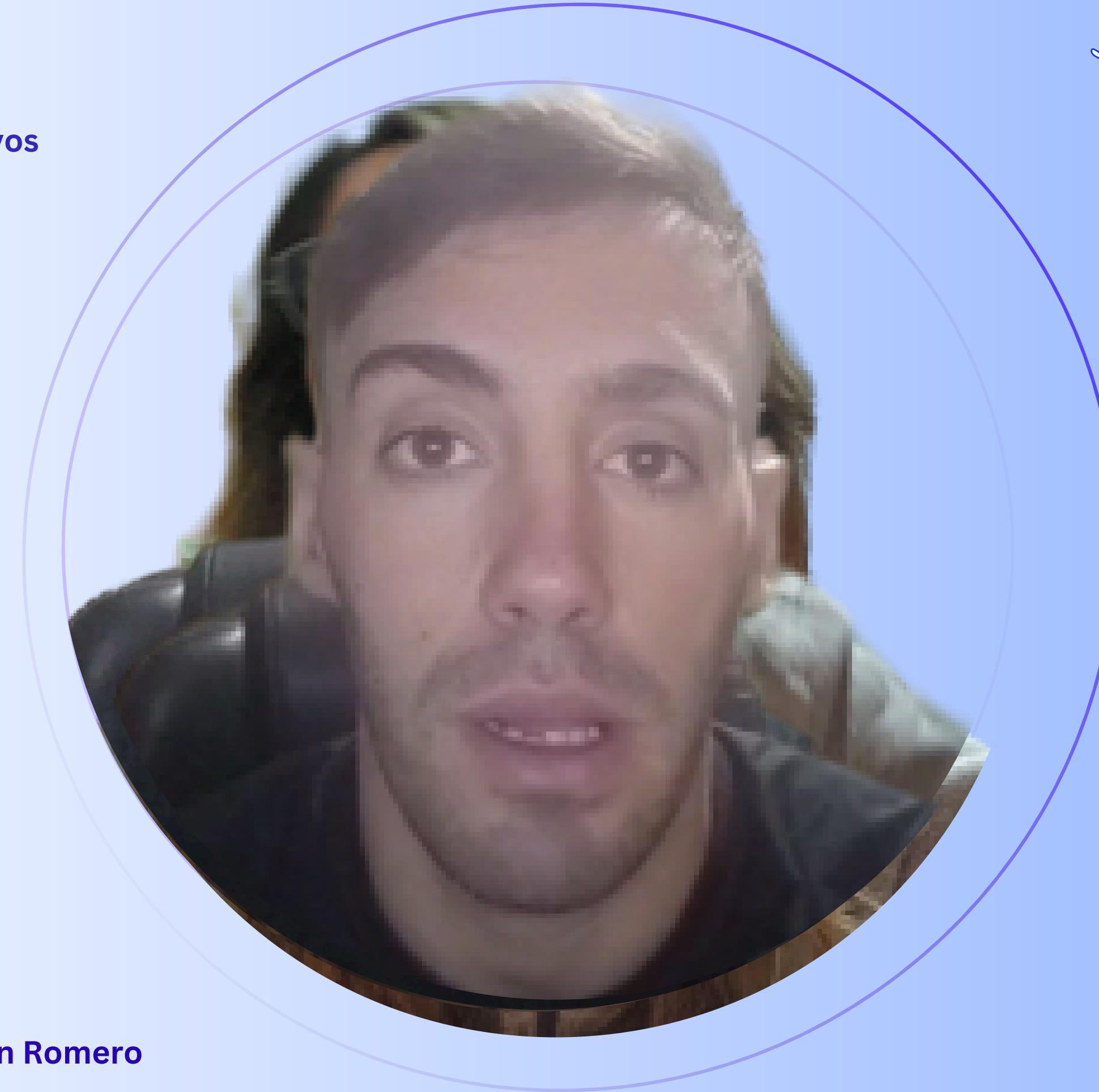
VM VS DOCKER

Daniela Romero - Juan Romero



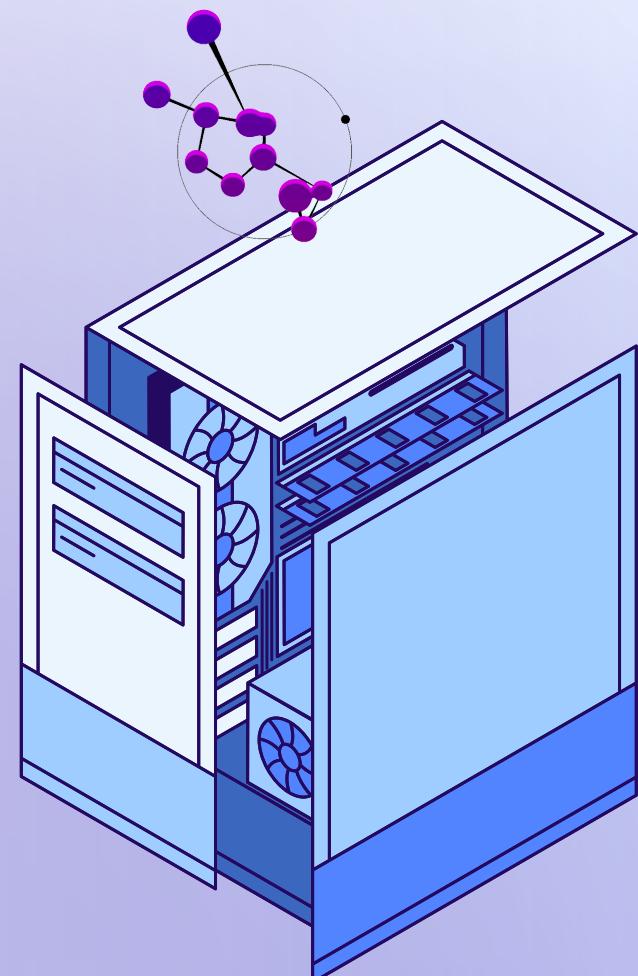


Arquitectura y
Sistemas Operativos



Daniela Romero - Juan Romero

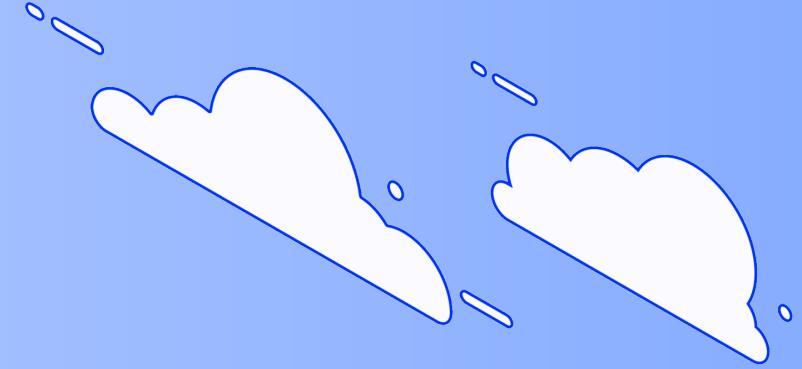
¿POR QUÉ LA VIRTUALIZACIÓN?



- Optimización de Recursos: Mejor uso del hardware.
- Aislamiento: Entornos seguros y separados.
- Flexibilidad y Despliegue: Facilita el desarrollo y la entrega de apps.
- Nube: Base de las infraestructuras cloud.



CONCEPTOS CLAVE



VM

Emula hardware completo.
Incluye Sistema Operativo Invitado.
Más aisladas y pesadas.

DOCKER

Comparte el Kernel del SO host.
Empaquea solo la app y sus dependencias.
Más ligeros y rápidos.



Daniela Romero - Juan Romero





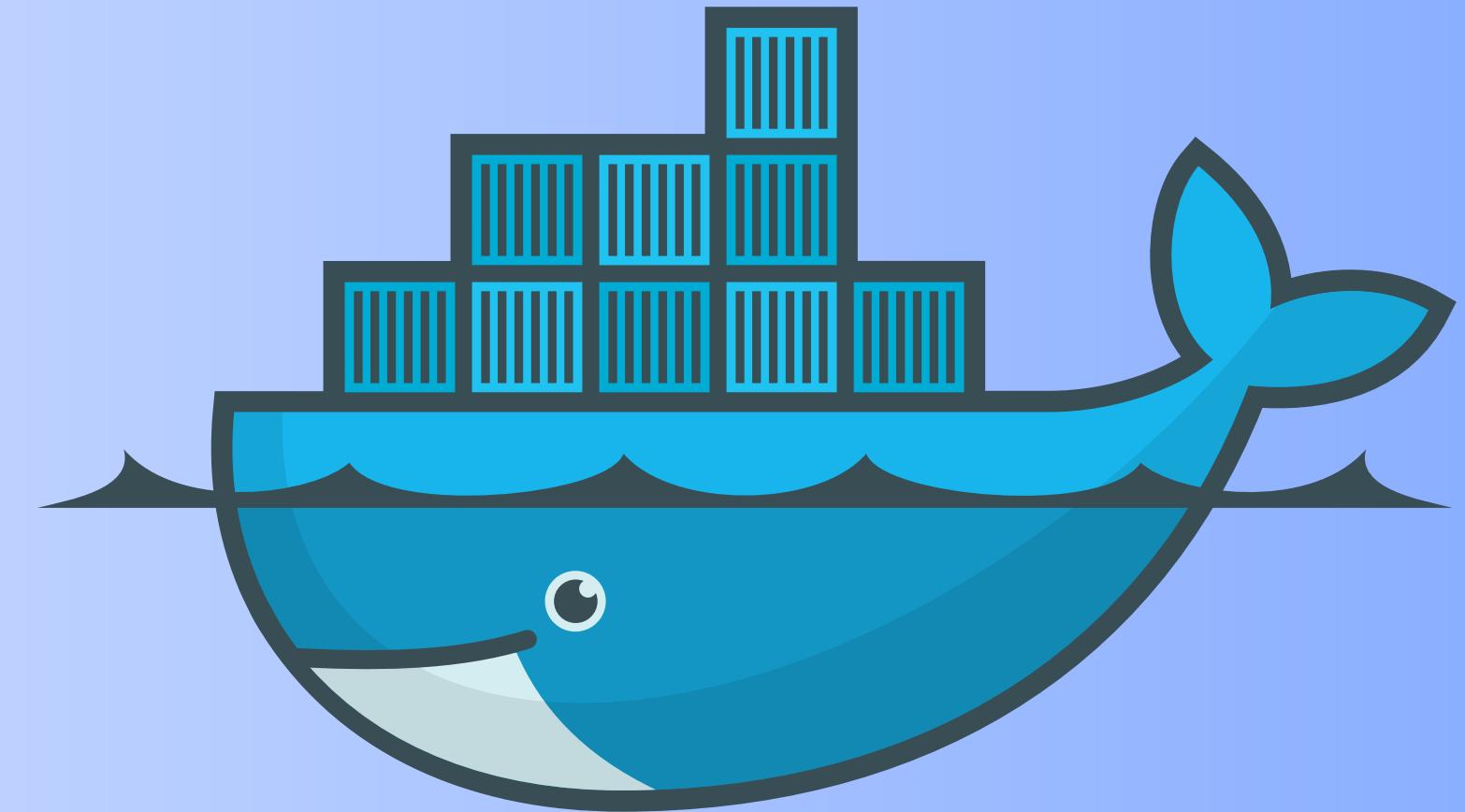
MÁQUINAS VIRTUALES

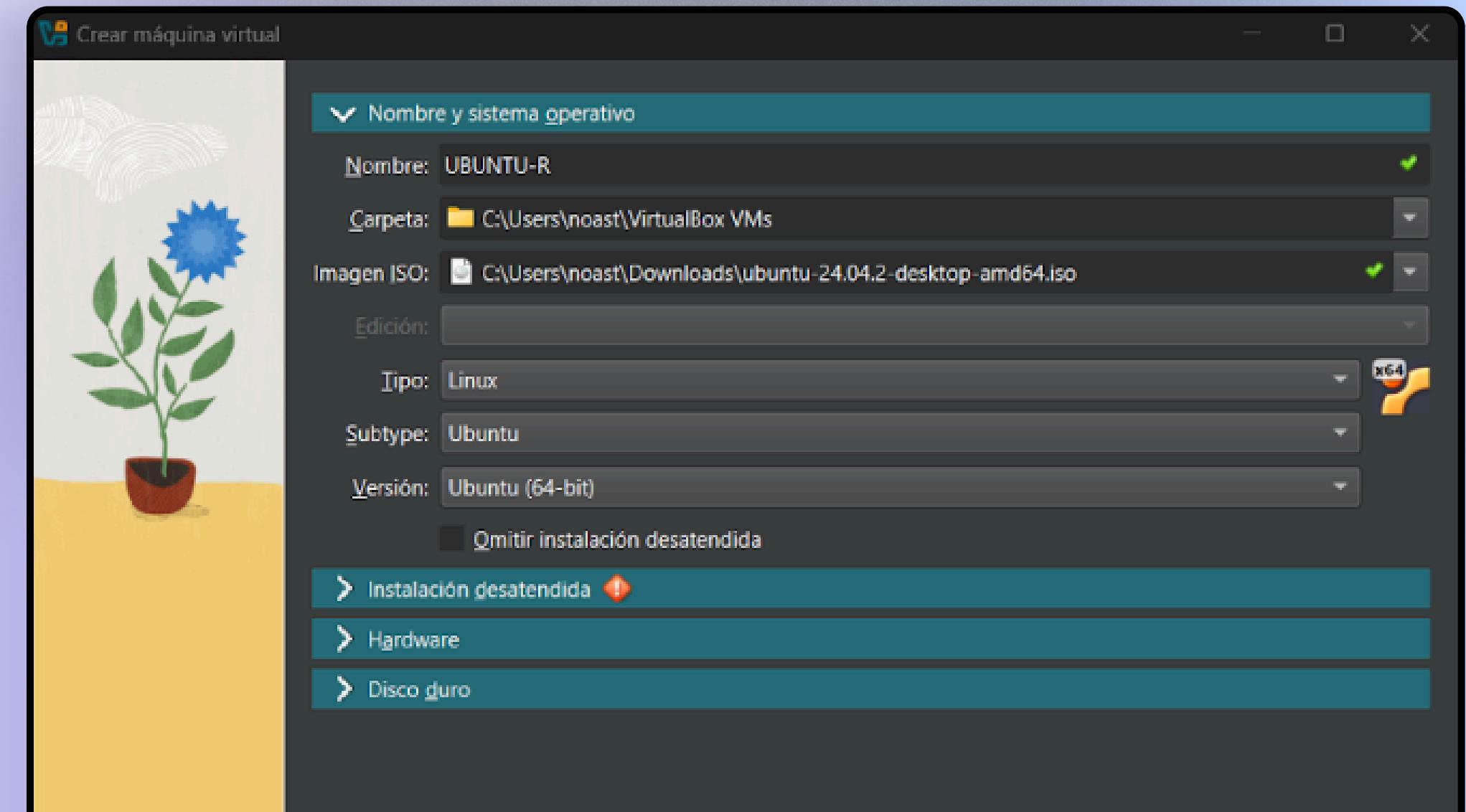
- Emulación de un sistema informático completo (hardware y SO).
- Hipervisor: Software que gestiona y asigna recursos.
 - Tipo 1: Directo sobre hardware
 - Tipo 2: Sobre un SO anfitrión



CONTENEDORES DOCKER

- Definición: Plataforma de código abierto para empaquetar y ejecutar aplicaciones en entornos aislados.
- Componentes Clave:
 - ↳ Imagen: "Receta" con todo lo necesario para la app.
 - ↳ Contenedor: Instancia en ejecución de una imagen.
 - ↳ Dockerfile: Archivo para construir imágenes.



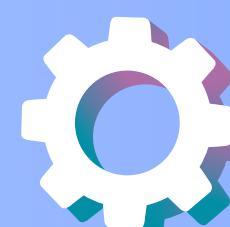


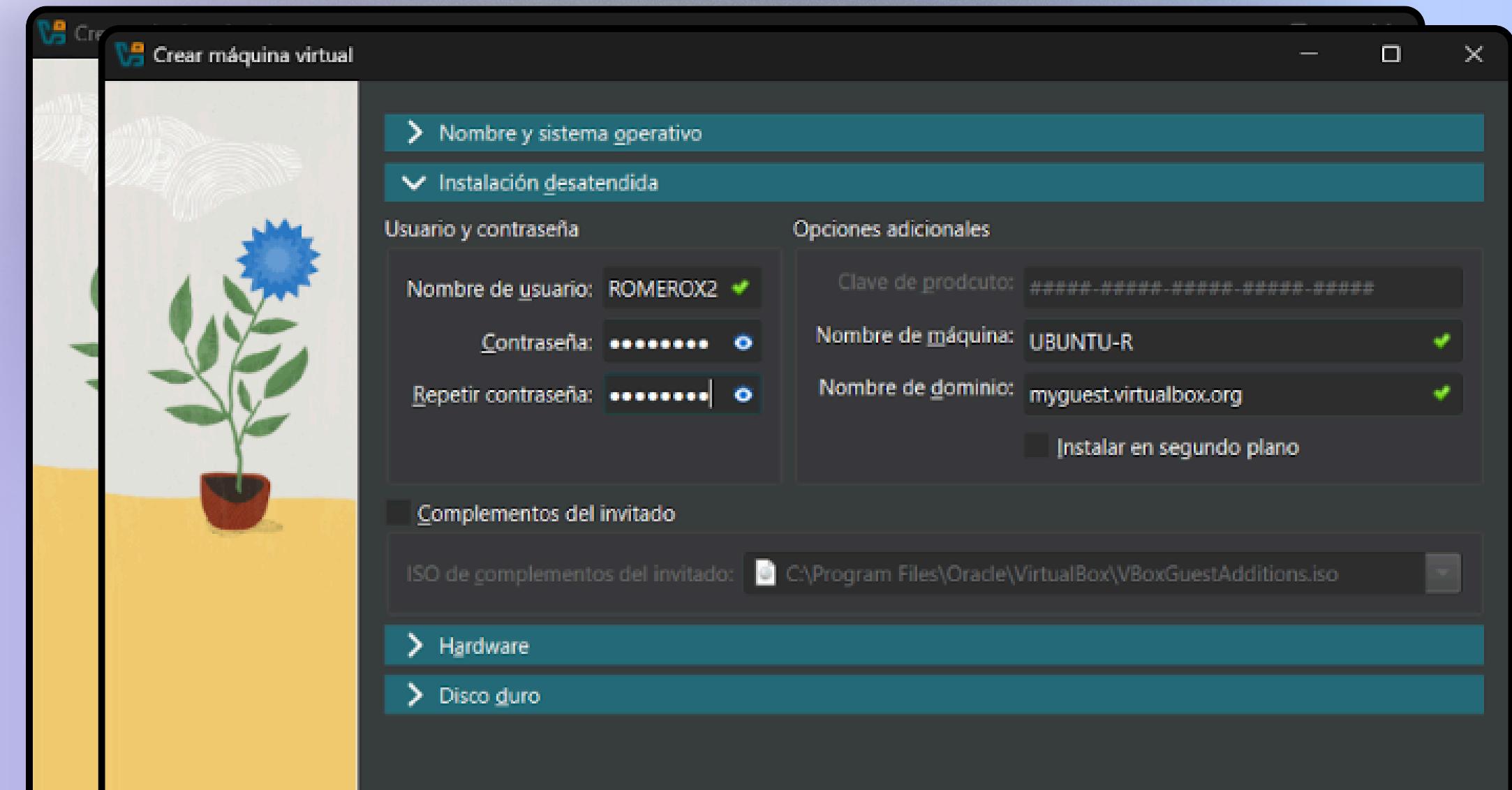
CASO PRÁCTICO: DESPLEGUE EN VM



Creación de MV (VirtualBox, Ubuntu Server).

- Configuraremos la ubicación de nuestra MV



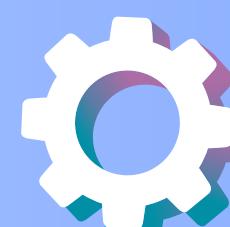


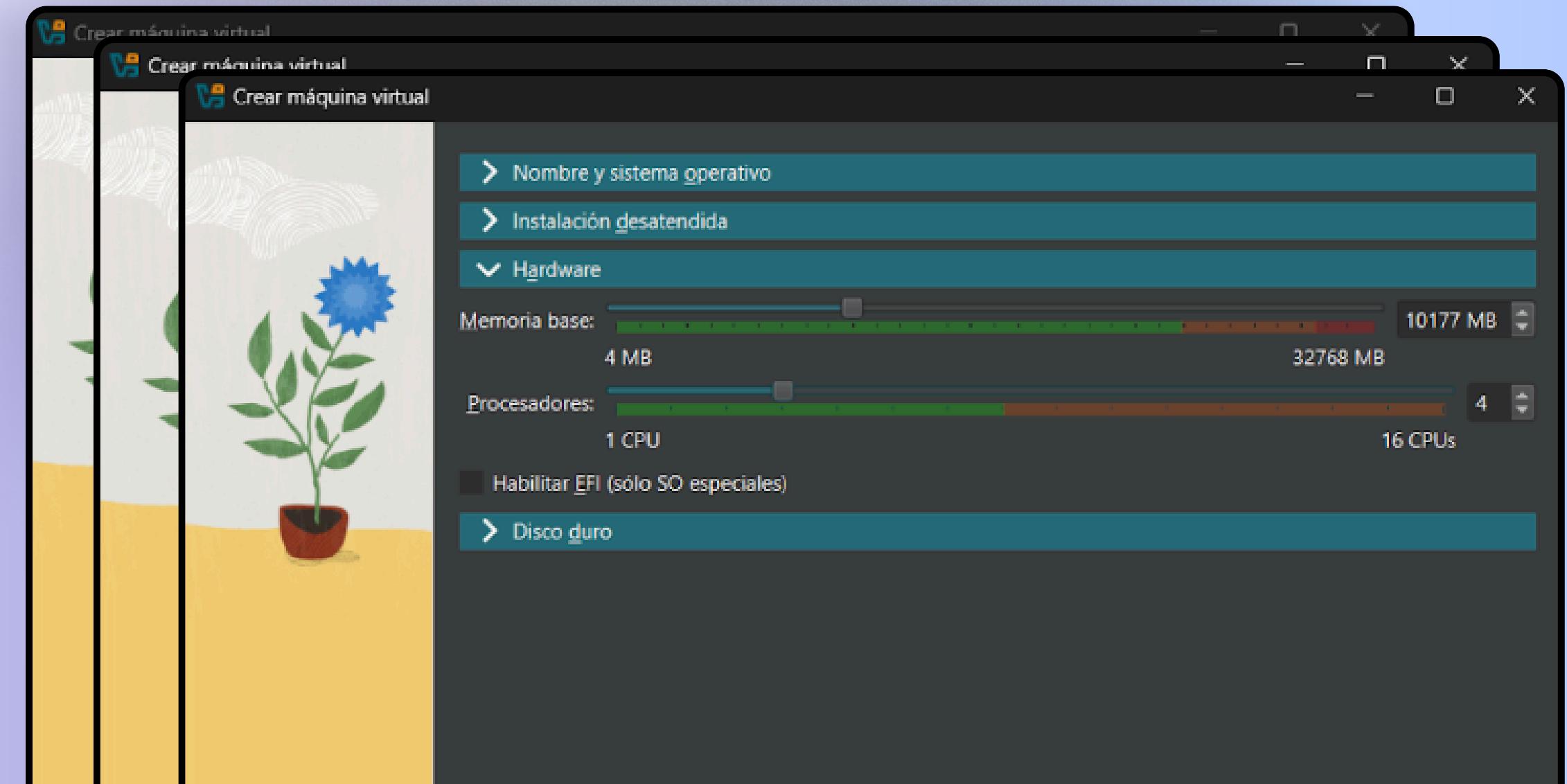
CASO PRÁCTICO: DESPLIEGUE EN VM



Creación de MV (VirtualBox, Ubuntu Server).

- Configuraremos la ubicación de nuestra MV
- Establecemos el usuario y la contraseña



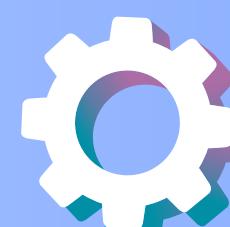


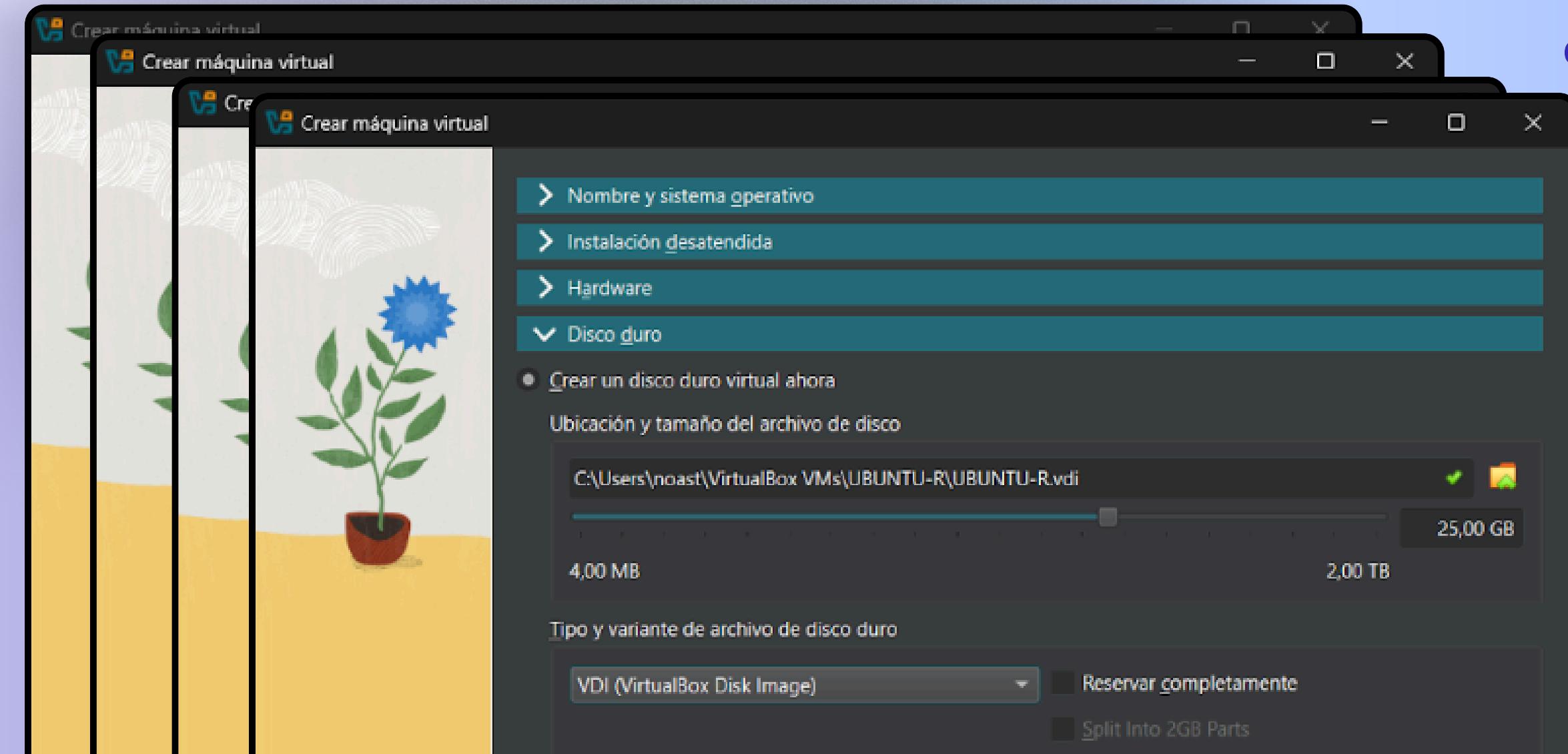
CASO PRÁCTICO: DESPLIEGUE EN VM



Creación de MV (VirtualBox, Ubuntu Server).

- Configuraremos la ubicación de nuestra MV
- Establecemos el usuario y la contraseña
- Designamos RAM y CPUs a nuestra MV



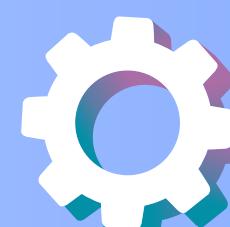


CASO PRÁCTICO: DESPLIEGUE EN VM



Creación de MV (VirtualBox, Ubuntu Server).

- Configuraremos la ubicación de nuestra MV
- Establecemos el usuario y la contraseña
- Designamos RAM y CPUs a nuestra MV
- Determinamos el tamaño del espacio de almacenamiento





CASO PRÁCTICO: DESPLIEGUE EN VM



```
GNU GRUB version 2.12

*Try or Install Ubuntu
Ubuntu (safe graphics)
Test memory

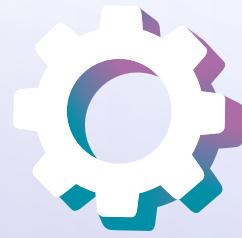
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line.
```

Instalamos el Sistema Operativo

```
sudo apt update
sudo apt upgrade -y
```

Actualizamos los paquetes



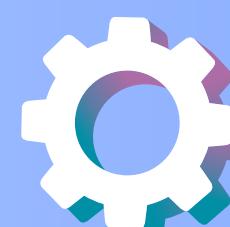


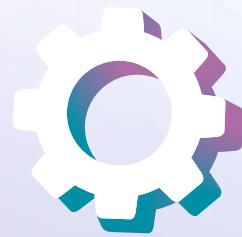
CASO PRÁCTICO: DESPLIEGUE EN VM



Instalamos Python3

```
root@UBUNTU-R:/home/ROMEROX2# apt install python3 -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu2).
python3 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 112 not upgraded.
root@UBUNTU-R:/home/ROMEROX2#
```





CASO PRÁCTICO: DESPLIEGUE EN VM

Desarrollamos la app

```
root@UBUNTU-R:/home/ROMEROX2# mkdir pythonAPP
root@UBUNTU-R:/home/ROMEROX2# cd pythonAPP
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# touch main.py
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# ls
main.py
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# chmod +x main.py
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# ls
main.py
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# nano main.py/
root@UBUNTU-R:/home/ROMEROX2/pythonAPP#
```

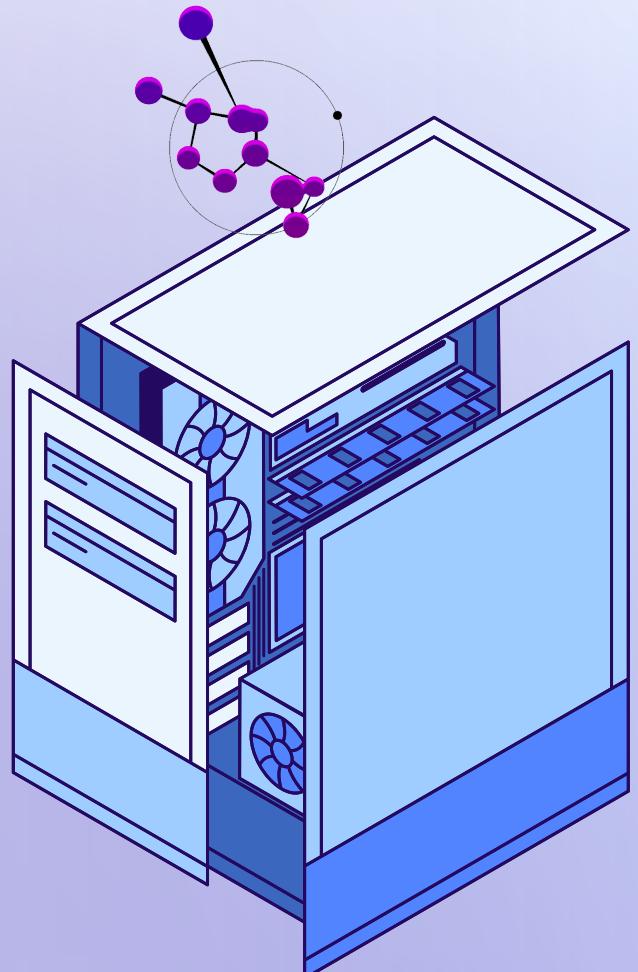
Ejecutamos y validamos

```
root@UBUNTU-R:/home/ROMEROX2/pythonAPP# python3 main.py
Hola desde mi app de python
Este mensaje se ejecuta en una maquina virtual!
root@UBUNTU-R:/home/ROMEROX2/pythonAPP#
```



Daniela Romero - Juan Romero

CASO PRÁCTICO: DESPLIEGUE EN DOCKER



```
└─ PYTHON_APP
    └── Dockerfile
    └── main.py
```

Creamos los archivos necesarios:
el Dockerfile y el main.py

```
usuario@DESKTOP-LB8I6K6 MINGW64 ~/Desktop/utn/ayso/integrador/python_app
● $ touch Dockerfile

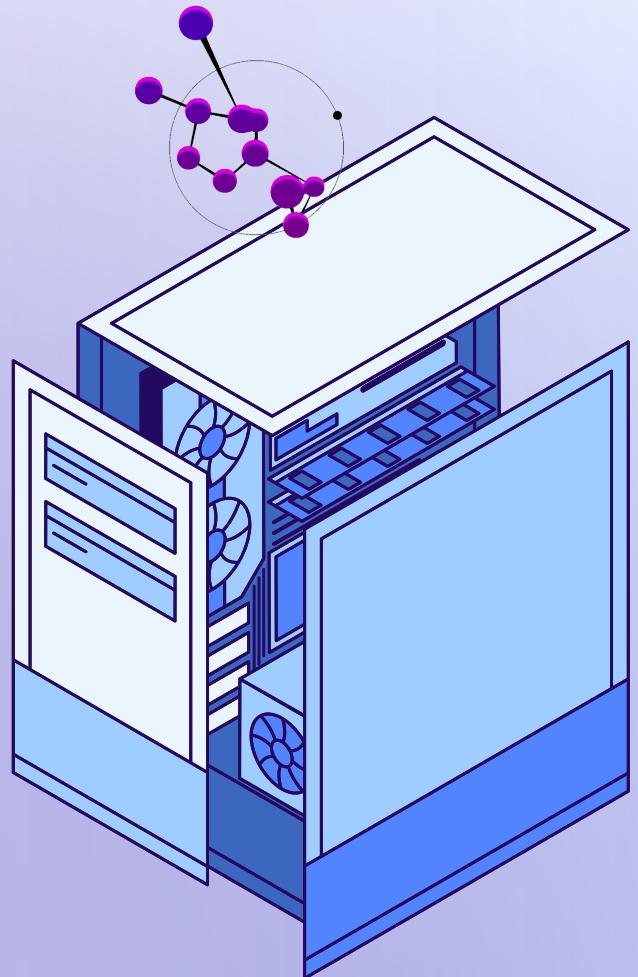
usuario@DESKTOP-LB8I6K6 MINGW64 ~/Desktop/utn/ayso/integrador/python_app
● $ touch main.py
```



Daniela Romero - Juan Romero

CASO PRÁCTICO:

DESPLIEGUE EN DOCKER



```
main.py  X
main.py > ...
1  nombre = input("¿Cuál es tu nombre? ")
2  print(f"Hola {nombre}, ¿cómo estás?")
```

Escribimos el código correspondiente
en cada uno

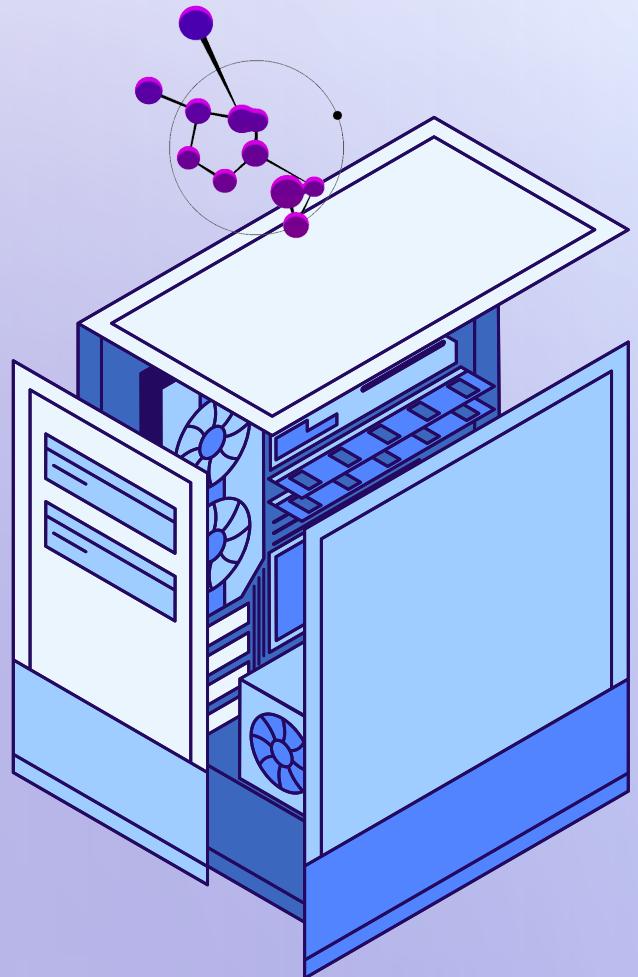
```
main.py
Dockerfile  X
Dockerfile
1  FROM python:3.9-slim-buster
2
3  WORKDIR /user/src/app
4
5  COPY main.py .
6
7  CMD ["python", "main.py"]
```





Daniela Romero - Juan Romero

CASO PRÁCTICO: DESPLIEGUE EN DOCKER



Creamos una imagen del programa:

```
docker build -t my-python-app .
```

Ejecutamos un contenedor con la imagen construida:

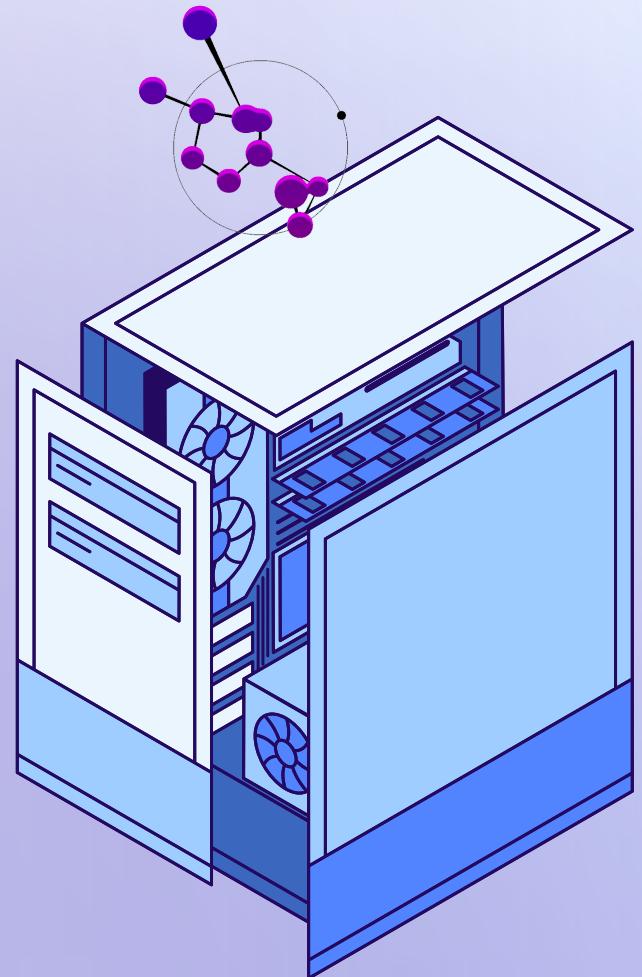
```
PS C:\Users\usuario\Desktop\utn\ayso\integrador\python_app> docker run -it --rm --name my-app -v ./user/src/app my-python-app
¿Cuál es tu nombre? Daniela
Hola Daniela, ¿cómo estás?
PS C:\Users\usuario\Desktop\utn\ayso\integrador\python_app>
```





Daniela Romero - Juan Romero

CASO PRÁCTICO: DESPLIEGUE EN DOCKER



Hacemos cambios al programa:

```
main.py  X  Dockerfile
main.py > ...
1  nombre = input("¿Cuál es tu nombre? ")
2  print(f"Hola {nombre}, ¿cómo estás?")
3  print("Corriste los cambios en tu programa con Docker con éxito.")
```

Ejecutamos un contenedor con la imagen construida anteriormente y se muestran los cambios:

```
PS C:\Users\usuario\Desktop\utn\ayso\integrador\python_app> docker run
● -it --rm --name my-app -v ./user/src/app my-python-app
¿Cuál es tu nombre? Daniela
Hola Daniela , ¿cómo estás?
Corriste los cambios en tu programa con Docker con éxito.
```





COMPARACIÓN

Característica	Máquinas Virtuales	Docker (Contenedores)
Peso/Recursos	Pesadas, mayor consumo de RAM/CPU	Ligeros, eficiente en recursos
Aislamiento	Completo, a nivel de hardware	A nivel de proceso, comparte kernel
Inicio	Lento (segundos/minutos)	Rápido (milisegundos/segundos)
Portabilidad	Menos portable (imagen grande de SO)	Altamente portable (imágenes pequeñas)
Casos de Uso	Entornos heterogéneos, Legacy	Microservicios, CI/CD, desarrollo ágil





RESULTADOS Y DESAFIOS

- Configuración de una maquina virtual desde cero.
- Despliegue exitoso de la app en ambos entornos.
- Creación de imágenes y contenedores en Docker.
- Comprobación de la funcionalidad de volúmenes en Docker.
- Mejora en la comprensión de manejo de contenedores.
- Creación de documentación clara de los pasos realizados, lo que permite su replicación.

- Manejo del ciclo de vida de los contenedores y los cambios registrados en los volúmenes.
- Entendimiento de las rutas en el Dockerfile.
- Interpretar explicaciones de la IA y el material de la catedra, con el objetivo de solventar problemáticas durante la ejecución del programa.





SOLUCIONANDO DESAFIOS

Confusión: ¿Por qué los cambios no se ven reflejados?

Respuesta: Discordancia entre el WORKDIR en el Dockerfile y la ruta utilizada en el script

Solución: ¡El WORKDIR y la ruta del volumen deben coincidir!

Prueba y error con los comandos \$PWD y %cd%

Consultas a la IA

La imagen y el volumen funcionan correctamente





CONCLUSIONES

TÉCNICAS

Control de versiones
del Entorno

Ventaja técnica
para desarrolladores

Experimentación sin
conflictos

Cloud computing

Compatibilidad entre
Plataformas

Optimización de
recursos

Pruebas y
validaciones

Manejo de
dependencias

Simulación de
Infraestructura Real

PERSONALES *(a futuro...)*

Practicar más el uso de
contenedores.

Crear proyectos más grandes.





GRACIAS

 Juan Romero

 Daniela Romero