



Trabajo Práctico Integrador

Virtualización: Máquinas Virtuales vs. Docker

Alumnos: Daniela Nahir Romero (danielanahirromero@gmail.com), Juan Romero (noastranoa@outlook.es)

Materia: Arquitectura y Sistemas Operativos

Profesor: Martín Aristiaran (Comisión 20)

Tutor: David Roco

Fecha de Entrega: 05/06/2025

Índice

1. Introducción.....	2
2. Marco Teórico.....	2
3. Caso Práctico.....	3
3.1. Resolución en MV.....	4
3.2. Resolución en Docker.....	5
4. Metodología Utilizada.....	6
5. Resultados Obtenidos.....	8
6. Conclusiones.....	9
7. Bibliografía.....	10
8. Anexo.....	11



1. Introducción

El presente trabajo toma el tema de virtualización por su relevancia en el desarrollo de software, donde la optimización de los recursos y la escalabilidad son esenciales. Se estudiará este tema desde dos herramientas en este campo: las máquinas virtuales y Docker, con el objetivo de explorar sus funcionalidades.

Para un técnico en programación, la comprensión de la virtualización es indispensable ya que facilita el despliegue de aplicaciones, ofrece flexibilidad en cuanto a los entornos de desarrollo, optimiza el rendimiento del hardware y fomenta un verdadero entendimiento acerca de cómo trabajar efectivamente con servicios que forman una infraestructura en la nube.

Los objetivos que se busca alcanzar son: comprender los conceptos fundamentales de la virtualización, diferenciar entre MV y Docker, analizar ventajas y desventajas de cada tecnología, y demostrar una implementación básica del uso de MV y de Docker.

2. Marco Teórico

A continuación se detallarán definiciones y conceptos relevantes para la comprensión de la virtualización que fueron necesarios para la realización de este trabajo.

- Virtualización: se refiere a una tecnología que permite crear versiones virtuales de recursos informáticos, como sistemas operativos, servidores, recursos de red, o dispositivos de almacenamiento. Es utilizada para poder aislar aplicaciones del hardware y las características del host, para poder desplegarlas y probar su funcionamiento de manera independiente. Promueve así un uso eficiente del hardware ya que se puede utilizar un mismo equipo para múltiples cargas de trabajo, sin generar conflictos.

- Máquinas virtuales: son emulaciones de sistemas informáticos, incluyendo un sistema operativo “invitado”, que se ejecutan sobre un solo hardware físico, pero



funcionan como equipos diferentes e independientes. Estas máquinas requieren un hipervisor que gestiona los recursos del host y los asigna a las MV.

- Hipervisor de Tipo 1: se instala sobre el hardware físico sin un sistema operativo intermedio. Ofrece un alto rendimiento.
- Hipervisor de Tipo 2: se ejecuta como una aplicación sobre un sistema operativo anfitrión. Es más sencillo para desarrollar.
- Docker: esta plataforma de código abierto ofrece otra posibilidad de virtualización. En el caso de Docker, el kernel del sistema operativo se comparte entre contenedores que empaquetan aplicaciones y sus dependencias. Por ello, Docker es más ligero y eficiente.
- Contenedor: es una instancia de ejecución de una imagen Docker. Se puede iniciar, detener y eliminar. Es un entorno aislado para ejecutar una aplicación.
- Imagen: es una “plantilla” o “receta” que contiene las instrucciones para crear un contenedor Docker. Incluye todo lo necesario para ejecutar una aplicación.
- Dockerfile: Es un archivo de texto que tiene configuraciones para construir una imagen, donde se define, entre otras cosas, las dependencias, comandos, etc.

3. Caso Práctico

El caso a desarrollar en este trabajo es la creación de una pequeña aplicación web en Python que muestre un mensaje en con y se despliegue en dos entornos de virtualización: una máquina virtual y otra en un contenedor de Docker. El objetivo es explorar las funcionalidades básicas de ambas tecnologías, implementarlas y comparar sus procesos de configuración y ejecución.



3.1. Resolución en MV

El procedimiento para implementar el programa en una máquina virtual fue el siguiente:

1. Creación e inicialización de la máquina virtual en Oracle VM VirtualBox (*Anexo: figuras 1-5*) con las siguientes características:

Nombre:: UBUNTU-R

Almacenamiento: 25gb

Memoria RAM: 10177mb

CPUs: 4

2. Instalación del sistema operativo “invitado” Ubuntu Server dentro de la MV (*Anexo: figuras 6-7*).

3. Actualización de los paquetes del sistema (*Anexo: figuras 8-9*). mediante los comandos:

```
sudo apt update
```

```
sudo apt upgrade
```

4. Instalacion de Python3 (*Anexo: figura 10*) con los comandos:

```
sudo apt install python3 -y
```

5. Desarrollo de la app: se creó un directorio pythonAPP dentro de /home/ROMEROX2/ y se navegó a él. Luego se creo un archivo llamado [main.py](#) y se lo editó con nano (*Anexo: figuras 11-14*). Los comandos a continuación:

```
mkdir pythonAPP
```

```
cd pythonAPP
```

```
touch main.py
```

```
nano main.py
```



6. Ejecución de la app:

```
python3 main.py
```

7. Validación de funcionamiento: se logró observar que en la consola efectivamente se mostraban los mensajes (*Anexo: figura 15*):

"¡Hola desde mi app de python"

"Este mensaje se ejecuta en una máquina virtual!".

3.2. Resolución en Docker

Resolver el problema con Docker requirió los siguientes pasos:

1. Creación de un Dockerfile y un archivo [main.py](#) en el mismo directorio (*Anexo: figuras 16-18*):

(Contenido en el Dockerfile)

```
FROM python:3.9-slim-buster
WORKDIR /user/src/app
COPY main.py .
CMD ["python", "main.py"]
```

(Contenido en [main.py](#))

```
nombre = input("¿Cuál es tu nombre? ")
print(f"Hola {nombre}, ¿cómo estás?")
```

2. Creación de una imagen (*Anexo: figura 19*) del programa con el comando:

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

3. Ejecución del contenedor con la imagen creada (*Anexo: figura 20*), con parámetros opcionales como -v, que en este caso se usa para permitir que los cambios en [main.py](#) se vean reflejados en la imagen si el contendor se vuelve a



correr:

```
docker run -it --rm my-app -v ./user/src/app my-python-app
```

4. Validación de funcionamiento (*Anexo: figura 20*): se logró observar que en la consola efectivamente se solicitaba el nombre al usuario y se mostraba el mensaje:

"¡Hola Daniela, cómo estás?"

5. Se producen cambios en main.py para verificar que la incorporación del volumen funcione (*Anexo: figura 21*). Se agrega:

```
print(f"Corriste los cambios en tu programa con Docker con éxito")
```

6. Se vuelve a correr el contenedor y se verifica su funcionamiento al observar lo impreso en la consola (*Anexo: figura 21*):

"Corriste los cambios en tu programa con Docker con éxito."

4. Metodología Utilizada

Para realizar este trabajo se comenzó por realizar una introducción a los temas del trabajo y sus conceptos relacionados. Para ello se utilizaron el material de la cátedra y los videos de los profesores, así como la ayuda de la inteligencia artificial para despejar dudas y lograr una comprensión profunda sobre el tema.

En la etapa de diseño del problema, se tomaron en cuenta los ejercicios resueltos por los profesores en los videos y los modelos de resolución de trabajos integradores. Esto incluye el diseño de una aplicación sencilla en Python, y la instalación y configuración de los entornos. De esta forma, se creó un script Python en un archivo llamado [main.py](#) que imprime un mensaje en la consola. (Nota: la aplicación fue luego modificada para incluir una pequeña interacción con el usuario para que este ingrese datos, lo cual sirvió para probar cómo los cambios en Docker



se pueden manejar con volúmenes.)

Para la configuración de los entornos se siguieron los siguientes pasos:

- Máquina Virtual en VirtualBox: se especificaron las características de la máquina virtual (RAM, CPU, SO, etc), se instaló Python3, se crearon un directorio y un archivo .py para crear la app (todo desde la consola de bash), se utilizó nano para modificar el archivo, y por último se lo ejecutó.
- Contenedor en Docker: Luego de instalar Docker en la pc, se procedió a crear a través de comandos un directorio, un archivo .py para el código Python y el Dockerfile. Desde VSC se procedió a modificar el archivo Python y el Dockerfile (con las configuraciones para la creación de la imagen), se creó la imagen y se ejecutó el contenedor con comandos que incluyen --rm, -it, y -v para que se elimine luego de ejecutarse, sea iterativo y maneje volúmenes de datos, respectivamente.

Para ambos entornos se ejecutó el script y se validó que el funcionamiento sea el esperado.

Se utilizaron las siguientes herramientas:

- VSC
- Python3
- Docker Desktop
- Hipervisor: Oracle VM VirtualBox con Ubuntu Server
- Bash y Powershell
- Terminal/Línea de comandos
- IA: Gemini

El trabajo fue resuelto de manera colaborativa por ambos integrantes, distribuyendo las tareas, en líneas generales, de la siguiente manera:



- Daniela Romero: responsable de la redacción del marco teórico y la descripción del diseño del programa, la edición del video, y la creación y ejecución del programa en el entorno de Docker.
 - Juan Romero: responsable de la redacción de la introducción y la sección de resultados obtenidos, la creación del anexo y la creación y ejecución del programa en la Máquina virtual.
 - Ambos integrantes se reunieron a través de videollamadas de Discord y editaron de manera conjunta lo previamente redactado individualmente utilizando la plataforma de Google Docs. A su vez, utilizaron la plataforma de videollamadas para realizar la creación y ejecución de los programas en vivo, permitiendo esto que ambos estudiantes participaran en la solución de los desafíos técnicos que a medida que iba surgiendo.
-

5. Resultados Obtenidos

Durante el desarrollo del caso práctico, se logró ejecutar correctamente una aplicación de Python en ambos entornos de virtualización: máquinas virtuales y contenedores Docker.

Se realizaron distintas pruebas para validar el uso de volúmenes en Docker, utilizando los siguientes comandos:

- En Linux:

```
$ docker run -it --rm --name my-app -v "$PWD":/usr/src/app my-python-app
```

- En windows:

```
docker run -it --rm --name my-app -v "%cd%":/usr/src/app my-python-app
```

Uno de los principales problemas detectados fue la diferencia entre el directorio de trabajo (WORKDIR) específicamente en el Dockerfile (al cual en un principio habíamos



nombrado `/app`) y el directorio en el que se montaba el volumen (ruta que tomamos del material audiovisual de la cátedra: `/user/src/app`). Esto provocaba que, al ejecutar el contenedor, se corriera el **main.py** de la imagen y no el de la máquina local.

Esto llevó a que, al no producir el volumen correctamente, ni tampoco a reconstruir la imagen tras modificar el archivo **main.py**, los cambios locales no se vieron reflejados en el contenedor. Intentamos resolverlo ejecutando:

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

Gracias a estas pruebas, se comprendió mejor como Docker gestiona los archivos entre el sistema anfitrión y el contenedor, y se logró corregir los errores para que el programa se ejecute de manera esperada.

Durante esta fase, se llevaron a cabo los siguientes casos de prueba:

- Prueba de ejecución sin volumen: se utilizó el **main.py** incorporado en la imagen.
- Prueba con volumen correctamente montado: se logró reflejar los cambios en tiempo real.
- Prueba de rutas relativas/absolutas y errores por no coincidir **WORKDIR** y volumen.

Los siguientes errores fueron corregidos durante el proceso:

- Problemas de rutas en Windows y Linux.
- Error al no reconstruir la imagen.
- Desalineación entre **WORKDIR** y la ruta del script en el contenedor.

6. Conclusiones

Este trabajo permitió al grupo comprender las bases técnicas de dos herramientas fundamentales en el mundo del desarrollo: las máquinas virtuales y los contenedores Docker.



Aprendimos cómo se ejecutan aplicaciones en entornos aislados, como se montan volúmenes en Docker, y la importancia de entender las rutas, el contexto de construcción y la persistencia de datos, como por ejemplo cuando se detiene o elimina un contenedor.

Possibles Mejoras o Extensiones

- Realizar más ejercicios prácticos, con ejemplos más complejos y cercanos a escenarios reales de producción.
- Explorar el uso de Docker Compose para manejar múltiples contenedores.
- Ampliar la práctica con redes y persistencia en contenedores.

Dificultades Enfrentadas

- Errores técnicos como rutas mal configuradas o imágenes desactualizadas.
- Situaciones en las que fue necesario borrar todo y volver a empezar, algo que si bien es frustrante, permitió aprender desde la experiencia.

Estas dificultades se resolvieron con el apoyo de la IA, biográfica técnica, videos explicativos y colaboración entre los integrantes del grupo.

Comparacion de Metodos:

- Máquinas virtuales (como VirtualBox): Permite emular un sistema operativo completo, útil para pruebas pesadas o de bajo nivel.
- Docker: Es ideal para desarrollo ágil de aplicaciones, al ser más liviano, rápido y fácilmente portable.

Ambas herramientas son valiosas. Conocerlas y saber cuándo usar una u otra fue uno de los aprendizajes más importantes del trabajo.

7. Bibliografía

- Docker Docs. (s.f). Docker Documentation. Recuperado el 1 de Junio de 2025 de <https://docs.docker.com/>



- Oracle. (s.f.). Oracle VM Virtual Box Documentation. Recuperado el 2 de junio de 2025 de <https://www.virtualbox.org/manual/UserManual.html>
- Universidad Tecnológica Nacional (UTN). 2025. Arquitectura y Sistemas Operativos. Material de Cátedra de la Tecnicatura en Programación a Distancia.

8. Anexo

A continuación se adjuntan:

- el enlace al video explicativo: [ver aquí](#).
- las capturas de pantallas referenciadas en secciones anteriores.

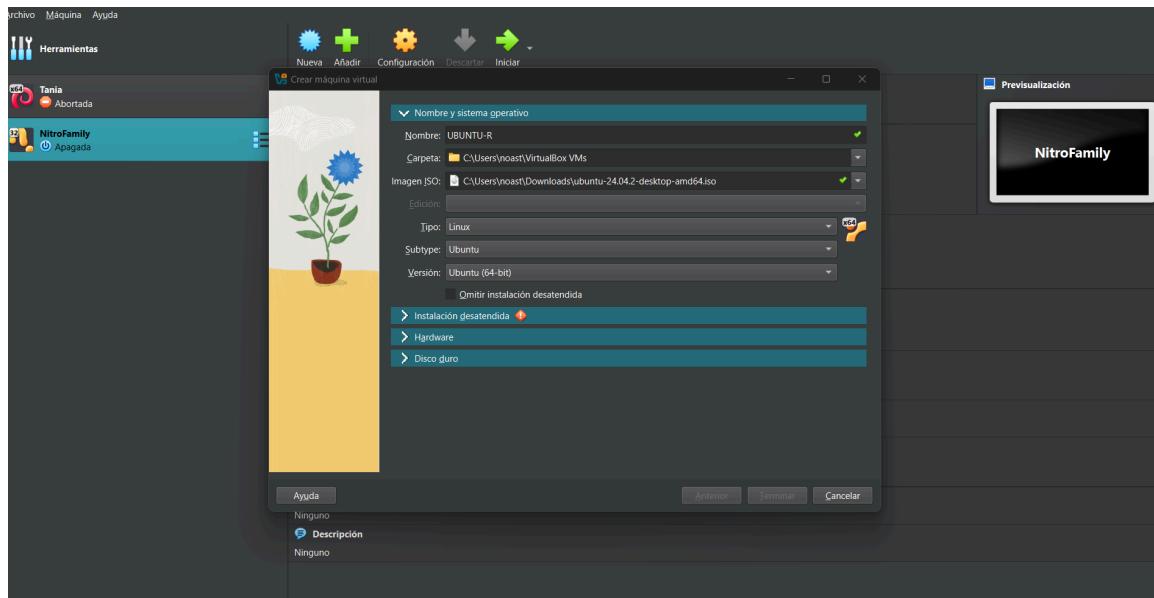


Figura 1: Creación de la máquina virtual en Oracle VM VirtualBox: Nombre y sistema operativo

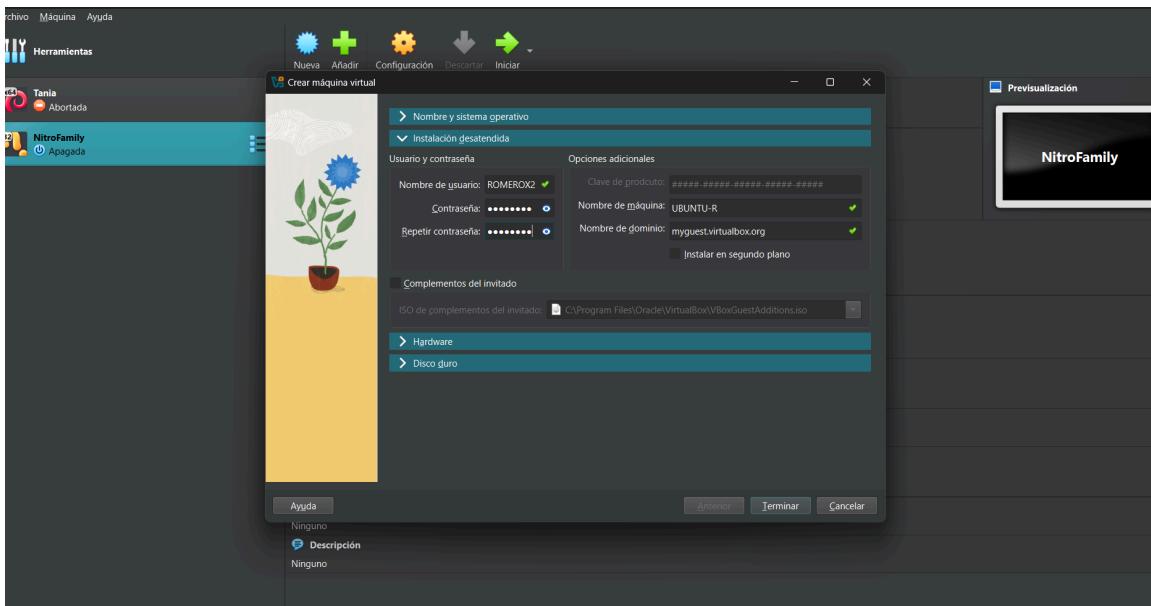


Figura 2: Creación de la máquina virtual en Oracle VM VirtualBox: Instalación desatendida

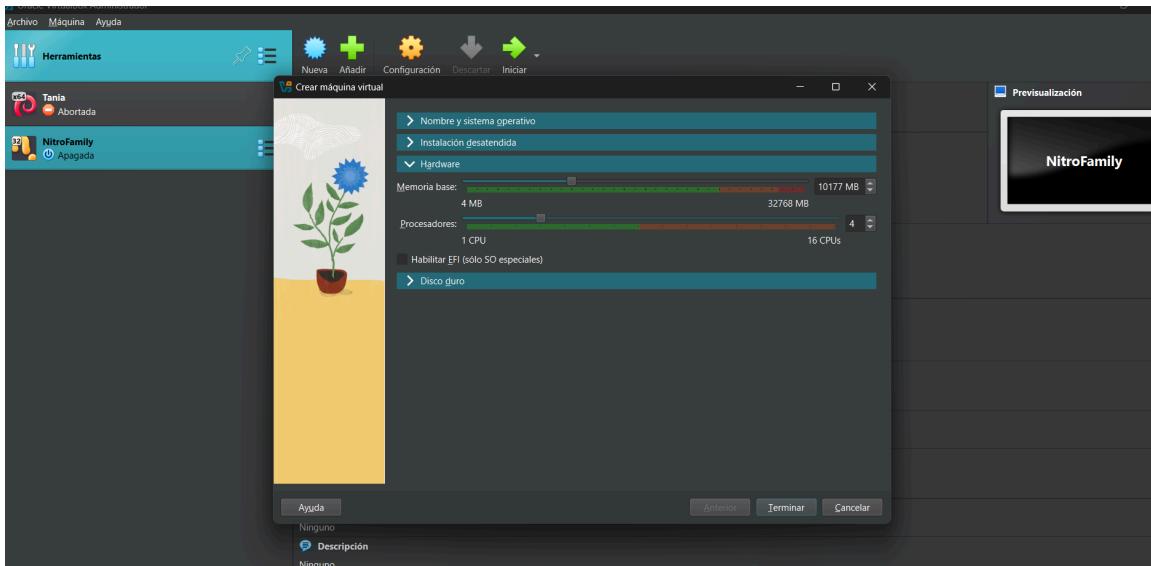


Figura 3: Creación de la máquina virtual en Oracle VM VirtualBox: Hardware

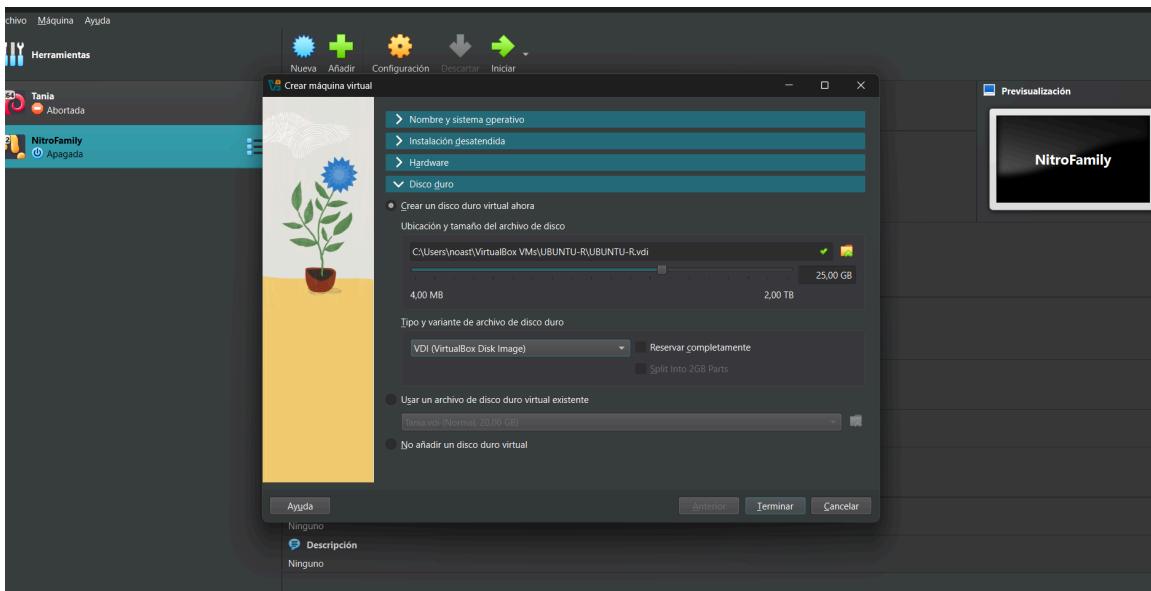


Figura 4: Creación de la máquina virtual en Oracle VM VirtualBox: Disco duro

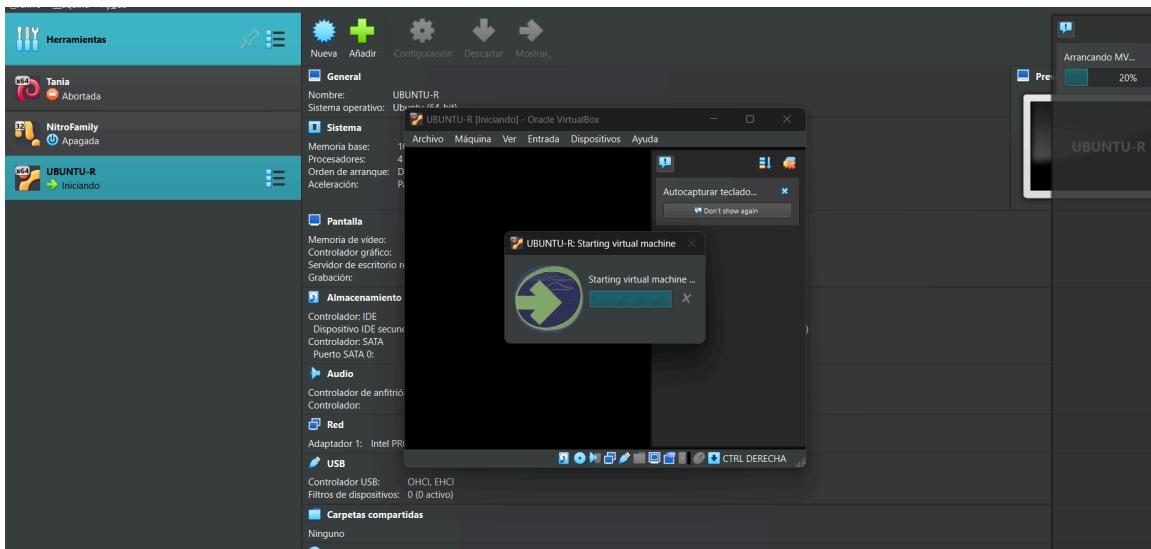


Figura 5: Creación de la máquina virtual en Oracle VM VirtualBox: Inicialización

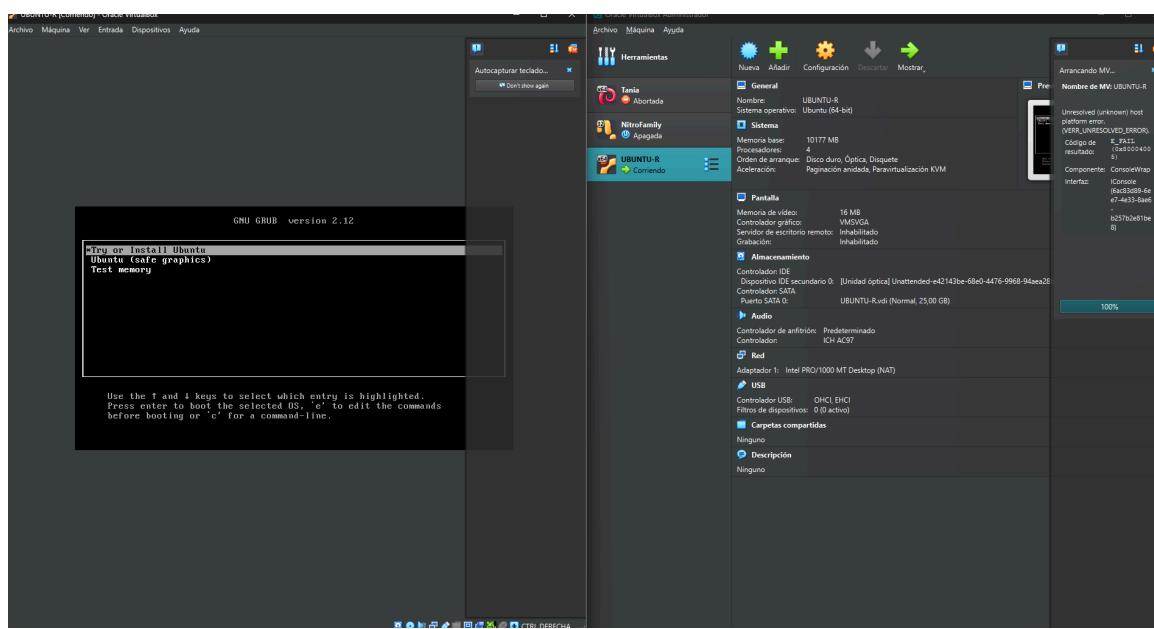


Figura 6: Instalación del sistema operativo “invitado” Ubuntu Server (1)

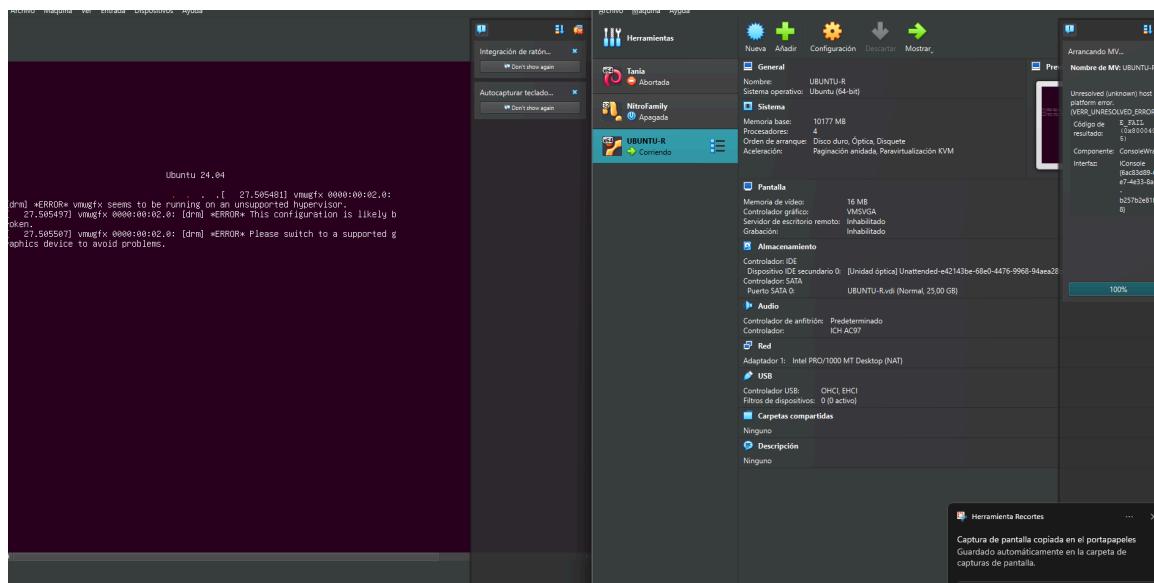


Figura 7: Instalación del sistema operativo “invitado” Ubuntu Server (2)

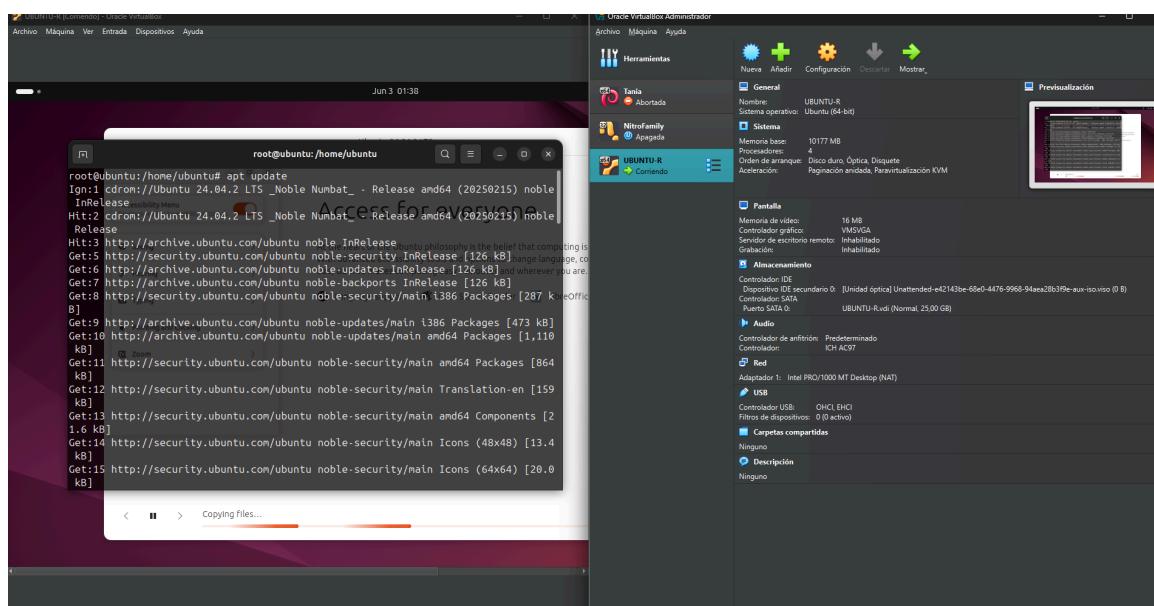


Figura 8: Actualización de los paquetes del sistema (1)

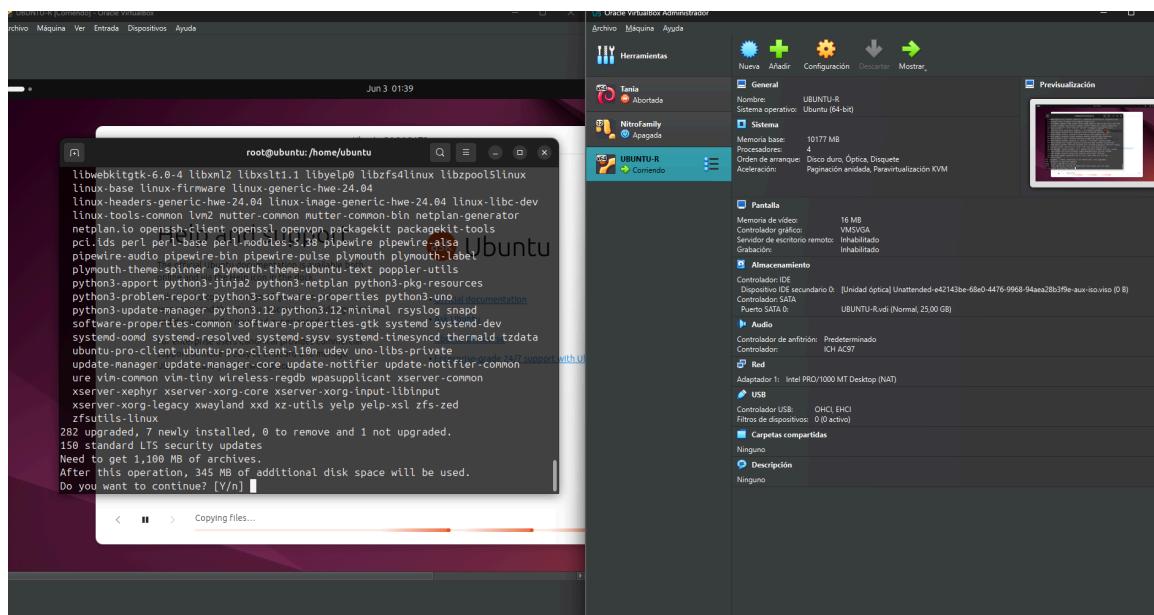


Figura 9: Actualización de los paquetes del sistema (2)

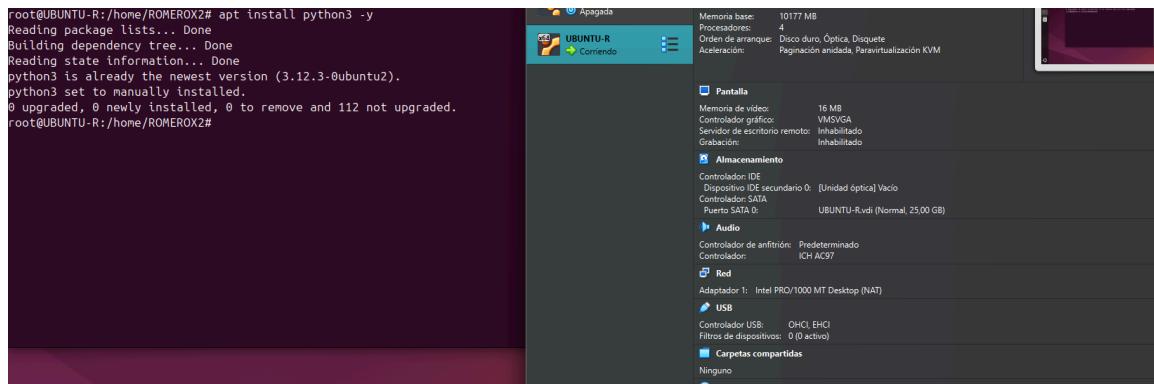


Figura 10: Instalacion de Python3

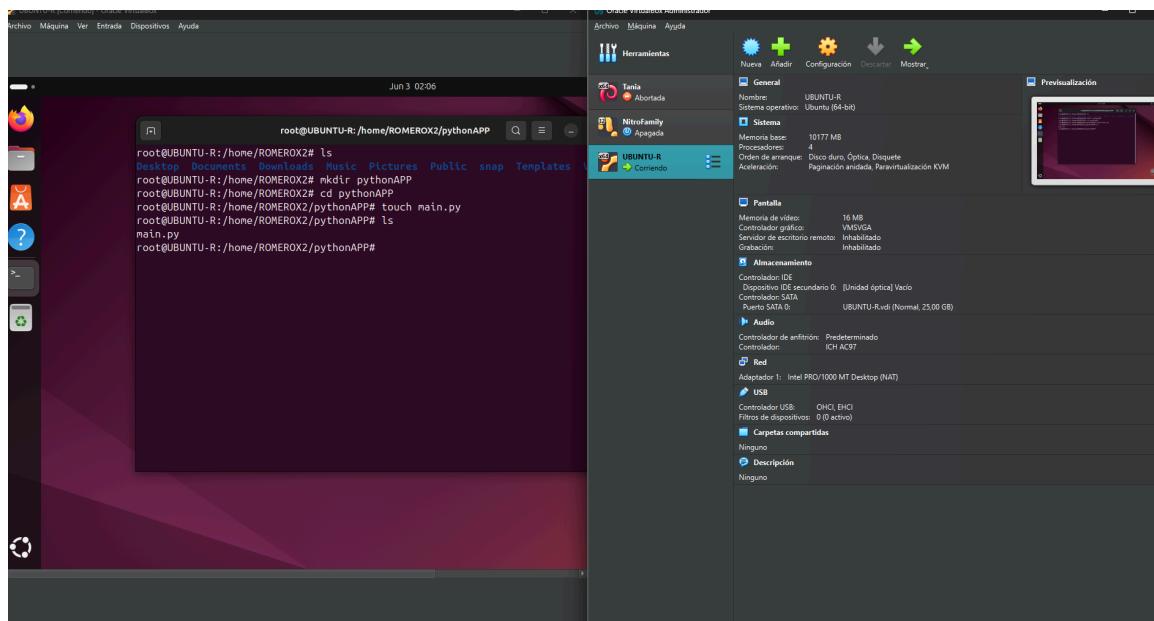


Figura 11: Desarrollo de la app (1)

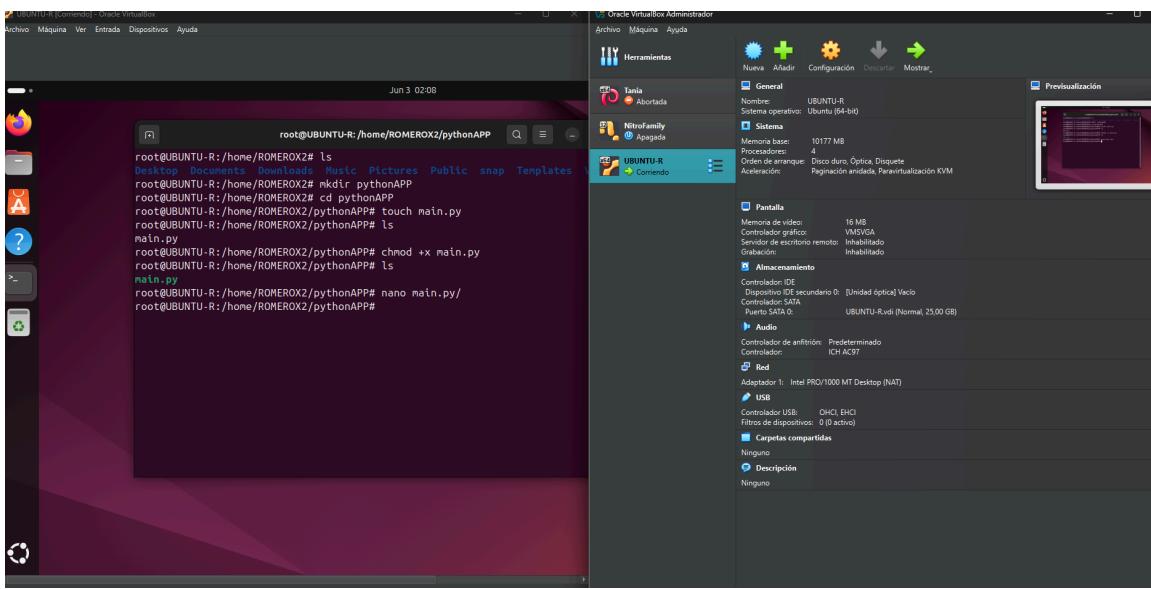


Figura 12: Desarrollo de la app (2)

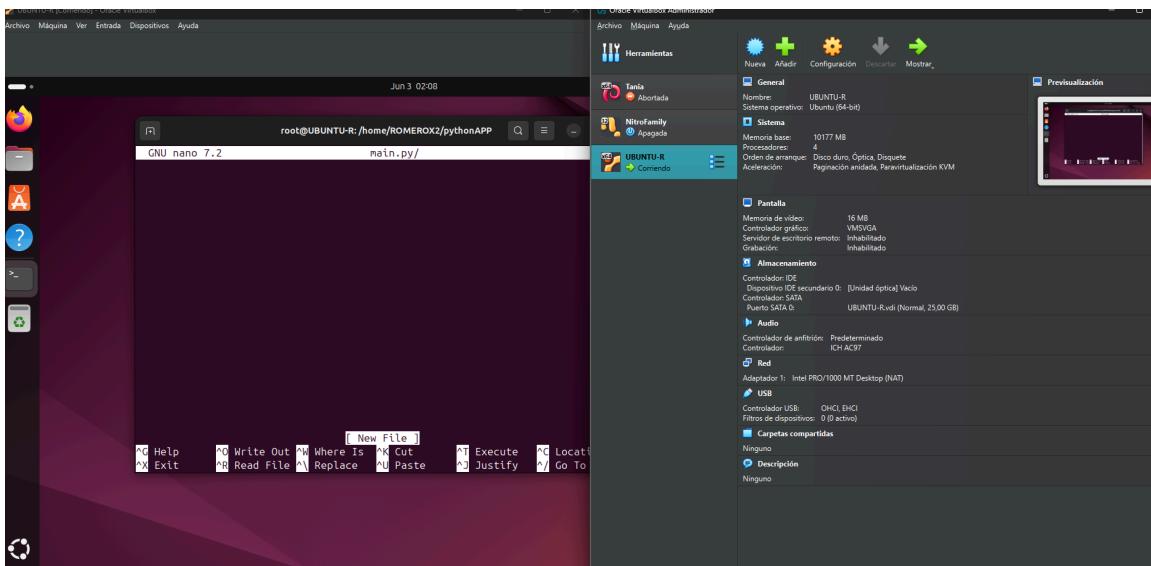


Figura 13: Desarrollo de la app (3)

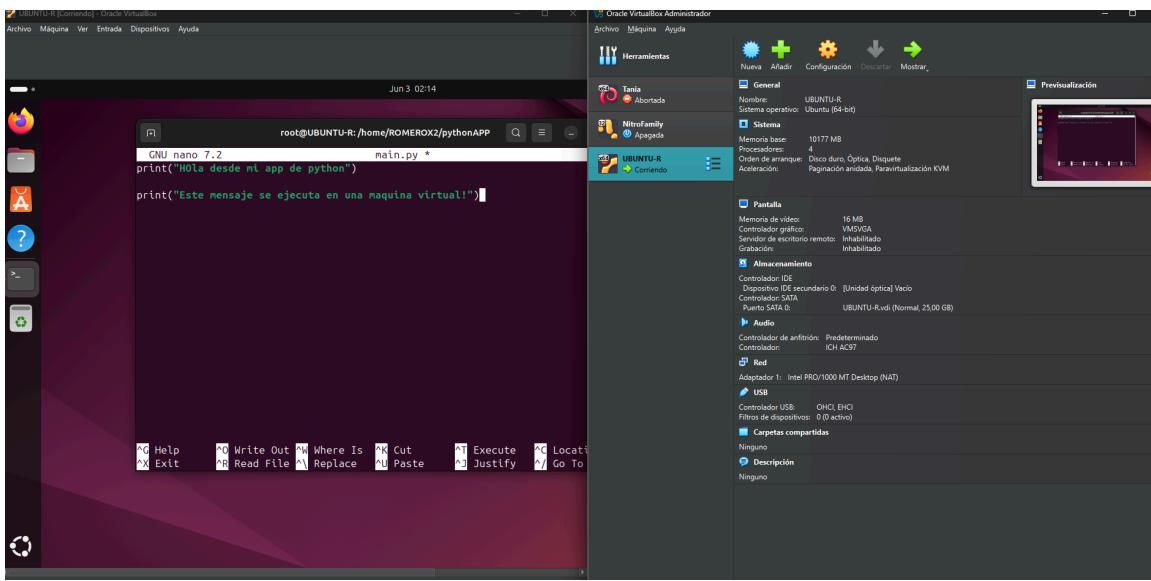


Figura 14: Desarrollo de la app (4)

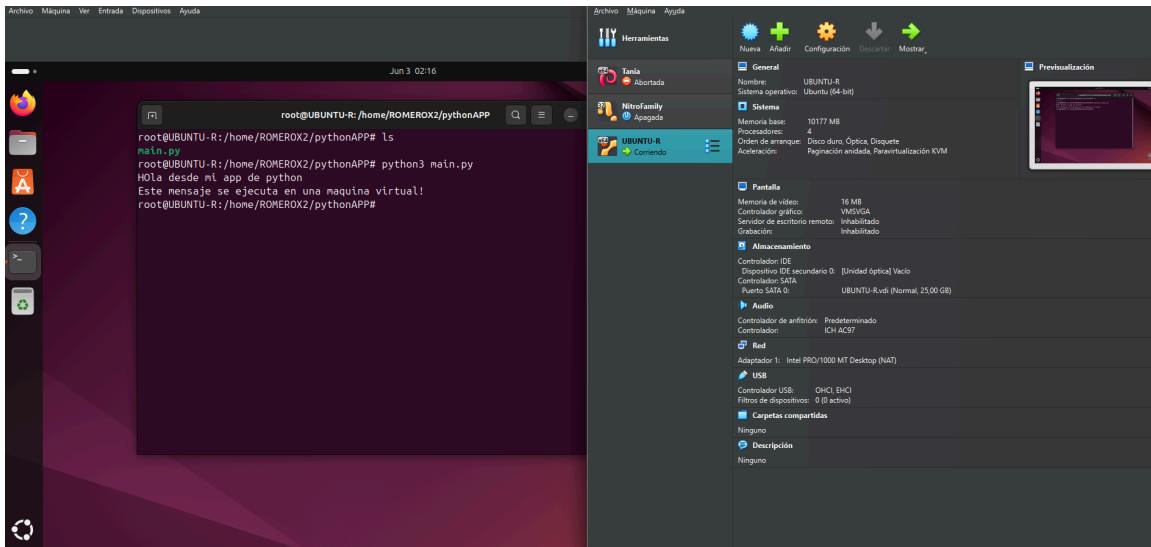


Figura 15: Ejecución y validación de la app

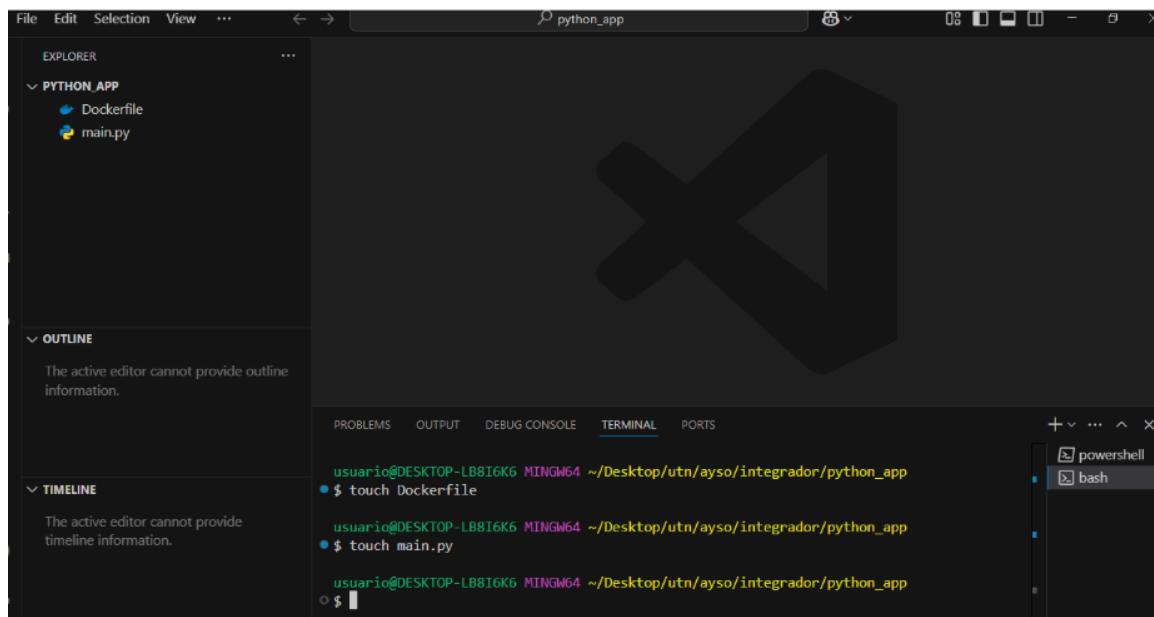


Figura 16: Creación del Dockerfile y main.py (1)

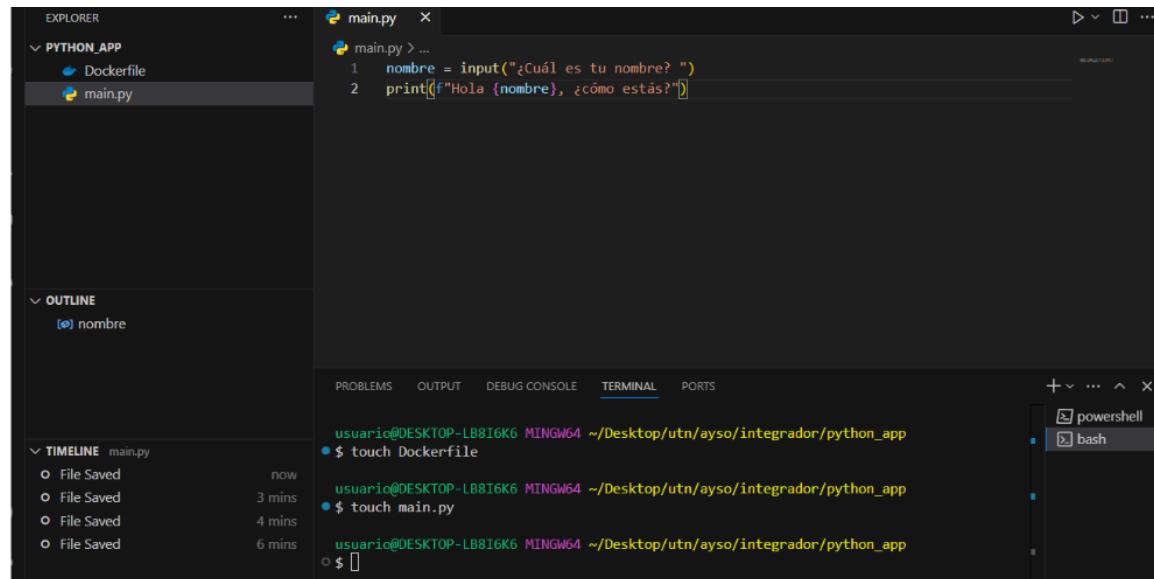


Figura 17: Creación del Dockerfile y main.py (2)



The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a folder named "PYTHON_APP" containing "Dockerfile" and "main.py".
- Dockerfile** tab: Contains the following Dockerfile code:

```

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

```
- OUTLINE**: Shows "No symbols found in document 'Dockerfile'".
- TIMELINE**: Shows activity log:
 - File Saved now
 - File Saved 8 mins
 - Undo / Redo 16 mins
- TERMINAL** tab: Shows terminal history:

```

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

```
- OUTPUT**, **DEBUG CONSOLE**, **PROBLEMS**, and **PORTS** tabs are also visible.

Figura 18: Creación del Dockerfile y main.py (3)

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a folder named "PYTHON_APP" containing "Dockerfile" and "main.py".
- Dockerfile** tab: Contains the same Dockerfile code as in Figure 18.
- TERMINAL** tab: Shows terminal output for building the Docker image:

```

PS C:\Users\usuario\Desktop\utn\ayso\integrador\python_app> docker build -t my-python-app .

```
- OUTPUT**, **DEBUG CONSOLE**, **PROBLEMS**, and **PORTS** tabs are also visible.

Figura 19: Creación la imagen



The screenshot shows a code editor interface with a dark theme. In the top left, there are tabs for 'main.py' and 'Dockerfile'. The main area displays the following Python code:

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

Below the code editor is a terminal window titled 'python_app'. It shows the command 'docker run -it --rm --name my-app -v ./user/src/app my-python-app'. The terminal output is:

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

At the bottom of the terminal window, there are tabs for 'powershell' and 'bash'.

Figura 20: Ejecución del contenedor

This screenshot is similar to Figure 20, showing the same code editor and terminal setup. However, the terminal output has been modified to include an additional message at the end:

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

Figura 21: Ejecución del contenedor con cambios en main.py