

Trabajo Integrador AySO

Virtualización

[LINK AL VIDEO DE YOUTUBE \(CLICK AQUÍ\)](#)

Alumnos

Enderson Eduardo Suarez Porras

Damián Eduardo Tristant

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Arquitectura y Sistemas Operativos

Docente Titular

Osvaldo Falabella

Docente Tutor

Adriel Ezequías Herrera

6 de Junio de 2025

Índice

1. Introducción
2. Marco Teórico
3. Caso práctico.
4. Metodología Utilizada
5. Resultados Obtenidos
6. Conclusiones
7. Bibliografía

1. Introducción

En el presente trabajo se aborda la tecnología de la virtualización motivados por su capacidad para la optimización de recursos y la agilización de procesos, y por los claros beneficios que esto significa para quienes se encuentran estudiando programación, aquellos que necesitan no solo lograr, hasta cierto punto, la independización del hardware y sortear sus limitaciones, sino generar entornos de experimentación seguros, que permitan, de ahora en adelante, desarrollar, explorar y testear nuevas prácticas y software, manteniendo la integridad de sus equipos físicos.

Es por esto que el manejo de tecnologías de virtualización se transforma en una competencia para analizar, y el acercamiento práctico y conceptual que se despliega en las siguientes páginas servirá para dicho fin. Haciendo un recorrido por los pasos para crear y configurar una máquina virtual, en cuyo entorno se instalará la plataforma de software Docker, y a partir de allí la creación de un contenedor en el cual se levantará un servidor web.

Esta elección de herramientas permitirá articular la virtualización en diferentes niveles y aislamiento, con el objetivo de profundizar en la comprensión de cómo interactúan y se pueden potenciar entre sí. Aunque Docker permite ejecutar aplicaciones en contenedores aislados, sin necesidad de virtualizar un sistema operativo completo, la decisión de usar esta herramienta para lograr un segundo nivel de virtualización dentro de una máquina virtual, es una apuesta lúdica para explorar los conceptos que guían este trabajo.

2. Marco Teórico

La virtualización es una tecnología que comprende un conjunto de prácticas que nos permite crear una versión virtual de un recurso físico, como un sistema operativo, un servidor o una red. Lo que permite, en lugar de utilizar directamente el hardware, el uso de un componente llamado hipervisor, el cual se encarga de simular ese hardware y permitir que uno o varios sistemas operativos se ejecuten de forma independiente dentro de máquinas virtuales. La virtualización, entonces, permite abstraer y gestionar recursos físicos con mayor flexibilidad, generando entornos independientes que permiten desarrollar, explorar y experimentar, sin comprometer el funcionamiento del sistema anfitrión (host)

Partiendo de esto, podemos pensar la virtualización como un proceso de intermediación que permite que un sistema operativo o una aplicación funcionen de manera aislada en un mismo equipo físico. Esta capa de intermediación, puede implementarse en diferentes niveles y orientarse a fines específicos. Podemos distinguir para fines de este trabajo, principalmente dos niveles: la virtualización con máquinas virtuales o la virtualización con contenedores.

La virtualización con máquinas virtuales, es una virtualización a nivel hardware, que permite ejecutar todo un sistema operativo en un entorno físico independiente, que recibe el nombre de máquina virtual. Mediante el uso de un hipervisor, como software que permite crear y gestionar máquinas virtuales, se abstraen los recursos físicos del equipo que han sido asignados a ese entorno, en el cual se va a alojar y ejecutar el sistema operativo invitado (Guest OS). Las máquinas virtuales existen dentro de un sistema operativo anfitrión (Host), que es el que se encuentra instalado directamente en el hardware físico del equipo. En el medio, el hipervisor tipo 2, que corre dentro del sistema operativo host como cualquier otro programa, por ejemplo, VirtualBox o VMware. También existen hipervisores tipo 1, que no se encuentran instalados sobre

un sistema operativo, sino que corren directamente sobre el hardware, por ejemplo, VMware ESXI

Por su parte la virtualización mediante contenedores, es una virtualización a nivel de sistema operativo, que utiliza tecnologías de contenedores como Docker. A diferencia de las máquinas virtuales que ya definimos, los contenedores no replican un sistema operativo completo, sino que comparten el núcleo del sistema operativo del host - es decir, utilizan el mismo kernel- y mantienen entornos aislados para cada aplicación o servicio.

Docker, es una de las plataformas para contenedores de uso más extendido. Sirve para facilitar la creación, distribución y ejecución de aplicaciones dentro de contenedores. En ellos, encapsula todo lo necesario para que una aplicación funcione -su código, librerías, dependencias, variables de entorno- todo en una única unidad ejecutable. Dentro de Docker, existen las imágenes, que son plantillas listas para ejecutarse, y se pueden usar para crear múltiples contenedores iguales, lo que favorece la portabilidad y la replicabilidad de los entornos.

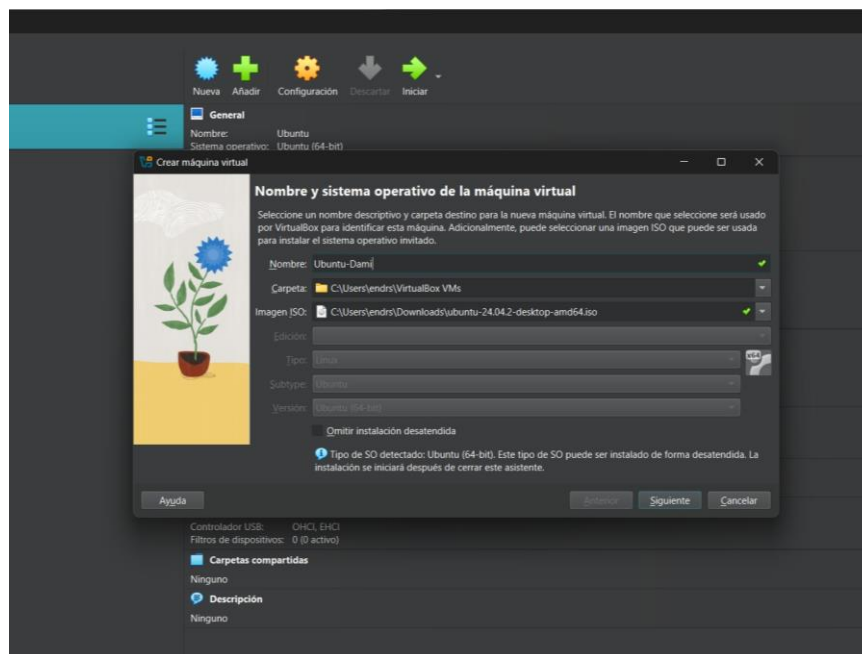
Para concluir este apartado, queremos destacar una de las características más valoradas de la virtualización, tanto en el uso de máquinas virtuales como de contenedores, se crean entornos aislados. Este aislamiento implica que el comportamiento de una aplicación, las consecuencias o acciones que se derivan de su ejecución, así como sus posibles errores y configuraciones específicas no afectan al sistema operativo anfitrión ni otras aplicaciones que estén corriendo a la par.

Entre todos los otros beneficios, como estudiantes de la Tecnicatura de Programación, encontramos esta característica fundamentalmente atractiva, porque permite generar entornos de prueba para ejecutar nuevas configuraciones, sin comprometer nuestro equipo real. Otra de las principales ventajas de la virtualización es que permite separar el software del hardware físico. Tanto las máquinas virtuales como los contenedores pueden ejecutarse en distintos equipos o servidores sin modificar su funcionamiento

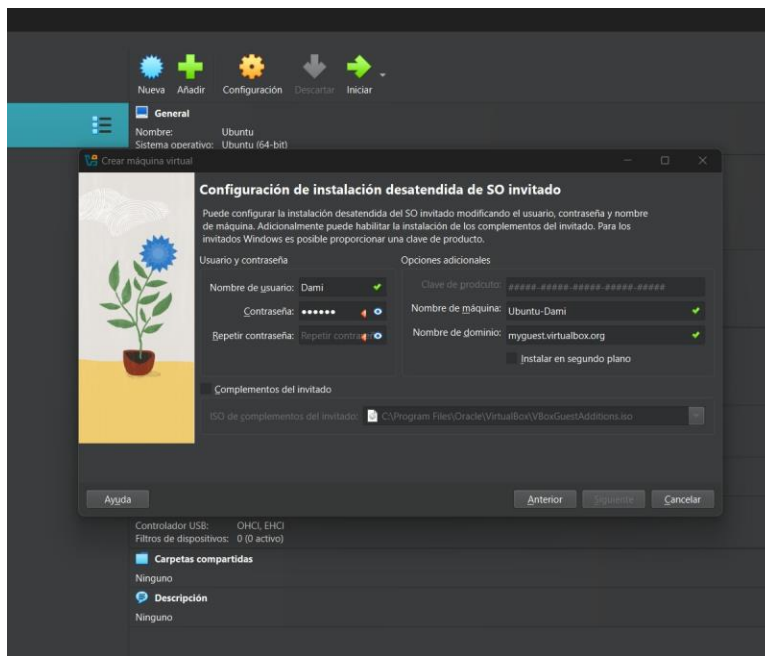
facilitando la portabilidad de los entornos, para replicarlos, moverlos y compartirlos sin depender de un dispositivo específico. Permiten ponerle fin a la penosa excusa de “en mi computadora sí anda” garantizando que el contenido funcione en cualquier entorno.

3. Caso práctico

1. Para nuestro caso práctico hemos desplegado la instalación de una máquina virtual. Para lo que hemos descargado VirtualBox, y realizado su instalación aceptando todos los pasos predeterminados. Al finalizar la instalación hemos abierto VirtualBox.
2. Para descargar Ubuntu, lo hicimos a través de la página web <https://ubuntu.com/download/dekstop> y descargamos el archivo ubuntu-24.04.2-desktop-amd64.iso. Este archivo es una imagen exacta del sistema operativo
3. Desde VirtualBox creamos y configuramos una nueva máquina virtual.
 - Nombre: Ubuntu-Dami
 - Imagen ISO: ubuntu-24.04.2-desktop-amd64.iso. (Le asignamos el archivo .ISO correspondiente al sistema operativo)



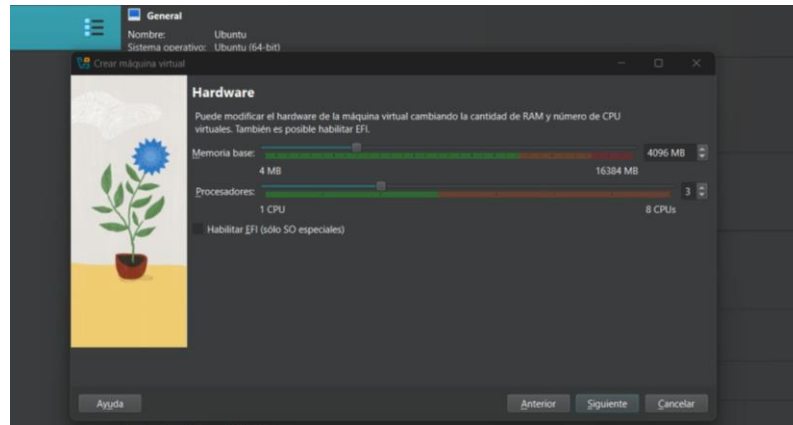
- Le asignamos una contraseña a nuestro nuevo entorno virtual, para aumentar la seguridad, y evitar accesos no deseados.



- Asignación de recursos a nuestra máquina virtual; En la configuración de nuestro entorno virtual le asignamos los recursos físicos que estaría destinados a su funcionamiento.

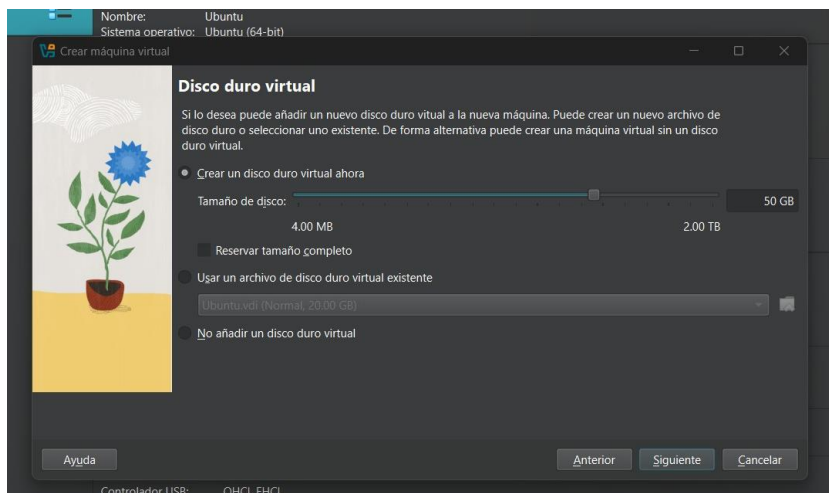
- **Memoria RAM:** 4096 MB, ya que nuestro equipo lo podría permitir.
- **Procesadores:** La decisión de asignar 3 CPUs a nuestra MV sobre el sistema anfitrión con 8 CPUs físicos se basa en equilibrar el rendimiento de nuestro entorno virtual con el rendimiento de nuestro sistema anfitrión.

• Por la de los hemos

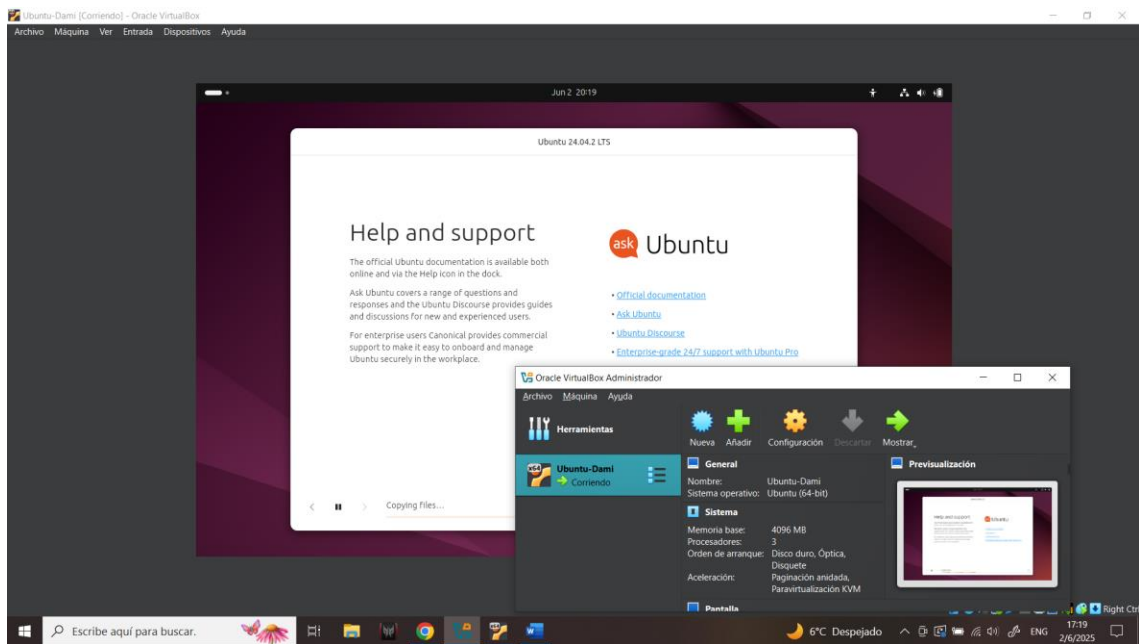
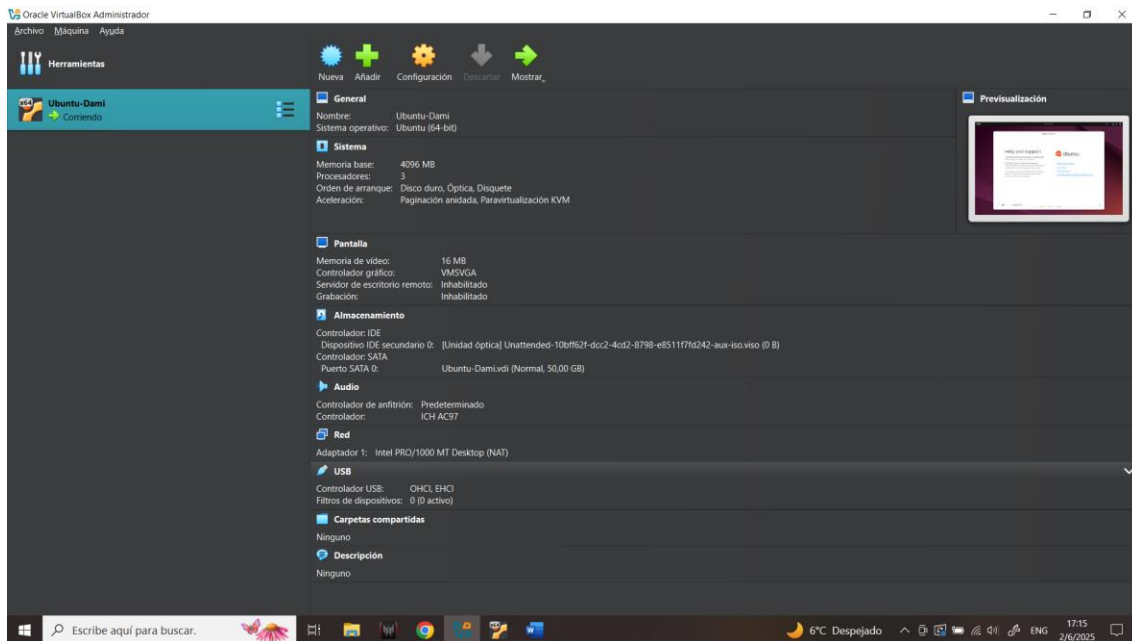


último, para culminar con asignación recursos le asignado un

espacio de 50GB de almacenamiento en el disco.

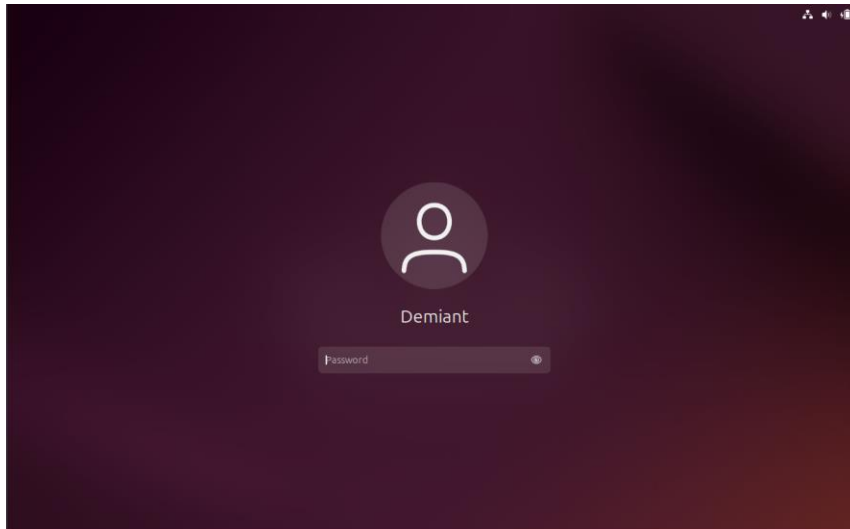


Quedando de esta manera configurada en VirtualBox nuestra máquina virtual, procedimos a la creación de la misma y a la consecuente instalación del sistema operativo Ubuntu en nuestro nuevo entorno ajustado a las características que le hemos asignado.



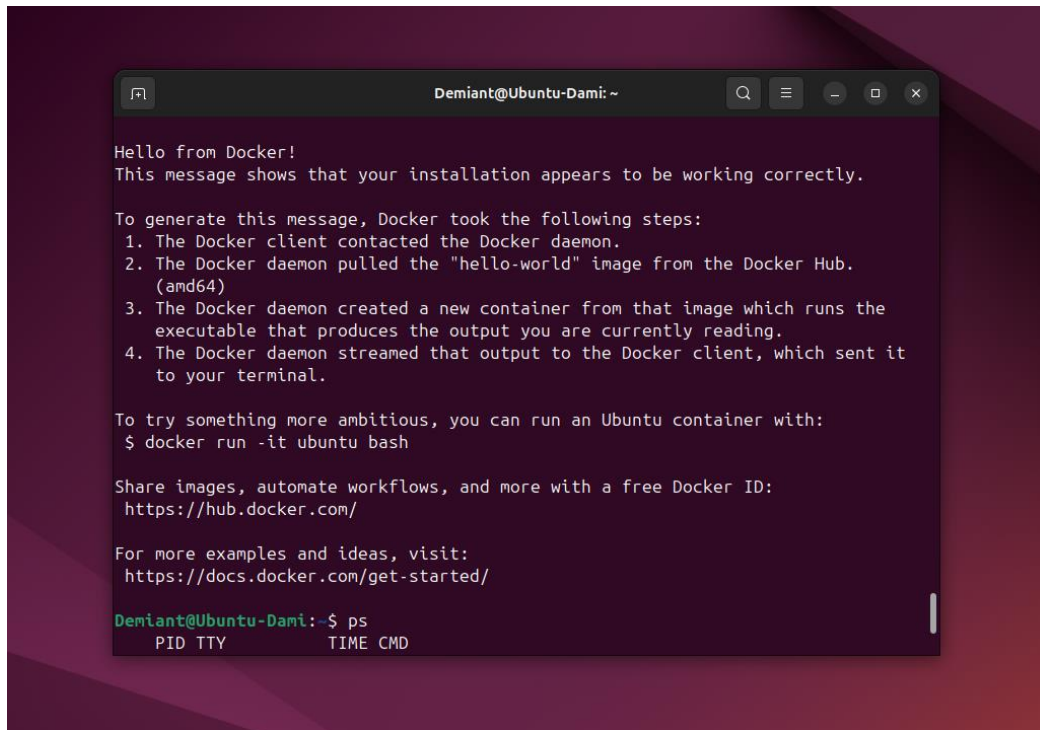
En la última captura de pantalla, podemos observar como se encuentran copiando los archivos necesarios para completar la instalación de Ubuntu.

Y una vez culminando este proceso, se procedió a ingresar al entorno virtual que aloja el sistema operativo Ubuntu para continuar con nuestra práctica de virtualización, con la instalación de Docker y la creación de un contenedor con un servidor web Nginx.



4. Instalación de Docker y prueba de funcionamiento

- Instalamos Docker usando el comando
 - **sudo apt update**
 - **sudo apt install [docker.io](https://docs.docker.com/engine/install/debian/)**
 - **sudo docker run hello-world**

A screenshot of a terminal window titled 'Demiant@Ubuntu-Dami: ~'. The terminal displays the 'hello-world' message from Docker, explaining the steps taken to run the container and providing links to Docker Hub and documentation. At the bottom, the 'ps' command is entered, showing the current processes.

```
Demiant@Ubuntu-Dami: ~  
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/  
  
Demiant@Ubuntu-Dami:~$ ps  
PID TTY          TIME CMD
```

5. Acto seguido intentamos descargar con el comando **docker pull nginx** la imagen del servidor web nginx desde Docker Hub, usando nuestra consola, pero recibimos un error de **permission denied**. Esto se debe a que Docker no cuenta con los permisos suficientes para ejecutarse de esa manera, por lo que tuvimos que usar el comando **sudo docker pull nginx**, y de esta manera pudimos descargar la imagen **nginx:latest**. Tuvimos que usar el comando de superusuario sudo, lo que demuestra que nuestro Docker funciona bien, pero necesita ejecutarse de esta manera porque no cuenta con los permisos necesarios.

```

Demiant@Ubuntu-Dami:~$ docker pull nginx
Using default tag: latest
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.50/containers/json?all=1": dial unix /var/run/docker.sock: connect: permission denied
Demiant@Ubuntu-Dami:~$ sudo docker pull nginx
[sudo] password for Demiant:
Using default tag: latest
latest: Pulling from library/nginx
61320b01ae5e: Pull complete
670a101d432b: Pull complete
405bd2df85b6: Pull complete
cc80efff8457: Pull complete
2b9310b2ee4b: Pull complete
6c4aa022e8e1: Pull complete
abddc69cb49d: Pull complete
Digest: sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
Demiant@Ubuntu-Dami:~$

```

6. Ya con nuestra imagen de Nginx correctamente descargada procedemos a ejecutar el comando:

- **sudo docker run -d -p 8080:80 --name webprueba nginx**

```

Demiant@Ubuntu-Dami:~$ sudo docker run -d -p 8080:80 --name webprueba nginx
91184eb24085601a1f17583232c641ba9ca30b88ef5ac437434a5996fd47862a
Demiant@Ubuntu-Dami:~$

```

Hagamos una revisión de las partes que componen nuestra línea de comando para adentrarnos en la comprensión del funcionamiento de Docker:

El comando **docker run** es el que nos sirve para crear y ejecutar un contenedor a partir de una imagen. En este caso basado en la imagen oficial de nginx,

-d Hace que el contenedor se ejecute en segundo plano,

-p 8080:80 Establece los puertos. Conecta el puerto 80 del contenedor al puerto 8080 de la computadora. En el contenedor, Nginx siempre escucha el puerto

80 porque es su configuración, mientras que 8080 en el host se elige para evitar conflictos con otros servicios y porque al ser un puerto libre y no privilegiado.

Esto se conoce como mapeo de puerto, y permite acceder al servidor Nginx desde un navegador escribiendo <http://localhost:8080>, ya que cualquier solicitud a ese puerto será redirigida por docker hacia el puerto 80 dentro del contenedor, donde está corriendo el servidor.

Continuando con el análisis de nuestra línea de comandos:

--name webprueba: asigna un nombre personalizado al contenedor para identificarlo.

nginx: especifica la imagen a utilizar para crear el contenedor, que en nuestro caso por defecto será nginx:latest, descargada anteriormente.

```
run docker run --help for more information
Demiant@Ubuntu-Dami:~$ sudo docker run -d -p 8080:80 --name webprueba nginx
91184eb24085601a1f17583232c641ba9ca30b88ef5ac437434a5996fd47862a
Demiant@Ubuntu-Dami:~$
```

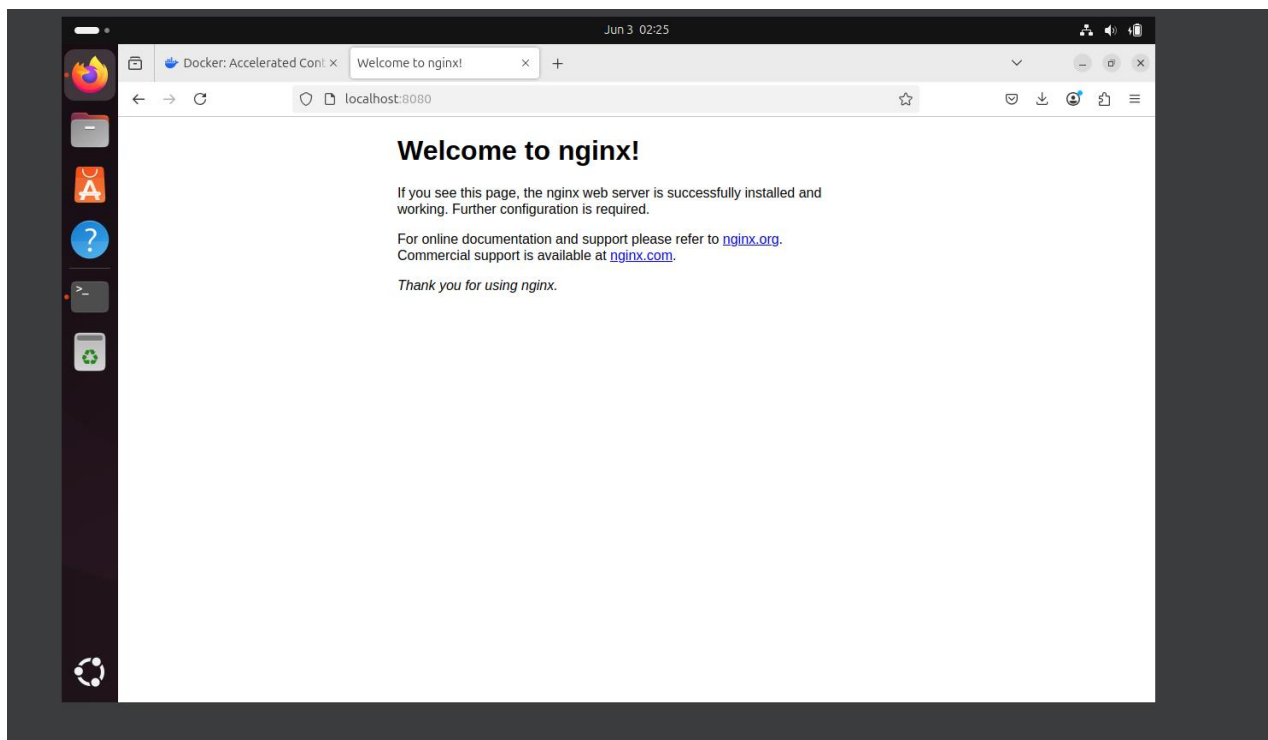
Una vez ejecutado nuestro comando, Docker regresa una línea con el identificador de nuestro contenedor, lo que indica que se ha creado correctamente.

7. Comprobamos la creación de nuestro contenedor con el comando **docker ps**, que nos devuelve una lista de los contenedores que se están corriendo actualmente.

```
Demiant@Ubuntu-Dami:~$ sudo docker run -d -p 8080:80 --name webprueba nginx
91184eb24085601a1f17583232c641ba9ca30b88ef5ac437434a5996fd47862a
Demiant@Ubuntu-Dami:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
PORTS         NAMES
91184eb24085   nginx    "/docker-entrypoint.    4 minutes ago Up 4 minutes
0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   webprueba
```

8. Para finalizar, accedemos al servidor web nginx desde el navegador del host (en este caso ubuntu es el host de nuestro contenedor)

Para esto escribimos: <http://localhost:8080> en el navegador y podemos acceder.



5. Resultados Obtenidos

Como resultado del proceso pudimos crear y configurar nuestra máquina virtual Ubuntu (guest) en nuestro entorno windows (host), utilizando un hipervisor tipo 2 como lo es VirtualBox. Esta máquina virtual sirvió como base para la instalación de Docker. A partir de allí, logramos ejecutar correctamente un contenedor con la imagen oficial de Nginx, al cual pudimos acceder a través del navegador del sistema operativo anfitrión del contenedor, es decir, Ubuntu.

Con la evaluación de la línea de código de creación del contenedor, pudimos conocer el concepto de mapeo de puertos mediante el parámetro `-p 8080:80`, y entender su función al redirigir las solicitudes del puerto 8080 del host virtual hacia el puerto 80 interno del contenedor, donde se ejecuta el servidor web.

Este proceso nos ayudó a comprobar el funcionamiento práctico de Docker dentro de una máquina virtual, y colaboró en la apropiación de conceptos claves del mundo de la virtualización como lo son las imágenes, contenedores, puertos, y a distinguir claramente entre el host físico (windows), el host virtual y los contenedores.

Hemos podido hacer un recorrido por los distintos niveles de la virtualización, y obtener en el proceso herramientas prácticas para trabajar de ahora en adelante.

6. Conclusiones

A lo largo del trabajo se logró comprender de manera práctica la integración de distintos niveles de virtualización y como herramientas como Docker y VirtualBox pueden combinarse para generar entornos de prueba seguros y fácilmente replicables. Esta experiencia fortaleció nuestros conocimientos técnicos y nos permitió iniciarnos en el dominio de tecnologías y prácticas de virtualización, lo cual dejó planteada una interrogante: ¿cuánto depende la innovación de la capacidad de simular, experimentar y fallar sin comprometer los sistemas físicos? Una pregunta que, aunque no corresponde responder aquí, sin duda representa un terreno fértil para futuros trabajos.

7. Bibliografía

- Documentación oficial de Docker: <https://docs.docker.com/>
- Nginx Docker Hub: https://hub.docker.com/_/nginx