

UNIDAD TRABAJO 5:

VIRTUALIZACIÓN.

CONTENEDORES. DOCKER.

CONSTRUCCIÓN DE IMAGENES: DOCKER BUILD Y DOCKERFILE.

Esta unidad didáctica se ha desarrollado para el Curso de Especialización de Ciberseguridad en entornos de las Tecnologías de la Información. En concreto para el módulo de Puesta en Producción Segura, cuyos aspectos básicos del currículo vienen recogidos en el Real Decreto 479/2020.

Licencia: *Creative Commons Atribución - No Comercial -Compartir Igual (CC BY-NC-SA 4.0)*

José Gaspar Sánchez García.

Pedro Antonio Santiago Santiago.

Normas de entrega:

Corrección:

-  Por cada día de retraso sin justificación en la entrega, se restará 1 punto a la nota.

- 🕊 Es muy importante cuidar la presentación y limpieza de las respuestas entregadas al profesor.
- ✍ En caso de que se detecte copia o plagio con los ejercicios de otros compañeros, se invalidará la nota del ejercicio.

Forma de Entrega:

- A través de la plataforma Moodle GVA AULAS: <https://aules.edu.gva.es/> **Documentos a entregar:**
- Este mismo fichero con las preguntas rellenas (usar fuente Verdana de tamaño 12 color azul en cursiva – **Estilo Respuesta**): *[Respuesta]*.
- Añade la dirección URL del repositorio GitHub creado al documento que contiene las respuestas.
- Documento con la respuesta a las cuestiones y explicación al desarrollo de la práctica, en formato del documento **PDF**.
- En el documento **PDF** con la memoria de la práctica incluye también un enlace al **repositorio GitHub** que contiene los recursos y ficheros necesarios para resolver la práctica.
- Cambia el nombre al documento, y añade tu PRIMER APELLIDO y tu NOMBRE.

Índice:

1. Construcción de imágenes: <i>docker build</i> y <i>Dockerfile</i>	3
1.1. Introducción:.....	3
1. Docker Build	3
2. Dockerfile	4
2. Optimización del <i>Dockerfile</i>	8
2.1. Uso de <i>.dockerignore</i>	8
2.2. Reducción del tamaño de los contenedores	8
2.3. Minimización del número de <i>layers</i>	9
2.4. Optimización del uso de la caché	11
2.5. Parametrización empleando argumentos (ARG)	11
2.6. Multi-stage build	14
3. Practicamos lo aprendido	17
3.1. KaliLinux en Docker	19
3. Contenedor Docker KaliLinux	19
4. Dockerfile para KaliLinux	23

5. Instalación de un cliente RDP en el host anfitrión.....	26
3.2. Optimización de contenedores	27
4. Referencias.....	30
4.1. Vídeos	30

1. Construcción de imágenes: *docker build* y *Dockerfile*

1.1. Introducción:

Los contenidos básicos que vamos a estudiar a través de esta lección son:

1. Docker build.
2. Dockerfile.
3. Optimización de Dockerfile.

Docker Build y **Dockerfile** son dos componentes clave en el proceso de construcción de imágenes de contenedores personalizadas en Docker. A continuación, se explica el funcionamiento de cada uno:

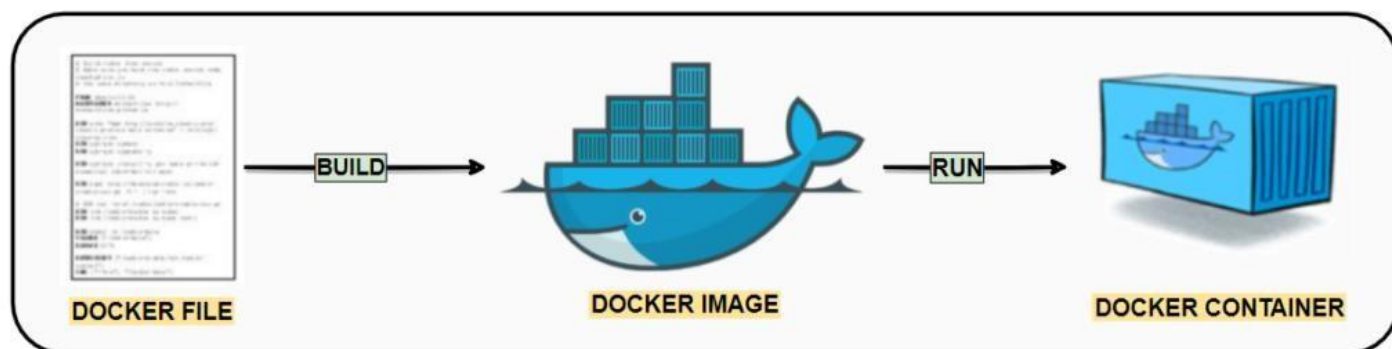


Figura 1. Construcción y ejecución imágenes con Dockerfile.

1. Docker Build.

Docker Build es el comando utilizado para construir una imagen de contenedor a partir de un **Dockerfile** y otros recursos necesarios. Este comando toma como entrada un contexto de construcción, que es un directorio local que contiene el **Dockerfile** y cualquier otro archivo requerido por la aplicación. **Docker Build** ejecuta las instrucciones definidas en el **Dockerfile** y crea una nueva imagen de contenedor basada en esas instrucciones.

El comando **docker build** realiza los siguientes pasos:

- Lee el *Dockerfile* y las instrucciones que contiene.

- Crea una imagen base inicializada a partir de una imagen existente o una imagen base predeterminada.
- Ejecuta cada instrucción del *Dockerfile* en orden, construyendo capas adicionales en la imagen base.
- Cada instrucción puede agregar, eliminar o modificar archivos o configuraciones en la imagen.
- Cada instrucción genera una nueva capa en la imagen, lo que permite la reutilización y la eficiencia en las construcciones posteriores.
- Al finalizar la construcción, se genera una imagen de contenedor lista para ser utilizada.

2. Dockerfile

El **Dockerfile** es un archivo de texto plano que contiene una serie de instrucciones y comandos utilizados por **Docker Build** para construir una imagen de contenedor. El **Dockerfile** define los pasos necesarios para configurar el entorno de ejecución de la aplicación dentro del contenedor.

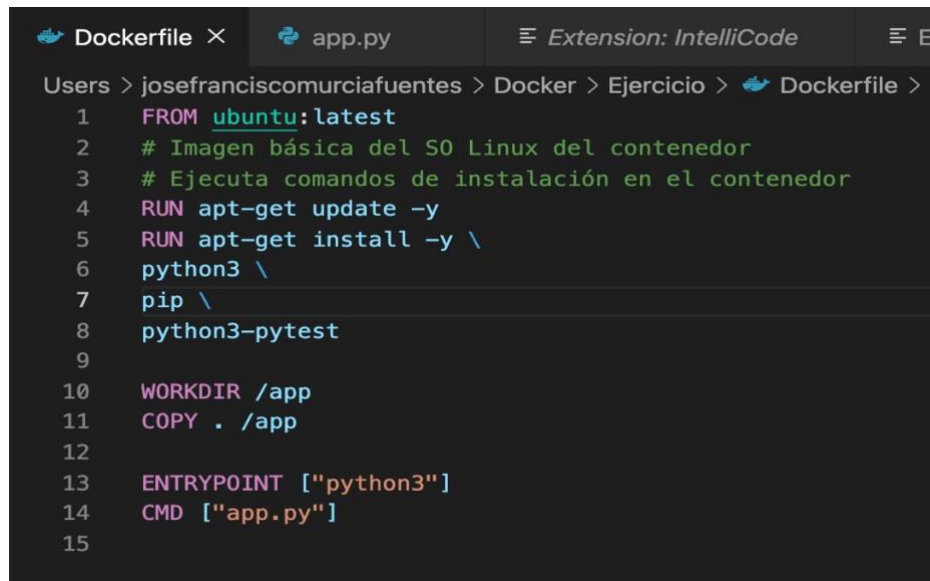
El *Dockerfile* incluye instrucciones como:

- **ARG**: permite parametrizar la construcción del contenedor mediante argumentos.
- **ENV**: permite especificar variables de entorno.
- **FROM**: especifica la imagen base a partir de la cual se construirá la imagen del contenedor.
- **RUN**: ejecuta comandos en el entorno del contenedor durante el proceso de construcción.
- **COPY**: copia archivos y directorios desde el contexto de construcción al contenedor.
- **ADD**: se utiliza para copiar archivos o directorios desde la máquina local (donde está instalado Docker) al contenedor Docker. A diferencia de la instrucción **COPY**, **ADD** tiene algunas características adicionales, como la posibilidad de copiar archivos comprimidos (gzip, bzip2, xz) y la capacidad de copiar archivos desde una URL. Es importante tener en cuenta que el uso de la instrucción **ADD** implica la creación de una nueva capa de imagen, por lo que debes ser cuidadoso al implementar esta opción.
- **WORKDIR**: establece el directorio de trabajo dentro del contenedor.
- **VOLUME**: permite definir un volumen para el almacenamiento de datos de forma persistente.
- **EXPOSE**: especifica los puertos en los que la aplicación dentro del contenedor escucha conexiones.
- **ENTRYPOINT**: define el comando que se ejecutará cuando se inicie un contenedor. Si se especifica un **ENTRYPOINT**, este comando se ejecutará siempre que se inicie el contenedor, y cualquier comando que se pase al contenedor se ejecutará como argumentos del **ENTRYPOINT**.

- **CMD**: define el comando predeterminado que se ejecutará cuando se inicie el contenedor.

Es decir, si se especifica un **ENTRYPOINT**, este comando se ejecutará siempre que se inicie el contenedor, y cualquier comando que se pase al contenedor se ejecutará como argumentos del **ENTRYPOINT**. Si se especifica un **CMD**, este comando se ejecutará solo si no se especifica un comando al iniciar el contenedor.

1. Implementa el siguiente fichero de ejemplo *Dockerfile*: [\[Capturas de pantallas de fichero Dockerfile : \]](#).



```
1 FROM ubuntu:latest
2 # Imagen básica del SO Linux del contenedor
3 # Ejecuta comandos de instalación en el contenedor
4 RUN apt-get update -y
5 RUN apt-get install -y \
6 python3 \
7 pip \
8 python3-pytest
9
10 WORKDIR /app
11 COPY . /app
12
13 ENTRYPOINT ["python3"]
14 CMD ["app.py"]
15
```

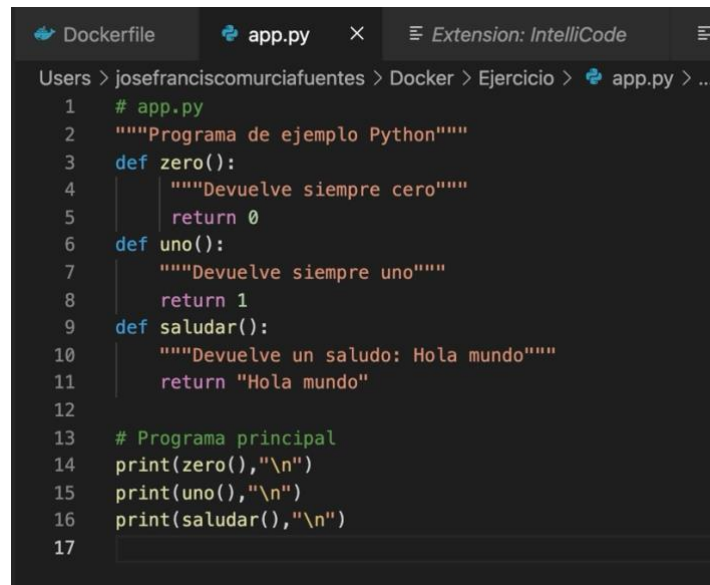
].

```
FROM ubuntu:latest
# Imagen básica del SO Linux del contenedor
# Ejecuta comandos de instalación en el contenedor
RUN apt-get update -y
RUN apt-get install -y \
python3 \ pip \ python3-
pytest

WORKDIR /app
COPY . /app

ENTRYPOINT ["python3"]
CMD ["app.py"]
```

Ejemplo fichero código fuente: [app.py](#) [Capturas de pantallas de la app donde se ejecutan las 3 funciones.



```
1  # app.py
2  """Programa de ejemplo Python"""
3  def zero():
4      """Devuelve siempre cero"""
5      return 0
6  def uno():
7      """Devuelve siempre uno"""
8      return 1
9  def saludar():
10     """Devuelve un saludo: Hola mundo"""
11     return "Hola mundo"
12
13  # Programa principal
14  print(zero(),"\n")
15  print(uno(),"\n")
16  print(saludar(),"\n")
17
```

].

```
# app.py
"""Programa de ejemplo Python"""
def zero():
    """Devuelve siempre cero"""
    return 0
def uno():
    """Devuelve siempre uno"""
    return 1
def saludar():
    """Devuelve un saludo: Hola mundo"""
    return "Hola mundo"

# Programa principal
print(saludar(),"\n")
```

2. A continuación, construimos la *imagen my-ubuntu-python*. *[Capturas de ejecución del comando y el resultado:]*

```
josefranciscomurciafuentes@MacBook-Pro-de-Jose-Ejercicio % docker build -t my-ubuntu-python .
[+] Building 30.3s (9/10)
=> [internal] load build definition from Dockerfile
=> Building 30.4s (9/10)
=> [internal] load build definition from Dockerfile
=> Building 30.6s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 30.7s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 30.9s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.0s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.2s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.3s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.5s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.6s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.8s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 31.9s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.1s (9/10)
=> [internal] load build definition from Dockerfile
[+] Building 32.1s (9/10)
=> [internal] load .dockerignore
[+] Building 34.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 312B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> transferring context: 2B
=> CACHED [1/5] FROM docker.io/library/ubuntu:latest@sha256:80dd3c3b9c6cceb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
=> resolve docker.io/library/ubuntu:latest@sha256:80dd3c3b9c6cceb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
[+] Building 34.7s (10/10)
=> [internal] load build context
=> transferring context: 338B
=> [2/5] RUN apt-get update -y
=> [3/5] RUN apt-get install -y python3 pip python3-pytest
=> [4/5] WORKDIR /app
=> [5/5] COPY . /app
=> exporting to image
=> exporting layers
=> exporting manifest sha256:6f5320de41cf55a57dd22726825e4a95a997307400b1a054dd41d6eac00b1f89
=> exporting config sha256:7621a011c3cf0a17e75997e7cb6fc5c3790e23df85f3701750027ec551007cce
=> exporting attestation manifest sha256:5ee62edbfff12dc180c558e3c8d229ea6511a2722a751246459cd139d4c8d51e
=> exporting manifest list sha256:a4cc12fe2336ffdaf2919350c54eb55fabc0f108e4ddeddbbb39addc3c1c9be
=> naming to docker.io/library/my-ubuntu-python:latest
=> unpacking to docker.io/library/my-ubuntu-python:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/6pxe1ojqy5x41x46z469qq06i
josefranciscomurciafuentes@MacBook-Pro-de-Jose-Ejercicio %
```

1.

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

3. A continuación, comprobamos los tamaños de las diferentes imágenes. *[Capturas donde se muestra el tamaño de la imagen]*

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/6pxe1ojqy5x41x46z469qq06i
josefranciscomurciafuentes@MacBook-Pro-de-Jose Ejercicio % docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-ubuntu-python	latest	a4cc12fe2336	4 minutes ago	805MB

7.

```
docker images
```

4. Por último, ejecutamos un contenedor a partir de la imagen construida. *[Capturas del resultado de la ejecución :*

```
josefranciscomurciafuentes@MacBook-Pro-de-Jose Ejercicio % docker run -it my-ubuntu-python
0

1

Hola mundo

josefranciscomurciafuentes@MacBook-Pro-de-Jose Ejercicio %
```

7.

```
docker run -it my-ubuntu-python
```


2. Optimización del *Dockerfile*

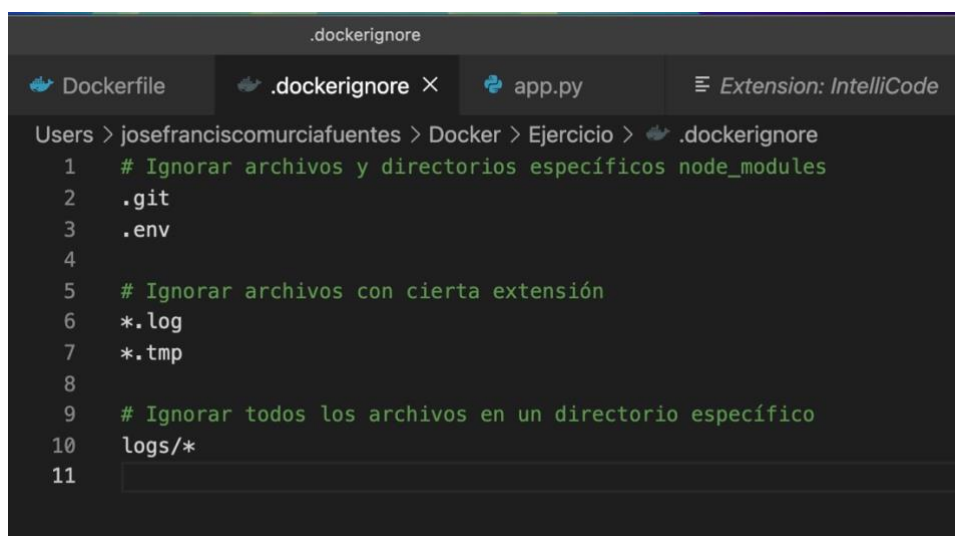
Estas son algunas de las buenas prácticas y técnicas de optimización que se pueden aplicar a **Dockerfile**:

1. Uso de **.dockerignore**.
2. Reducción del tamaño de los contenedores.
3. Minimización del número de layers.
4. Optimización del uso de la caché.
5. Parametrización empleando argumentos (ARG).
6. Multi-stage build.

2.1. Uso de .dockerignore

El archivo **.dockerignore** permite especificar patrones de archivos y directorios que Docker debe omitir al construir la imagen. Esto es útil para evitar incluir archivos innecesarios en la imagen, lo que puede reducir el tamaño final del contenedor y acelerar el proceso de construcción.

Ejemplo fichero .dockerignore [[Capturas de pantallas del documento creado](#)]



```
.dockerignore
1  # Ignorar archivos y directorios específicos node_modules
2  .git
3  .env
4
5  # Ignorar archivos con cierta extensión
6  *.log
7  *.tmp
8
9  # Ignorar todos los archivos en un directorio específico
10 logs/*
11
```

].

2.2. Reducción del tamaño de los contenedores

- **Utiliza imágenes base más pequeñas:** Elige imágenes base que sean lo más pequeñas posible y que contengan solo los componentes necesarios para tu aplicación.
- **Minimiza los paquetes y dependencias:** Instala solo los paquetes y dependencias necesarios para que tu aplicación funcione correctamente.
- **Limpia los archivos temporales:** Elimina los archivos temporales y los paquetes de instalación después de que se hayan utilizado en el *Dockerfile*.

- **Utiliza volúmenes para datos persistentes:** Evita almacenar datos persistentes dentro del contenedor y utiliza volúmenes de Docker para almacenarlos fuera del contenedor.

2.3. Minimización del número de *layers*

- **Combina instrucciones RUN:** Agrupa varias instrucciones RUN en una sola para reducir el número de capas generadas. Por ejemplo, en lugar de ejecutar varios comandos RUN para instalar paquetes, puedes combinarlos en uno solo.
- **Utiliza instrucciones COPY y ADD en una sola línea:** En lugar de copiar o agregar archivos uno por uno, puedes utilizar una sola instrucción COPY o ADD con comodines para copiar varios archivos en una sola capa.

Ejemplo combinación instrucciones RUN:

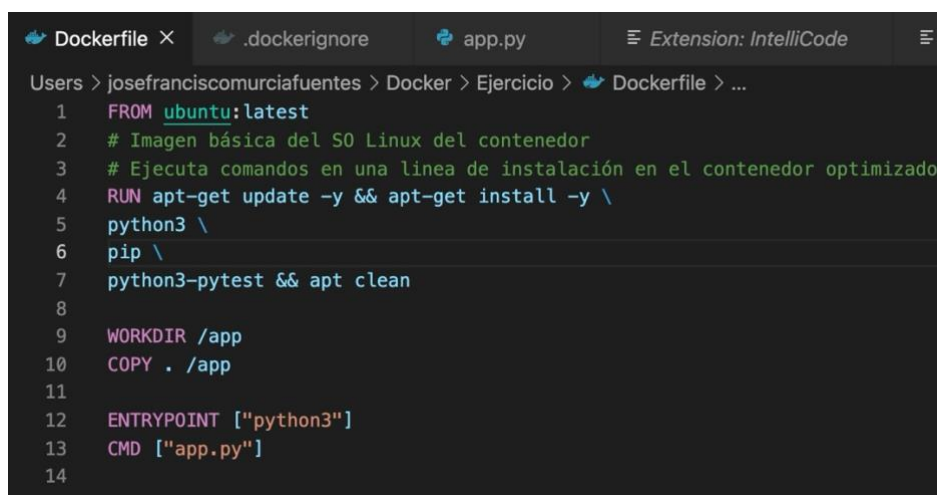
- **Varias instrucciones RUN**

```
RUN apt-get update -y
RUN apt-get install -y \
python3 \ pip \ python3-
pytest
```

- **Instrucciones RUN combinadas**

```
RUN apt-get update && apt-get install -y \ python3 \ pip \
python3-pytest && apt clean
```

5. Vuelva a describir el fichero **Dockerfile** incluyendo la optimización que permite reducir el número de capas gracias a la combinación de instrucciones **RUN**: [\[Capturas de pantallas del archivo optimizado\]](#)



```
Dockerfile X .dockerignore app.py Extension: IntelliCode
Users > josefranciscomurciafuentes > Docker > Ejercicio > Dockerfile > ...
1 FROM ubuntu:latest
2 # Imagen básica del SO Linux del contenedor
3 # Ejecuta comandos en una línea de instalación en el contenedor optimizado
4 RUN apt-get update -y && apt-get install -y \
5 python3 \
6 pip \
7 python3-pytest && apt clean
8
9 WORKDIR /app
10 COPY . /app
11
12 ENTRYPOINT ["python3"]
13 CMD ["app.py"]
14
```

].

6. A continuación, construimos la *imagen opt-ubuntu-python*. *[Capturas de pantallas de la nueva imagen :*

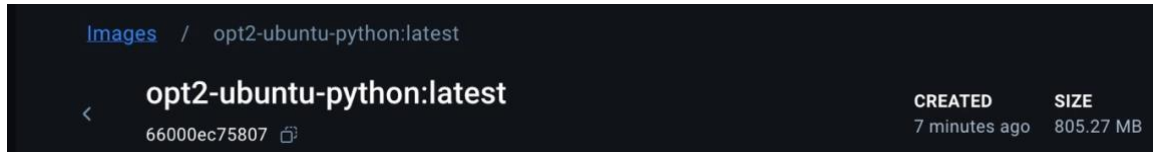
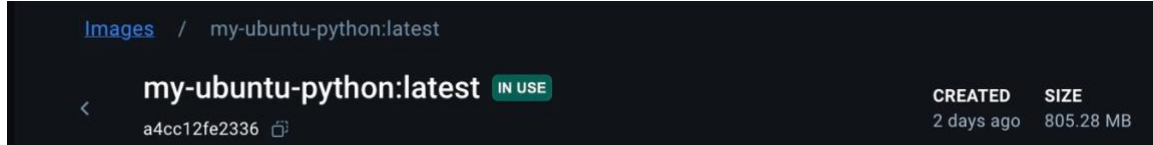
```
josefranciscoturciafuentes@MacBook-Pro-de-Jose Ejercicio % docker build -t opt-ubuntu-python .
[+] Building 37.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 346B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 287B
=> CACHED [1/4] FROM docker.io/library/ubuntu:latest@sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2
=> => resolve docker.io/library/ubuntu:latest@sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02c
=> [internal] load build context
=> => transferring context: 661B
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [2/4] RUN apt-get update -y && apt-get install -y python3 pip python3-pytest && apt clean
=> [3/4] WORKDIR /app
=> [4/4] COPY . /app
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:4f7fba46ccb3bb5fbbc62cf987d9d8f1214bc72733f8dfc415d30b67f87e7d7b
=> => exporting config sha256:d46efdb9596b2ad547b89ce69b65e062e58fa2864c03a6451116f1e61b32d8c3
=> => exporting attestation manifest sha256:8f36129e89048ff53db6653626557eed107fdc0c0d8ef201e92cd
=> => exporting manifest list sha256:9b0fe9d5a76d1681b1c3611fac1c6969a1b89036a51e5aba48d2ec926ea2
=> => naming to docker.io/library/opt-ubuntu-python:latest
=> => unpacking to docker.io/library/opt-ubuntu-python:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/lr2yvbv1exx5xr7xxzpk1oigz
josefranciscoturciafuentes@MacBook-Pro-de-Jose Ejercicio %
```

].

```
docker build -t opt-ubuntu-python .
```

7. A continuación, comprobamos los tamaños de las diferentes imágenes. *[Capturas de pantallas de los tamaños de las imágenes solo se ha reducido 1mb :*

```
josefranciscoturciafuentes@MacBook-Pro-de-Jose Ejercicio % docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
opt2-ubuntu-python  latest     66000ec75807  18 seconds ago  805MB
opt-ubuntu-python   latest     9b0fe9d5a76d  7 minutes ago  805MB
my-ubuntu-python    latest     a4cc12fe2336  47 hours ago  805MB
```

].

```
docker images
```

8. Por último, ejecutamos un contenedor a partir de la imagen construida. *[Capturas de pantalla de la ejecución de la imagen:*

```
josefranciscoturciafuentes@MacBook-Pro-de-Jose Ejercicio % docker run -it opt2-ubuntu-python
0
1
Hola mundo
josefranciscoturciafuentes@MacBook-Pro-de-Jose Ejercicio %
```

].

```
docker run -it opt-ubuntu-python
```

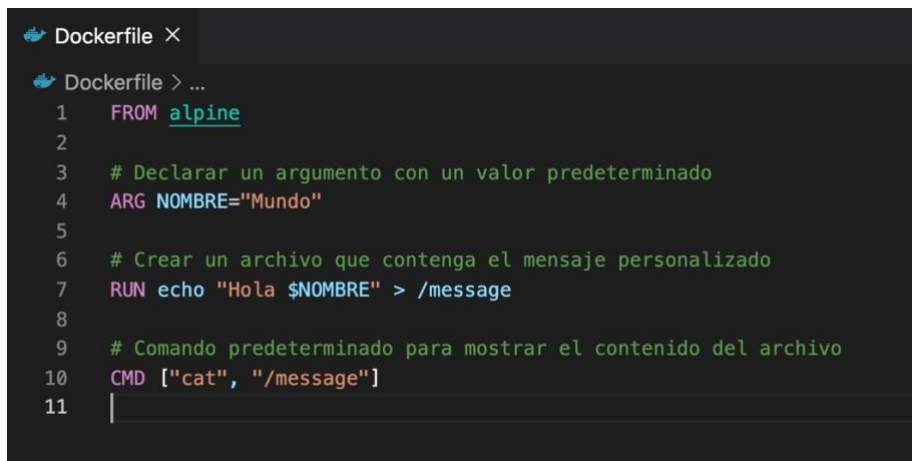
2.4. Optimización del uso de la caché

- **Ordena las instrucciones en el *Dockerfile*:** Coloca las instrucciones que cambian con mayor frecuencia al final del *Dockerfile*. Esto permite que Docker reutilice las capas en caché siempre que sea posible.
 - Despliegue del sistema operativo. ○ Instalación de aplicaciones y lenguajes de programación.
 - Incorporación de librerías y complementos de los lenguajes de programación.
- **Utiliza capas inmutables:** Evita modificar archivos o directorios existentes en las capas anteriores. Esto garantiza que Docker pueda reutilizar las capas en caché y no tenga que volver a construir las capas posteriores.

2.5. Parametrización empleando argumentos (ARG)

Los argumentos (**ARG**) en el *Dockerfile* te permiten pasar valores durante la construcción de la imagen. Puedes utilizar **ARG** para parametrizar valores como versiones de software, rutas de archivos o cualquier otro valor que pueda cambiar según el entorno. Esto hace que tu *Dockerfile* sea más flexible y reutilizable.

9. Por ejemplo, si queremos pasar un argumento a nuestro *Dockerfile*, podemos hacerlo de la siguiente manera: [\[Capturas de pantallas del Dockerfile : \]](#).



```
Dockerfile ×
Dockerfile > ...
1 FROM alpine
2
3 # Declarar un argumento con un valor predeterminado
4 ARG NOMBRE="Mundo"
5
6 # Crear un archivo que contenga el mensaje personalizado
7 RUN echo "Hola $NOMBRE" > /message
8
9 # Comando predeterminado para mostrar el contenido del archivo
10 CMD ["cat", "/message"]
11 |
```

].

```
FROM alpine
# Declarar un argumento con un valor predeterminado
ARG NOMBRE="Mundo"
# Crear un archivo que contenga el mensaje personalizado
RUN echo "Hola $NOMBRE" > /message
# Comando predeterminado para mostrar el contenido del archivo
CMD ["cat", "/message"]
```

10. En este caso, estamos definiendo un argumento **NOMBRE** con el valor por defecto **Mundo**. Si construimos la imagen: *[Captura de pantalla :*

```
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$ docker build -t hola .
[+] Building 1.5s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 314B
=> [internal] load metadata for docker.io/library/alpine:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/2] FROM docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686b9f45
=> => resolve docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686b9f45
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [2/2] RUN echo "Hola Mundo" > /message
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:e61580861f6b84aebc4f833b0a8e27c7b4f746f24ccf0aa1d76257a0583db00
=> => exporting config sha256:1ac27f8f537baffaf8286d4f2879b2847c43c9a17ce10b08ae18b517dca31cac1
=> => exporting attestation manifest sha256:01d3f0cd48375b8a2a2508a9034fb1bce8ed1d7b36210362f2d44553bfaf4e61
=> => exporting manifest list sha256:eec87d45d6e0e121827f42ee9f9d7a93f021a5470d962602e87bb49eac6baf3
=> => naming to docker.io/library/hola:latest
=> => unpacking to docker.io/library/hola:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ygpf485a055a4g6mk93487wke
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$
```

].

```
docker build -t hola .
```

11., Cuando iniciamos un contenedor se ejecutará el comando **echo Hola Mundo**.

[Captura de pantalla :

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ygpf485a055a4g6mk93487wke
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$ docker run -it hola
Hola Mundo
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$
```

].

```
docker run -it hola
```

12. Podemos sobrescribir el argumento **NOMBRE**, durante el proceso de construcción.

[Captura de pantalla de la entrada de argumento sobrescrito:

```
Start a build
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$ docker build --build-arg NOMBRE=Jose -t hola .
[+] Building 1.4s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 314B
=> [internal] load metadata for docker.io/library/alpine:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/2] FROM docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686b9f45
=> => resolve docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686b9f45
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [2/2] RUN echo "Hola Jose" > /message
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:92e4e34f912d3b172695eded53bb81bf0fd146ee20d440848742d02341760af7
=> => exporting config sha256:8988206af8e4575e3e73a2e007af51860667292fd60900e3f45c7b41ae953c63
=> => exporting attestation manifest sha256:a7c33c464f7839d8a2bacba8b6447bcab0d04d0e2235bd878d85dce1b401532e
=> => exporting manifest list sha256:4d8d86ca0ac7ca3621be30bcd91b27bd698f4985bdd1a6ef39e9feaba4bcfebc
=> => naming to docker.io/library/hola:latest
=> => unpacking to docker.io/library/hola:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/88p7d24v7xx1gjpikcpy0b7w
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$ docker run -it hola
Hola Jose
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscumurciafuentes$
```

].

```
docker build --build-arg NOMBRE=Gaspar -t hola .
```

13. Al iniciar un contenedor con el comando `docker run` se ejecutará el comando **echo Hola Gaspar**. Podríamos especificar el parámetro **--build-arg** tantas veces como argumentos tengamos en nuestro *Dockerfile*. [\[Captura de pantalla resultado ejecución de la imagen:\]](#)

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/88p7d24v7xx1gjpikcpy0ob7w
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscosmurciafuentes$ docker run -it hola
Hola Jose
MacBook-Pro-de-Jose:Ejercicio-2 josefranciscosmurciafuentes$
```

].

```
docker run -it hola
```

14. Realiza una captura de pantalla de la implementación del siguiente fichero *Dockerfile* de ejemplo: [\[Captura de pantalla del fichero Dockerfile:\]](#)

```
Dockerfile > ...
1 ARG VERSION=latest
2
3 FROM busybox:${VERSION:-latest}
4
5 RUN echo "Hola Mundo ${VERSION:-latest}" > /home/imagen_version
6 WORKDIR /home
7
8 VOLUME /home
9
10 CMD ["cat", "/home/imagen_version"]
11
```

].

```
ARG VERSION=latest

FROM busybox:${VERSION:-latest}

RUN echo "Hola Mundo ${VERSION:-latest}" > /home/imagen_version
WORKDIR /home

VOLUME /home

CMD ["cat", "/home/imagen_version"]
```

Podemos construir una imagen a partir del *dockerfile* del *ejemversion* con los siguientes comandos:

- Podemos indicar el valor de los argumentos (**ARG**) a través de la línea de comandos.

[Captura de pantalla:]

```
MacBook-Pro-de-Jose:Ejercicio-14 josefranciscumurciafuentes$ docker build --build-arg VERSION=1.33.1 -t ejemversion .
[+] Building 1.4s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 231B
=> [internal] load metadata for docker.io/library/busybox:1.33.1
=> [auth] library/busybox:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/3] FROM docker.io/library/busybox:1.33.1@sha256:f7ca5a32c10d51aeda3b4d01c61c6061f497893d7f6628b92f822f7117182a57
=> => resolve docker.io/library/busybox:1.33.1@sha256:f7ca5a32c10d51aeda3b4d01c61c6061f497893d7f6628b92f822f7117182a57
=> [2/3] RUN echo "Hola Mundo ${VERSION:-latest}" > /home/imagen_version
=> [3/3] WORKDIR /home
=> => exporting to image
=> => exporting layers
=> => exporting manifest sha256:d115d282624669ad0874b7a22ffa44f48007cdc6c9a8674e4a3a8245bf48671
=> => exporting config sha256:77aea8d67c83d44edcae7ed372ef3a2f81e64225e22c94ad66536da5731c05c
=> => exporting attestation manifest sha256:37bacd6ce9623d81b9b63f57751bffd20338fecd5f10f0e1f4111c6ef32314ce
=> => exporting manifest list sha256:a9eddef90a1196d3d9ee29b880bed1f2f8820b5c94b794eca95b5cdafe0cf718
=> => naming to docker.io/library/ejemversion:latest
=> => unpacking to docker.io/library/ejemversion:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/kupqaypo7yza76aw1twzhkjhg
```

].

```
docker build --build-arg VERSION=1.33.1 -t ejemversion .
```

15. O simplemente ejecutar el comando sin más. *[Capturas de pantallas:]*

```
MacBook-Pro-de-Jose:Ejercicio-14 josefranciscumurciafuentes$ docker build -t ejemversion .
[+] Building 1.1s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 206B
=> [internal] load metadata for docker.io/library/busybox:latest
=> [auth] library/busybox:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/busybox:latest@sha256:a5d0ce49aa801d475da48f8cb163c354ab95cab073cd
=> => resolve docker.io/library/busybox:latest@sha256:a5d0ce49aa801d475da48f8cb163c354ab95cab073cd
=> CACHED [2/3] RUN echo "Hola Mundo $VERSION" > /home/imagen_version
=> CACHED [3/3] WORKDIR /home
=> => exporting to image
=> => exporting layers
=> => exporting manifest sha256:69d2ae2e55b0bf4019f391662f34f4c059ce8dd4bf82a160c63f3aecf67c4323
=> => exporting config sha256:118e6a9d3f768c592edf011ed268887d459af1dd748b02fe92cc68bb118bdfd2
=> => exporting attestation manifest sha256:6d40b119e66852bf94213891d5673d8d6583cec2fda5aefebb110a
=> => exporting manifest list sha256:02d1ac3960852b4190dfd62723f75ea487c63493339df77874492a4f85b1a
=> => naming to docker.io/library/ejemversion:latest
=> => unpacking to docker.io/library/ejemversion:latest
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/s7movyf49c2vy2150v19k0w2p
```

].

```
docker build -t ejemversion .
```

Para lanzar la ejecución de un contenedor a partir de esta imagen empleamos: *[Capturas de pantalla resultado:]*

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/kupqaypo7yza76aw1twzhkjhg
MacBook-Pro-de-Jose:Ejercicio-14 josefranciscumurciafuentes$ docker run -it ejemversion
Hola Mundo latest
```

].

```
docker run -it ejemversion
```

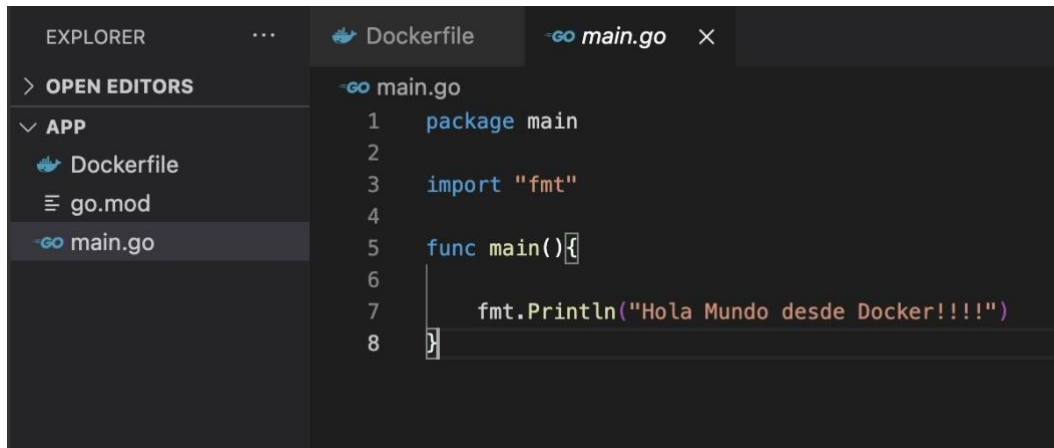
Una vez ejecutado el ejemplo **ejemversion** podemos comprobar el resultado de dicha ejecución en el fichero **/home/imagen_version**. *[Capturas de pantallas.]*

2.6. Multi-stage build

El **multi-stage build** es una técnica que te permite construir diferentes etapas en un solo *Dockerfile*. Puedes utilizar múltiples etapas para compilar y construir tu aplicación en una etapa

y luego copiar solo los archivos necesarios en una etapa posterior. Esto ayuda a reducir el tamaño final de la imagen y a separar las dependencias de desarrollo de las de producción.

15. Creamos un fichero llamado **main.go** con un sencillo código en **Go**. [\[Capturas de pantallas del fichero :](#)

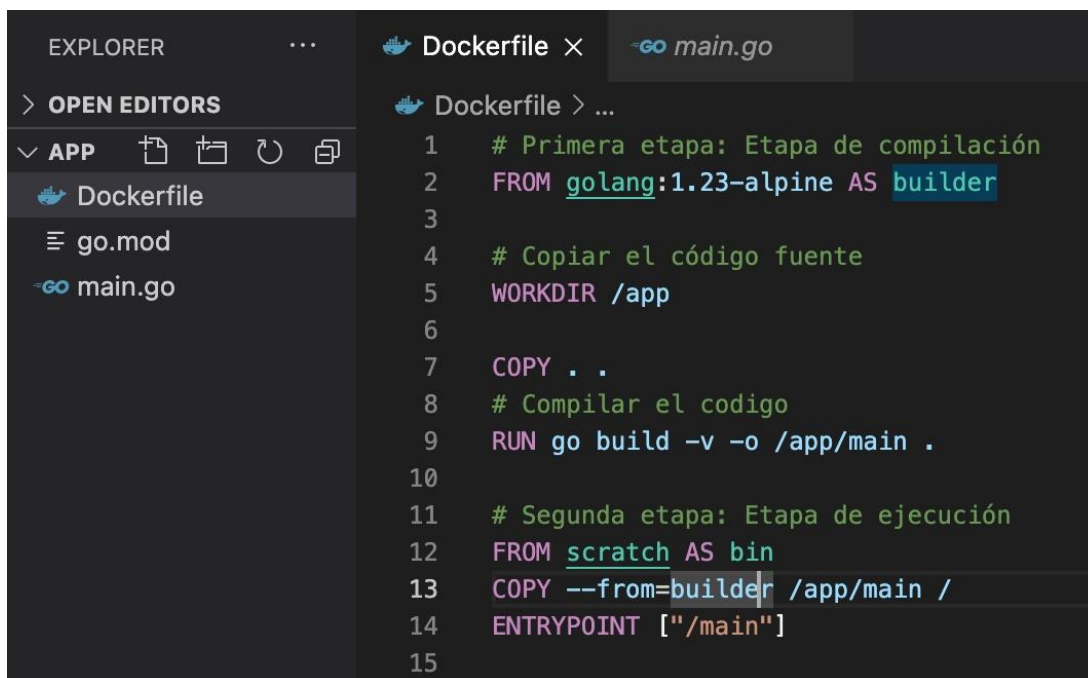


The screenshot shows the VS Code interface with the Explorer on the left and the main.go file open in the editor. The Explorer shows the 'APP' folder containing 'Dockerfile', 'go.mod', and 'main.go'. The main.go file contains the following Go code:

```
1 package main
2
3 import "fmt"
4
5 func main(){
6
7     fmt.Println("Hola Mundo desde Docker!!!!")
8 }
```

].

16. Editamos siguiente fichero **Dockerfile** de ejemplo de una compilación **multi-stage**: [\[Capturas de pantalla Dockerfile:](#)



The screenshot shows the VS Code interface with the Explorer on the left and the Dockerfile file open in the editor. The Explorer shows the 'APP' folder containing 'Dockerfile', 'go.mod', and 'main.go'. The Dockerfile file contains the following multi-stage Dockerfile:

```
1 # Primera etapa: Etapa de compilación
2 FROM golang:1.23-alpine AS builder
3
4 # Copiar el código fuente
5 WORKDIR /app
6
7 COPY . .
8
9 # Compilar el código
10 RUN go build -v -o /app/main .
11
12 # Segunda etapa: Etapa de ejecución
13 FROM scratch AS bin
14 COPY --from=builder /app/main /
15 ENTRYPOINT ["/main"]
```

].


```
# Primera etapa: Etapa de compilación
FROM golang:1.14.2-alpine AS builder
# Copiar el código fuente
WORKDIR /src COPY
. .
# Compilar el código
RUN go build -o /out/hola .

# Segunda etapa: Etapa de ejecución
FROM scratch AS bin
COPY --from=builder /out/hola /
ENTRYPOINT ["/hola"]
```

En este ejemplo, hemos utilizado una etapa de compilación para compilar nuestra aplicación **Go** y una etapa de producción para crear la imagen final del contenedor.

17. Construimos y ejecutamos la imagen. *[Capturas de pantallas de la construcción y la ejecución de la imagen.]*

```
MacBook-Pro-de-Jose:app josefranciscumurciafuentes$ docker build -t holago .
[+] Building 2.9s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 337B                                0.0s
=> [internal] load metadata for docker.io/library/golang:1.23-alpine 0.6s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [builder 1/4] FROM docker.io/library/golang:1.23-alpine@sha256:47d337594bd9e667d35514b241569f95fb6 0.0s
=> => resolve docker.io/library/golang:1.23-alpine@sha256:47d337594bd9e667d35514b241569f95fb6d95727c2 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 388B                                       0.0s
=> CACHED [builder 2/4] WORKDIR /app                               0.0s
=> [builder 3/4] COPY . .                                          0.0s
=> [builder 4/4] RUN go build -v -o /app/main .                    2.1s
=> CACHED [bin 1/1] COPY --from=builder /app/main /                0.0s
=> exporting to image                                              0.0s
=> => exporting layers                                              0.0s
=> => exporting manifest sha256:aeeb3c973074c7156ba4807fbfc72cac1024b644d437e3026cb7c4b7e7cdd3cf 0.0s
=> => exporting config sha256:4ee8bfea5767eb7051997c99a0050775e501bbd782862a4ef0263dc470660795 0.0s
=> => exporting attestation manifest sha256:6c3b027756043942226ef0ac509faf8cd87198d95f722ead87be52f71 0.0s
=> => exporting manifest list sha256:e137aec05f077dbc72e8af33bbd363b59e6da3f2ecde50a9eb12c7b348683323 0.0s
=> => naming to docker.io/library/holago:latest                    0.0s
=> => unpacking to docker.io/library/holago:latest                  0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/r0f53o9fm3nyy7jddqtk5g4nq
MacBook-Pro-de-Jose:app josefranciscumurciafuentes$ docker run -it holago
Hola Mundo desde Docker!!!!
MacBook-Pro-de-Jose:app josefranciscumurciafuentes$
```

].

```
docker build -t holaGo Docker
run -it holaGo.
```

3. Practicamos lo aprendido

Vamos a realizar una instalación de **Nginx** mediante **Dockerfile**.

18. Creamos un fichero **index.html** en nuestro directorio actual y lo publicaremos mediante un servidor web **Nginx**. [\[Capturas de pantallas index.html\]](#)

```
<> index.html x Dockerfile
appnginx > <> index.html > html > body > p
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Puesta en Producción Segura -Dockerfile</title>
7  </head>
8  <body>
9    <h1> Puesta en Producción Segura - Dockerfile </h1>
10   <p>Servidor Nginx desplegado por José Francisco Murcia Fuentes para el módulo de PePS</p>
11
12 </body>
13 </html>
```

].

19. Implementamos el fichero **Dockerfile** que nos permitirá poner en marcha el servidor.

[\[Capturas de pantalla Dockerfile\]](#)

```
<> index.html Dockerfile x
appnginx > Dockerfile > ...
1  #Utilizamos la imagen oficial de nginx en su ultima versión.
2  FROM nginx:alpine
3
4  #Copiamos el archivo index.html al directorio de trabajo.
5  COPY index.html /usr/share/nginx/html/index.html
6
7  |
```

].

Para poder ejecutar el fichero **Dockerfile** emplearemos el comando **docker build**. El parámetro **-t** nos permite dar un nombre y tag a la imagen que estamos construyendo. Si no lo especificamos, **Docker** le asignará un nombre aleatorio.

```
docker build -t <nombre_imagen> <directorio_contexto>
```

20. Construimos la imagen del servidor web **Nginx**: [\[Captura de pantalla\]](#)

```
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$ docker build -t jfm-mi-nginx .
[+] Building 3.3s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 228B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [auth] library/nginx:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 430B
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:814a8e88df978ade80e584cc5b333144b9372a8e3c98872d07137dbf3b44d0e4
=> => resolve docker.io/library/nginx:alpine@sha256:814a8e88df978ade80e584cc5b333144b9372a8e3c98872d07137dbf3b44d0e4
=> => sha256:57520d670e0d8bc245b21cc19bba2075c8898a3c5e47df542f48fb322863279c 15.64MB / 15.64MB
=> => sha256:876850e427bb633799d790f32738cc2577159ed8acef5a1b35cf81a2a3b6cf70 1.40kB / 1.40kB
=> => sha256:685f57786a721fd2c076258ea2df427b5152f13ee322a6e17216b276397eba4a 1.21kB / 1.21kB
=> => sha256:5d89794bca8fd43a9206b2a53125c3edc4c4a2fbd8d4e795421b61ac067aa02d 404B / 404B
=> => sha256:9b12d344c7bc2a044da5530d98159ef627a396ce731ec02e02b4779c4b37ef58 954B / 954B
=> => sha256:bb6bc1d08cd9f3ef90a462e966930f90b203297392596a7810bd91f1fc8d736d 628B / 628B
=> => sha256:34f467366d7ed3eeecf4db9f420b1af3fcca7a5738bf972b1e99a6f1de30697 1.81MB / 1.81MB
=> => sha256:0152682790bba2052e0b7dc52872083c01ea53c598fe87e3fe3eab5f9d4228cb 4.09MB / 4.09MB
=> => extracting sha256:0152682790bba2052e0b7dc52872083c01ea53c598fe87e3fe3eab5f9d4228cb
=> => extracting sha256:34f467366d7ed3eeecf4db9f420b1af3fcca7a5738bf972b1e99a6f1de30697
=> => extracting sha256:bb6bc1d08cd9f3ef90a462e966930f90b203297392596a7810bd91f1fc8d736d
=> => extracting sha256:9b12d344c7bc2a044da5530d98159ef627a396ce731ec02e02b4779c4b37ef58
=> => extracting sha256:5d89794bca8fd43a9206b2a53125c3edc4c4a2fbd8d4e795421b61ac067aa02d
=> => extracting sha256:685f57786a721fd2c076258ea2df427b5152f13ee322a6e17216b276397eba4a
=> => extracting sha256:876850e427bb633799d790f32738cc2577159ed8acef5a1b35cf81a2a3b6cf70
=> => extracting sha256:57520d670e0d8bc245b21cc19bba2075c8898a3c5e47df542f48fb322863279c
=> [2/2] COPY index.html /usr/share/nginx/html/index.html
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:883354f28729be854985f3efaeae1fec07283cab8c527cbd7a8fcad901d099be
=> => exporting config sha256:c57cc893406bc81b0142118d0ce4b29fff7c871ae6e984a16bd12236af30ee08
=> => exporting attestation manifest sha256:2b4555e334074feaeab905ab9380701cd0b2061f4856b8be1293bd81386efba
=> => exporting manifest list sha256:9c67a238f493a911335bd76ad7ad6367938c5566565c1d376441250ed801c80c
=> => naming to docker.io/library/jfm-mi-nginx:latest
=> => unpacking to docker.io/library/jfm-mi-nginx:latest
```

View build details: [docker-desktop://dashboard/build/desktop-linux/desktop-linux/y4gi07ap5ic0yyaguhimej27q](#)

].

```
$ docker build -t jg-mi-nginx .
```

21. Si todo ha ido bien, ya deberías tener tu imagen construida. Podemos comprobarlo con el comando: [\[Capturas de pantalla\]](#)

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/y4gi07ap5ic0yyaguhimej27q
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
jfm-mi-nginx         latest     9c67a238f493  18 seconds ago  76.8MB
holago              latest     e137aec05f07  50 minutes ago  3.4MB
ejemversion         latest     a9eddef90a11  21 hours ago   2.35MB
hola                latest     4d8d86ca0ac7  22 hours ago   12.8MB
opt2-ubuntu-python latest     66000ec75807  24 hours ago   805MB
my-ubuntu-python    latest     a4cc12fe2336  2 days ago     805MB
debian-apachemiweb-jfmf latest     b8aa9d7c1af2  2 weeks ago    417MB
mialpine            latest     d261f41acd19  3 weeks ago    12.8MB
miubuntu            latest     b4a46d8167d6  3 weeks ago    216MB
```

].

```
$ docker images.
```

22. Ahora solo nos queda probar las imágenes. Para ello, ejecutamos un contenedor a partir de la imagen: [\[Capturas de pantallas\]](#).

```
$ docker run -d --name jg-mi-nginx -p 8080:80 jg-mi-nginx
```


23. Abrimos la página web servida por Nginx en un navegador. [\[Capturas de pantalla\]](#)



].

3.1. KaliLinux en Docker

3. Contenedor Docker KaliLinux

24. Descargamos la imagen de Kali Linux para Docker. [\[Capturas de pantalla\]](#)

```
MacBook-Pro-de-Jose:appkali josefranciscomurciafuentes$ docker pull kalilinux/kali-rolling
Using default tag: latest
latest: Pulling from kalilinux/kali-rolling
Digest: sha256:4c516126b5ef76dbf1423366853862c0d885e3d3d8f02fd0ab966cce1c6e11ab
Status: Image is up to date for kalilinux/kali-rolling:latest
docker.io/kalilinux/kali-rolling:latest
MacBook-Pro-de-Jose:appkali josefranciscomurciafuentes$
```

].

```
$ docker pull kalilinux/kali-rolling
```

25. Comprobamos que la imagen de Kali se ha descargado. [\[Captura de pantalla\]](#)

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jfm-mi-nginx	latest	9c67a238f493	28 minutes ago	76.8MB
holago	latest	e137aec05f07	About an hour ago	3.4MB
ejemversion	latest	a9eddef90a11	21 hours ago	2.35MB
hola	latest	4d8d86ca0ac7	22 hours ago	12.8MB
opt2-ubuntu-python	latest	66000ec75807	24 hours ago	805MB
my-ubuntu-python	latest	a4cc12fe2336	3 days ago	805MB
kalilinux/kali-rolling	latest	4c516126b5ef	12 days ago	222MB
debian-apachemiweb-jfmf	latest	b8aa9d7c1af2	2 weeks ago	417MB
mialpine	latest	d261f41acd19	3 weeks ago	12.8MB
miubuntu	latest	b4a46d8167d6	3 weeks ago	216MB
my-alpine-python	latest	ea748ce8acb	5 weeks ago	110MB
alpine-python	v1.0	782fb5d43a1a	5 weeks ago	115MB
ubuntu-python	v1.0	ae563b060e37	5 weeks ago	823MB
node	latest	0b50ca11d81b	5 weeks ago	1.6GB
alpine	latest	21dc6063fd67	6 weeks ago	12.8MB
ping	google	cdb4d38b95cb	6 weeks ago	12.8MB
ping	youtube	a254755dd847	6 weeks ago	12.8MB
python	latest	9255d1993f6d	6 weeks ago	1.47GB

].

```
$ docker images
```

26. Ejecutamos la imagen descargada. Vamos a personalizar el nombre del contenedor y el nombre de la máquina con nuestras propias iniciales. [\[Captura de pantalla\]](#)

```
MacBook-Pro-de-Jose:appkali josefranciscoscurciafuentes$ docker run -it --name jfmf-kalilinux --hostname jfmf-kali-peps kalilinux/kali-rolling
(root@jfmf-kali-peps)-[/]
```

].

```
$ docker run -it --name jg-kalilinux --hostname jg-kali-peps kalilinux/kali-rolling
```

27. Actualizamos la máquina Kali que se está ejecutando en el contenedor. [\[Capturas de pantallas\]](#)

```
MacBook-Pro-de-Jose:appkali josefranciscoscurciafuentes$ docker run -it --name jfmf-kalilinux --hostname jfmf-kali-peps kalilinux/kali-rolling
(root@jfmf-kali-peps)-[/]
# apt update -y
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/non-free arm64 Packages [151 kB]
Get:3 http://kali.download/kali kali-rolling/non-free-firmware arm64 Packages [9707 B]
Get:4 http://kali.download/kali kali-rolling/contrib arm64 Packages [96.4 kB]
Get:5 http://kali.download/kali kali-rolling/main arm64 Packages [20.1 MB]
Fetched 20.4 MB in 2s (13.0 MB/s)
19 packages can be upgraded. Run 'apt list --upgradable' to see them.

(root@jfmf-kali-peps)-[/]
# apt upgrade -y
Upgrading:
apt base-passwd diffutils gcc-14-base libcom-err2 libext2fs2t64 liblz4-1 libstdc++6 libudev1 sysvinit-utils
base-files dash e2fsprogs libapt-pkg6.0t64 libdebconfclient0 libgcc-s1 libss2 libsystemd0 logsave

Installing dependencies:
sqv

Summary:
Upgrading: 19, Installing: 1, Removing: 0, Not Upgrading: 0
Download size: 5690 kB
Space needed: 1690 kB / 48.2 GB available
```

].

```
root@jg-kali-peps# apt update -y root@jg-kali-peps# apt upgrade -y
```

28. Instalamos las herramientas para *pentesting* web: **nmap**, **nikto**. [\[Capturas de pantallas\]](#)

```
(root@jfmf-kali-peps)-[/]
# apt install -y nmap nikto
Installing:
nikto nmap

Installing dependencies:
adduser dbus-daemon libapparmor1 libexpat1 liblinear4 libpcap0.8t64 netbase perl-modules-5.40
dbus dbus-session-bus-common libblas3 libgdbm-compat4t64 liblua5.4-0 libperl5.40 nmap-common perl-openssl-defaults
dbus-bin dbus-system-bus-common libdbus-1-3 libgdbm6t64 libnet-ssleay-perl libssh2-1t64 perl

Suggested packages:
liblocale-gettext-perl default-dbus-session-bus liblinear-tools ncat perl-doc make
cron | dbus-session-bus liblinear-dev ndiff libterm-readline-gnu-perl libtap-harness-archive-perl
quota gdbm-l10n sensible-utils zenmap | libterm-readline-perl-perl debhelper

Summary:
Upgrading: 0, Installing: 25, Removing: 0, Not Upgrading: 0
Download size: 16.0 MB
Space needed: 89.8 MB / 48.2 GB available
```

].

```
root@jg-kali-peps# apt install -y nmap nikto
```

29. Comprobamos que se han instado correctamente las aplicaciones. [\[Captura de pantalla\]](#)

```
(root@jfmf-kali-peps)-[/]
# nmap version
Starting Nmap 7.95 ( https://nmap.org ) at 2025-01-18 16:49 UTC
Failed to resolve "version".
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 0.09 seconds

(root@jfmf-kali-peps)-[/]
# nikto version
- Nikto v2.5.0
```

].

```
root@jg-kali-peps# nmap
root@jg-kali-peps# nikto
```

30. Confirmamos los cambios realizados sobre el contenedor en una nueva imagen.

[\[Capturas de pantallas\]](#)

```
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$ docker ps
CONTAINER ID   IMAGE                                COMMAND      CREATED        STATUS        PORTS          NAMES
dae0fe999fec   kalilinux/kali-rolling              "bash"      23 hours ago   Up 23 hours   jfmf-kalilinux
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$ docker commit jfmf-kalilinux jfmf-kali-web
sha256:144814ab21cdeb8526672bcd1bbe6b5bb142624085fda78be8c1c29335e59727
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$
```

].

```
$ docker commit jg-kalilinux jg-kali-web
```

31. Comprobamos el tamaño de las diferentes imágenes de Kalilinux disponibles en nuestro equipo. [\[Capturas de pantalla\]](#)

```
MacBook-Pro-de-Jose:appnginx josefranciscomurciafuentes$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
jfmf-kali-web        latest     144814ab21cd  About a minute ago  433MB
jfm-mi-nginx         latest     9c67a238f493  23 hours ago    76.8MB
holago               latest     e137aec05f07  24 hours ago    3.4MB
ejemversion          latest     a9eddef90a11  44 hours ago    2.35MB
hola                 latest     4d8d86ca0ac7  45 hours ago    12.8MB
opt2-ubuntu-python   latest     66000ec75807  47 hours ago    805MB
my-ubuntu-python     latest     a4cc12fe2336  3 days ago      805MB
kalilinux/kali-rolling latest     4c516126b5ef  13 days ago     222MB
debian-apachemiweb-jfmf latest     b8aa9d7c1af2  2 weeks ago     417MB
```

].

```
$ docker images
```

32. Detenemos el contenedor *kalilinux* (en mi caso se trata de *jg-kalilinux*). [

Captura de pantalla

```
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$ docker ps
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
dae0fe999fec   kalilinux/kali-rolling "bash"          23 hours ago     Up 23 hours           jfmf-kalilinux
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$ docker stop jfmf-kalilinux
jfmf-kalilinux
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$ docker ps
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$
```

].

```
$ docker stop jg-kalilinux
```

33. Volvemos a lanzar el contenido que hemos detenido y volvemos a comprobar que tiene las aplicaciones instaladas. [Capturas de pantalla arranque, ejecución y comprobación de aplicaciones instaladas.

```
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$ docker start jfm-mi-nginx
jfm-mi-nginx
MacBook-Pro-de-Jose:appnginx josefranciscumurciafuentes$ docker run -it --name kali-web --hostname kali jfmf-kali-web
(root@kali)-[/]
# nmap
Nmap 7.95 ( https://nmap.org )
```

```
MacBook-Pro-de-Jose:appkali josefranciscumurciafuentes$ docker ps
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
264b3b9ef5b5   jfmf-kali-web  "bash"          8 minutes ago    Up 8 minutes           kali-web
acdb525ddc52   jfm-mi-nginx   "/docker-entrypoint..." 25 hours ago     Up 2 hours           0.0.0.0:8080->80/tcp   jfm-mi-nginx
MacBook-Pro-de-Jose:appkali josefranciscumurciafuentes$ docker exec -it kali-web
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.
```

```
Usage: docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

```
Execute a command in a running container
```

```
MacBook-Pro-de-Jose:appkali josefranciscumurciafuentes$ docker exec -it kali-web bash
```

```
(root@kali)-[/]
```

```
#
```

```
MacBook-Pro-de-Jose:appkali josefranciscumurciafuentes$ docker exec -it kali-web bash
(root@kali)-[/]
# nmap -Version
Nmap version 7.95 ( https://nmap.org )
Platform: aarch64-unknown-linux-gnu
Compiled with: liblua-5.4.7 openssl-3.3.2 libssh2-1.11.1 libz-1.3.1 libpcap-1.10.5 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

```
(root@kali)-[/]
```

```
# nikto -Version
```

```
Nikto 2.5.0 (LW 2.5)
```

```
(root@kali)-[/]
```

```
# exit
```

```
exit
```

```
MacBook-Pro-de-Jose:appkali josefranciscumurciafuentes$
```

].

```
$ docker start jg-kalilinux
$ docker exec -it jg-kalilinux bash
root@jg-kali-peps# nmap
root@jg-kali-peps# nikto
root@jg-kali-peps# exit
```


4. Dockerfile para KaliLinux

34. Preparamos un fichero *Dockerfile* que nos permita desplegar el sistema KaliLinux con las herramientas necesarias para realizar *pentesting* web. [\[Captura de pantalla Dockerfile\]](#)

```
Dockerfile X
appkali > Dockerfile > ...
1 FROM kalilinux/kali-rolling
2
3 ENV HOSTNAME=kali-PePS-jfmf
4
5 ENV DEBIAN_FRONTEND=noninteractive
6
7 WORKDIR /appkali
8
9 #Instalacion y actualizacion de KALI
10 RUN apt -y update && apt -y dist-upgrade && apt -y autoremove && apt clean
11
12 #Instalacion de Herramientas
13 RUN apt-get install -y \
14 hydra \
15 john \
16 metasploit-framework \
17 nmap \
18 sqlmap \
19 wfuzz \
20 exploitdb \
21 nikto
22
23 #Install utilidades
24 RUN apt-get install -y \
25 vim \
26 git
27
28 # Setup gdb-peda
29 RUN git clone https://github.com/longld/peda.git ~/peda && \
30 echo "source ~/peda/peda.py" >> ~/.gdbinit
31
32 # Instalamos paquetes necesarios para proporcionar interfaz gráfica a KaliLinux
33 RUN apt install -y kali-desktop-xfce xrdp dbus-x11 && service xrdp start
34
35 EXPOSE 3389
36
37 CMD [ "/bin/bash" ]
```

].

35. Construimos y ejecutamos un contenedor con la imagen KaliLinux personalizada.

```
MacBook-Pro-de-Jose:appkali josefranciscoturciafuentes$ docker build -t mi-kali-jfmf .
[+] Building 0.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> [internal] load metadata for docker.io/kalilinux/kali-rolling:latest
=> [internal] load .dockerignore
=> [internal] load context: 2B
=> [1/7] FROM docker.io/kalilinux/kali-rolling:latest@sha256:4c516126b5ef76dbf1423366853862c0d885e3d3d8f02fd0ab966cce1c6e11a
=> resolve docker.io/kalilinux/kali-rolling:latest@sha256:4c516126b5ef76dbf1423366853862c0d885e3d3d8f02fd0ab966cce1c6e11a
=> CACHED [2/7] WORKDIR /appkali
=> CACHED [3/7] RUN apt -y update && apt -y dist-upgrade && apt -y autoremove && apt clean
=> CACHED [4/7] RUN apt-get install -y hydra john metasploit-framework nmap sqlmap wfuzz exploitdb nikto
=> CACHED [5/7] RUN apt-get install -y vim git
=> CACHED [6/7] RUN git clone https://github.com/longld/peda.git ~/peda && echo "source ~/peda/peda.py" >> ~/.gdbinit
=> CACHED [7/7] RUN apt install -y kali-desktop-xfce xrdp dbus-x11 && service xrdp start
=> exporting to image
=> exporting layers
=> exporting manifest sha256:3c6b0673634a7cb8e45a1883c0ad05e0ad9f383e8c6a121c161584a67ff773f5
=> exporting config sha256:56ecdf3bc2319b11712911d8f04c5093c803bfff7c4e4ac6c8f0355f161efc4f8
=> exporting attestation manifest sha256:be61975b0820a137eddb5a7eb0d41df41ca5d37f326104a45623f7b1cbf90cc5
=> exporting manifest list sha256:9d7456415d08baa7aeab218612457a8312231e8ccad1de196452be09d9b9e4af
=> naming to docker.io/library/mi-kali-jfmf:latest
=> unpacking to docker.io/library/mi-kali-jfmf:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/6dkqda7208of3pvuch7iwpvc
MacBook-Pro-de-Jose:appkali josefranciscoturciafuentes$
```

```
MacBook-Pro-de-Jose:appkali josefranciscoturciafuentes$ docker run --name jfmf-kali-PePS --hostname kali-PePS-jfmf -it mi-kali-jfmf
(root@kali-PePS-jfmf)-[/appkali]
# passwd root
```

```
MacBook-Pro-de-Jose:appkali josefranciscoturciafuentes$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
mi-kali-jfmf        latest      9d7456415d08 12 minutes ago 7.22GB
jfmf-kali-web       latest      144814ab21cd 24 hours ago   433MB
jfm-mi-nginx        latest      9c67a238f493 47 hours ago   76.8MB
holago              latest      e137aec05f07 2 days ago     3.4MB
```

```
docker build -t mi-kali-jg .
```

```
docker run -name jg-kali-PePS --hostname jg-ciber-PePS-it mi-kali-jg
```

36. Antes de acceder al **Escritorio** desde un cliente **RDP** es muy recomendable cambiar la contraseña del usuario `root` de la maquina KaliLinux que se está ejecutando en el contenedor. *[Captura de pantalla cambio contraseña root]*

```
(root@kali-PePS-jfmf)-[/appkali]
# passwd root
New password:
Retype new password:
passwd: password updated successfully
```

].

```
# passwd root
```

37. También, será necesario iniciar el servicio `xrdp`. *[*

-Capturas de pantalla de la configuración del servicio de escritorio remoto xrdp:

`sudo update-alternatives --config x-session-manager`

```
(root@jfmf-kali-peps)-[/appkali]
# sudo update-alternatives --config x-session-manager
There are 2 choices for the alternative x-session-manager (providing /usr/bin/x-session-manager).

  Selection    Path                                Priority  Status
  * 0          /usr/bin/startxfce4                50      auto mode
    1          /usr/bin/startxfce4                50      manual mode
    * 2        /usr/bin/xfce4-session              40      manual mode

Press <enter> to keep the current choice[*], or type selection number: 2
```

-Configuramos el archivo `xrdp.ini` modificamos:

autorun=sesman-any

max_bpp=16

-Para finalizar debemos agregar al final del documento

[sesman-any]

ip=127.0.0.1

```
(root@jfmf-kali-peps)-[/appkali]
# sudo nano /etc/xrdp/xrdp.ini
```

-Para finalizar configuramos los servicios para que se activen en el inicio:

```
sudo update-rc.d xrdp enable
```

```
sudo systemctl enable xrdp-sesman.service
```

```
(root@jfmf-kali-peps)-[/appkali]
# sudo systemctl enable xrdp-sesman.service
Created symlink '/etc/systemd/system/multi-user.target.wants/xrdp-sesman.service' → '/usr/lib/systemd/system/xrdp-sesman.service'.
```

-Reinicio del servicio xrdp .

```

❏ (root@jfmf-kali-peps)-[/appkali]
# service xrdp stop
Stopping Remote Desktop Protocol server: xrdpxrdp[10]: [INFO ] Received termination signal, stopping the server accept new connections thread

xrdp-sesman.

❏ (root@jfmf-kali-peps)-[/appkali]
# service xrdp start
Starting Remote Desktop Protocol server: xrdp-sesmanxrdp-sesman[197]: [INFO ] starting xrdp-sesman with pid 197

xrdp.

```

Fuentes:

<https://youtu.be/Yx-Hqsw7el8?si=Wm1TvcrOeMz0v72h>

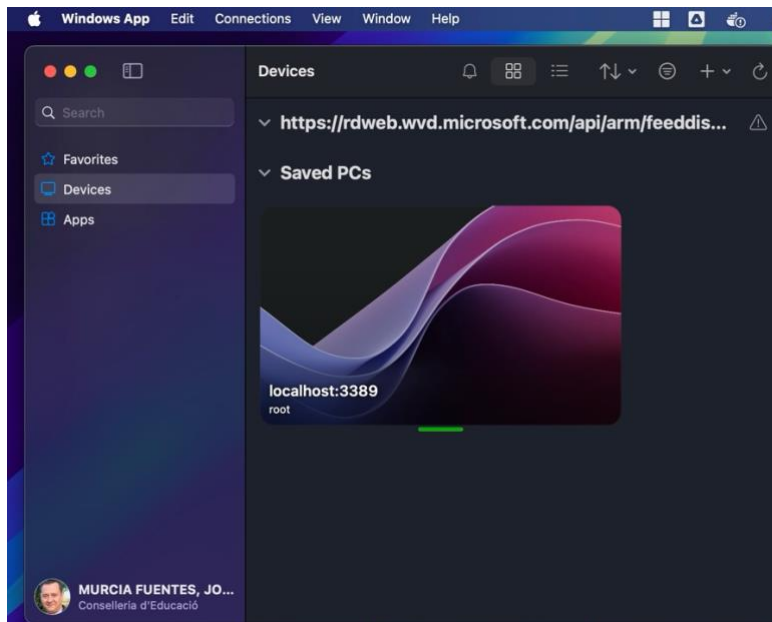
<https://www.blackmantisecurity.com/acceso-grafico-a-kali-linux-por-remote-desktop-protocol/>

1.

```
# service xrdp start
```

5. Instalación de un cliente RDP en el host anfitrión

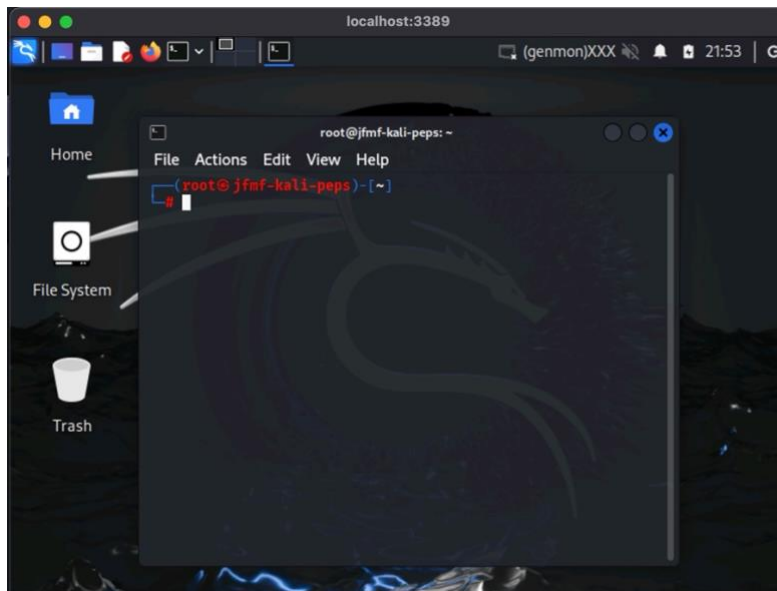
38. En una máquina externa o en el host anfitrión instalamos un Escritorio Remoto para poder acceder al Escritorio gráfico de la máquina KaliLinux. *[He utilizado el escritorio remoto de Windows para Mac en el host anfitrión:*



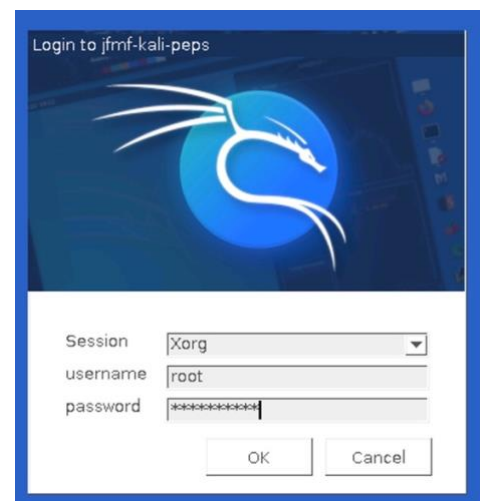
].

```
apt update && apt install -y remmina
```

39. Para conectar con el Escritorio Kali Linux en el cliente RDP introduciremos. **RDP: dirección_IP_kali-PePS:3389**. *[Capturas de pantallas de Kali desde escritorio remoto:*



].



3.2. Optimización de contenedores

40. Siguiendo los pasos descritos en los videos de la sección de Referencias documenta y explica como realizar la optimización de un contenedor Python con Flask y Anaconda.

[Construimos el contenedor con flask y mostramos el tamaño de la imagen:

Ejemplo: Construcción de una aplicación flask

-Mostramos la ejecución del contenedor sin optimizar Dockerfile.old:

*-Para la optimización del contenedor partimos de una imagen descargada con la etiqueta **Slim** Docker en sus repositorios dispone de imágenes optimizadas y comprimidas en nuestro repositorio ocupan más espacio.*

-Otra de las optimizaciones es agrupar las instrucciones en una sola línea.

*- El concepto de **multi-stage build** en Docker se refiere a una técnica que permite construir imágenes Docker de manera más eficiente y optimizada. Esta técnica es útil para reducir el tamaño de la imagen final al separar las etapas de construcción y ejecución.*

-Con multi-stage, puedes usar varias etapas dentro de un Dockerfile. Cada etapa puede tener su propio entorno, y al final solo se copian los artefactos necesarios de una etapa a otra, eliminando archivos o dependencias intermedias que no son necesarios en la imagen final.

-Explicación del ejemplo:

1.Primer etapa (build):

- Se utiliza la imagen base para compilar la aplicación.

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like `app.py`, `compose.yml`, `Dockerfile`, `Dockerfile.old`, `requirements.txt`, and `miniconda`. The code editor shows the content of `Dockerfile.old`, which is a multi-stage Dockerfile. The build output at the bottom shows the command `docker images` and the resulting image details.

```

flask > Dockerfile.old > FROM
1 FROM python:3-slim-bullseye
2
3 WORKDIR /workspace
4
5 RUN apt-get update && apt-get -y install gcc musl-dev libffi-dev python3-dev openssl
6
7 #Cryptography module's dependencies
8 RUN apt-get -y install rustc cargo
9
10 # Copy the project and install Python dependencies from requirements.txt if it exists
11 COPY requirements.txt .
12 RUN pip install -r requirements.txt
13
14 # Copy code into the container
15 COPY app.py .
16
17 EXPOSE 5000
18 # Run the application
19 CMD ["python", "app.py"]
20
21
  
```

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/t9zpxky97jfupzjf8hzhxjxoj
 MacBook-Pro-de-Jose:flask josefranciscumurciafuentes\$ docker images

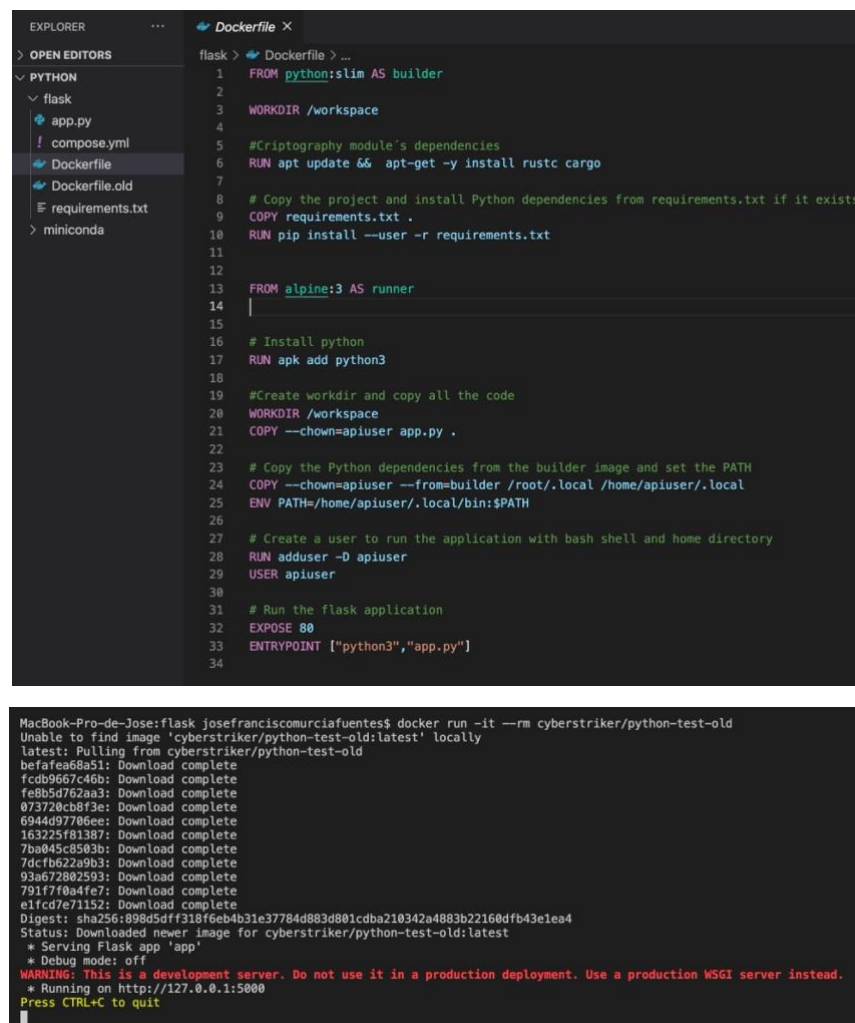
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ciberstriker/python-test-old	latest	5be626bafc9a	About a minute ago	1.15GB

Ventajas de Multi-Stage:

1. *Imágenes más pequeñas: Solo se incluyen los archivos necesarios para la ejecución.*
 2. *Separación de responsabilidades: Cada etapa puede usar diferentes imágenes base según sus necesidades.*
 3. *Mantenimiento más sencillo: Puedes manejar varias etapas en un único Dockerfile.*
- El mismo concepto puede aplicarse a cualquier lenguaje, como Node.js, Python, o Java, ajustando las herramientas y comandos necesarios para cada caso.

2. Segunda etapa (imagen final):

- Se usa una imagen ligera de Alpine para ejecutar la aplicación.
- Solo se copia el binario generado en la primera etapa.
- Esto minimiza el tamaño de la imagen final, ya que elimina las herramientas y dependencias usadas para compilar.



```

EXPLORER
... Dockerfile X
flask > Dockerfile > ...
1 FROM python:slim AS builder
2
3 WORKDIR /workspace
4
5 #Cryptography module's dependencies
6 RUN apt update && apt-get -y install rustc cargo
7
8 # Copy the project and install Python dependencies from requirements.txt if it exists
9 COPY requirements.txt .
10 RUN pip install --user -r requirements.txt
11
12
13 FROM alpine:3 AS runner
14
15
16 # Install python
17 RUN apk add python3
18
19 #Create workdir and copy all the code
20 WORKDIR /workspace
21 COPY --chown=apiuser app.py .
22
23 # Copy the Python dependencies from the builder image and set the PATH
24 COPY --chown=apiuser --from=builder /root/.local /home/apiuser/.local
25 ENV PATH=/home/apiuser/.local/bin:$PATH
26
27 # Create a user to run the application with bash shell and home directory
28 RUN adduser -D apiuser
29 USER apiuser
30
31 # Run the flask application
32 EXPOSE 80
33 ENTRYPOINT ["python3", "app.py"]
34

MacBook-Pro-de-Jose:flask josefranciscoscuriafuentes$ docker run -it --rm cyberstriker/python-test-old
Unable to find image 'cyberstriker/python-test-old:latest' locally
latest: Pulling from cyberstriker/python-test-old
befeae68a51: Download complete
fcd9667c46b: Download complete
fe8b5d762aa3: Download complete
073728cb8f3e: Download complete
6944697786ee: Download complete
163225f81387: Download complete
7ba045c8503b: Download complete
7dcfb622a9b3: Download complete
93a672802593: Download complete
791f7f0a4fe7: Download complete
e1fcd7e71152: Download complete
Digest: sha256:1898d5dff310f6cb4b31e37784d883d801cda210342a4883b22160dfb43e1ea4
Status: Downloaded newer image for cyberstriker/python-test-old:latest
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit

```

```

=> exporting to image
=> exporting layers
=> exporting manifest sha256:ede1bdeab862af72b1dd119be5aa8c8fc38ca4784a11319d42740a3051b4f13
=> exporting config sha256:8eeffc446c46f296f8b49440508a67219e33cf233b54ec3db138e-dbd578667
=> exporting attestation manifest sha256:525ac7d1546513a0a4083a2e8f7a08a2f748c3e777f441a6824c89138b8ebf0
=> exporting manifest list sha256:70db02c6b3cef3d37c99346f509f33bd130f6a51a123049494f533500bba794
=> naming to docker.io/cyberstriker/miniconda-test-old:latest
=> unpacking to docker.io/cyberstriker/miniconda-test-old:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/3me8urfsaizkh47zhpvp23b5e
MacBook-Pro-de-Jose:miniconda josefranciscumrciafuentes$

=> [unknown] load build context
=> transferring context: 814B
=> [runner 1/6] FROM docker.io/library/alpine:3@sha256:56fa17d2a7e7f168a043a2712e63aed1f8543aeafdcce47c58dcffe38ed51099
=> resolve docker.io/library/alpine:3@sha256:56fa17d2a7e7f168a043a2712e63aed1f8543aeafdcce47c58dcffe38ed51099
=> [runner 2/6] RUN apk add python3
=> [runner 3/6] WORKDIR /workspace
=> [runner 4/6] COPY --chown=apiuser app.py .
=> [builder 2/5] WORKDIR /workspace
=> [builder 3/5] RUN apt update && apt-get -y install rustc cargo
=> [builder 4/5] COPY requirements.txt .
=> [builder 5/5] RUN pip install --user -r requirements.txt
=> [runner 5/6] COPY --chown=apiuser --from=builder /root/.local /home/apiuser/.local
=> [runner 6/6] RUN adduser -D apiuser
=> exporting to image
=> exporting manifest sha256:849764374b96629a87f8d744fd4c4e2083246f35c880784d025c12e8e05ec7039
=> exporting config sha256:9346415ee64b94d5a022d65290b1d8e8984e490af02504db2a61a82b3135588
=> exporting attestation manifest sha256:d5048f27c62436dbceac99e8e35809156d3eab2f372245fcb5abf9f437c659
=> exporting manifest list sha256:f44697d878ed189c3a0e728fcbcc004f52a68394be56b780e12ba052f7c2
=> naming to docker.io/cyberstriker/python-test:latest
=> unpacking to docker.io/cyberstriker/python-test:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/vuyji1n9cumotos4j27muunxh
MacBook-Pro-de-Jose:flask josefranciscumrciafuentes$

```

-Captura y comparación de las imágenes después de la optimización donde vemos drásticamente reducido su tamaño:

```

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/vuyji1n9cumotos4j27muunxh
MacBook-Pro-de-Jose:flask josefranciscumrciafuentes$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
cyberstriker/python-test   latest      f44697d878ed     3 minutes ago    138MB
ciberstriker/python-test-old latest      5be626baf9a      20 minutes ago   1.15GB

```

-El contenedor con miniconda captura tamaño de las imágenes y los Dockerfile:

```

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/yar
MacBook-Pro-de-Jose:miniconda josefranciscumrciafuentes$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
cyberstriker/miniconda-test   latest      6e3b0f79b23d     3 days ago       1.45GB
cyberstriker/miniconda-test-old latest      1a86c2b92dd7     3 days ago       3.46GB

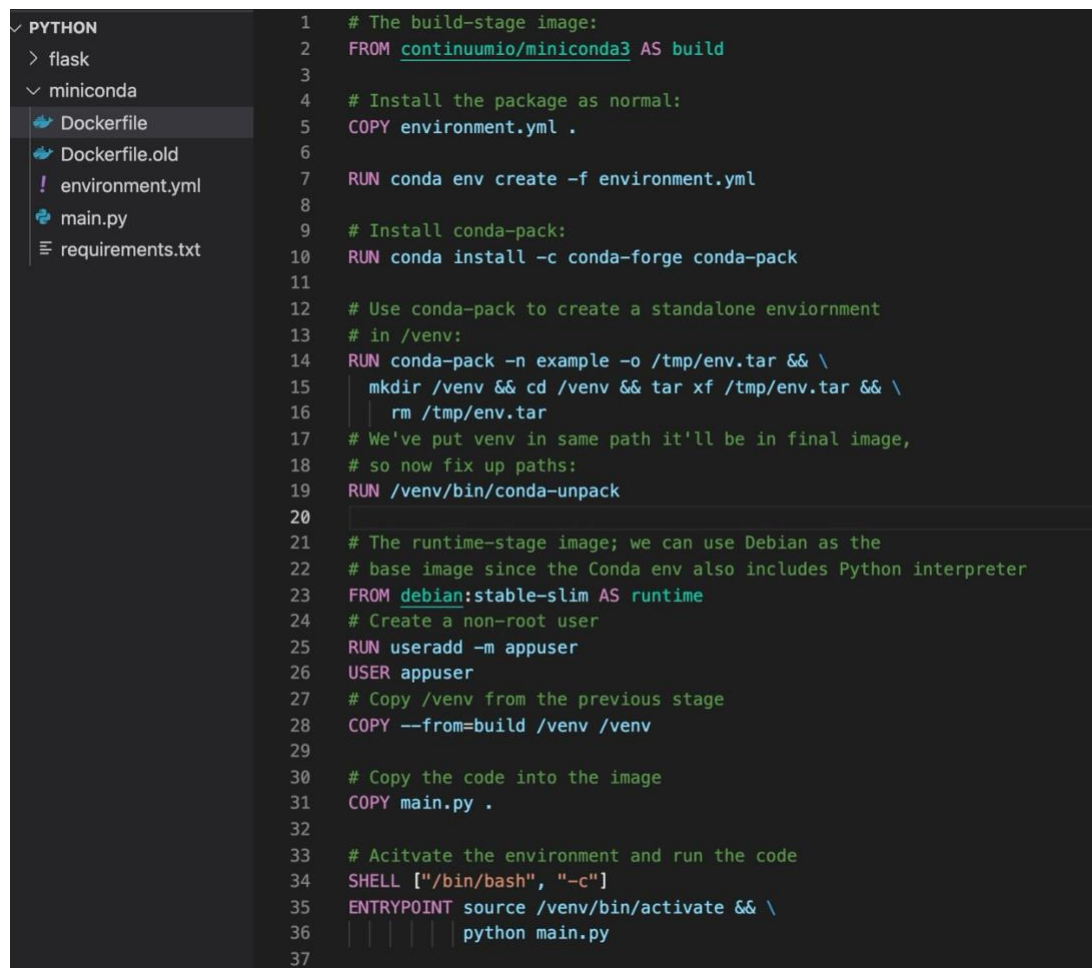
```

```

🐳 Dockerfile.old ● 🐳 Dockerfile

miniconda > 🐳 Dockerfile.old > ...
1  # The build-stage image:
2  FROM continuumio/miniconda3 AS build
3
4  COPY environment.yml .
5
6  RUN conda env create -f environment.yml
7
8  # Install conda-pack:
9  RUN conda install -c conda-forge conda-pack clear
10
11

```

```

1  # The build-stage image:
2  FROM continuumio/miniconda3 AS build
3
4  # Install the package as normal:
5  COPY environment.yml .
6
7  RUN conda env create -f environment.yml
8
9  # Install conda-pack:
10 RUN conda install -c conda-forge conda-pack
11
12 # Use conda-pack to create a standalone environment
13 # in /venv:
14 RUN conda-pack -n example -o /tmp/env.tar && \
15     mkdir /venv && cd /venv && tar xf /tmp/env.tar && \
16     rm /tmp/env.tar
17 # We've put venv in same path it'll be in final image,
18 # so now fix up paths:
19 RUN /venv/bin/conda-unpack
20
21 # The runtime-stage image; we can use Debian as the
22 # base image since the Conda env also includes Python interpreter
23 FROM debian:stable-slim AS runtime
24 # Create a non-root user
25 RUN useradd -m appuser
26 USER appuser
27 # Copy /venv from the previous stage
28 COPY --from=build /venv /venv
29
30 # Copy the code into the image
31 COPY main.py .
32
33 # Activate the environment and run the code
34 SHELL ["/bin/bash", "-c"]
35 ENTRYPOINT source /venv/bin/activate && \
36     python main.py
37

```

].

4. Referencias

4.1. Vídeos

- [YouTube \(@Pabpereza\): Dockerfile y Docker build, construir imágenes, optimizar cache - Curso Docker gratuito en español.](#)
- [YouTube \(@Pabpereza\): Optimizar imágenes de docker / contenedor - Multi stage & Distroless.](#)
- [YouTube \(@Pabpereza\): Optimizar python en docker - flask y anaconda - Multi stage & Distroless.](#)
- [YouTube \(@Pabpereza\): Optimizar Node y Nextjs en docker - Multi stage & Distroless.](#)
- [YouTube \(@Pabpereza\): KaliLinux en Docker \(Parte 1/3\) - Descarga, ejecución y creación de diferentes plantillas.](#)
- [YouTube \(@Pabpereza\): KaliLinux en Docker \(Parte 2/3\) - Dockerfiles y repositorio en hub.docker.com](#)
- [YouTube \(@Pabpereza\): KaliLinux en Docker \(Parte 3/3\) - Interfaz Gráfica.](#)

Resultados de Aprendizaje y Criterios de Evaluación

Contenidos:	
5. Implantación de sistemas seguros de despliegado de software: <ul style="list-style-type: none"> • Puesta segura en producción. • Prácticas unificadas para el desarrollo y operación del software (DevOps). • Sistemas de control de versiones. • Sistemas de automatización de construcción (build). • Escalado de servidores. Virtualización. Contenedores. • Gestión automatizada de configuración de sistemas. • Orquestación de contenedores. 	
Resultados de Aprendizaje	Criterios de Evaluación
RA5. Implanta sistemas seguros de despliegado de software, utilizando herramientas para la automatización de la construcción de sus elementos.	<p>a) Se han identificado las características, principios y objetivos de la integración del desarrollo y operación del software.</p> <p>b) Se han implantado sistemas de control de versiones, administrando los roles y permisos solicitados.</p> <p>c) Se han instalado, configurado y verificado sistemas de integración continua, conectándolos con sistemas de control de versiones.</p> <p>d) Se han planificado, implementado y automatizado planes de despliegado de software.</p> <p>e) Se ha evaluado la capacidad del sistema desplegado para reaccionar de forma automática a fallos.</p> <p>f) Se han documentado las tareas realizadas y los procedimientos a seguir para la recuperación ante desastres.</p> <p>g) Se han creado bucles de retroalimentación ágiles entre los miembros del equipo.</p>