Joám Ferreiro Tobío

PPS – DevSecOps

Ejercicio 1

1. Crear un directorio para el proyecto PHP

Crea un nuevo directorio en tu sistema anfitrión donde almacenarás tu aplicación PHP y el Dockerfile. Esto ayuda a organizar los archivos necesarios para construir la imagen Docker.

```
joamft@joamft-VirtualBox:~$ mkdir php-app
joamft@joamft-VirtualBox:~$ cd php-app/
```

2. Crear un archivo PHP sencillo

Dentro del directorio creado, crea un archivo PHP (index.php) con un código simple que imprime un mensaje. Este archivo será la aplicación que se ejecutará dentro del contenedor Docker.

```
 GNU nano 6.2                                                    index.php
<?php
echo "Hello, Rafa, Wololo!";
?>
```

3. Crear un Dockerfile

El Dockerfile es un archivo de texto que contiene las instrucciones para construir una imagen Docker. Instruye a Docker para que use una imagen base de PHP con Apache, y copie el archivo PHP al directorio adecuado dentro del contenedor.

```
 GNU nano 6.2                                                    Dockerfile
# Usar una imagen base de PHP
FROM php:7.4-apache

# Copiar la aplicación PHP al directorio del contenedor
COPY . /var/www/html/
```

4. Construir la imagen Docker

Utiliza el comando de Docker para construir una nueva imagen a partir del Dockerfile. Esta imagen contendrá tu aplicación PHP y estará lista para ejecutarse.

```
joamft@joamft-VirtualBox:~/php-app$ sudo usermod -aG docker joamft
joamft@joamft-VirtualBox:~/php-app$ newgrp docker
joamft@joamft-VirtualBox:~/php-app$ docker build -t php-app .
[+] Building 27.8s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 169B
 => [internal] load metadata for docker.io/library/php:7.4-apache
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 238B
 => [1/2] FROM docker.io/library/php:7.4-apache@sha256:c9d7e608f73832673479770d66aacc8100011ec751d1905ff63fae3fe2e0ca6d
 => => resolve docker.io/library/php:7.4-apache@sha256:c9d7e608f73832673479770d66aacc8100011ec751d1905ff63fae3fe2e0ca6d
 => => sha256:18b3497ee7f2099a90b66c23a0bc3d5261b12bab367263e1b40e9b004c39e882 3.04kB / 3.04kB
 => => sha256:20a3732f422b7b28dcf99e8597f093a8c135efca62ff0dc02a2d92d916369413 12.51kB / 12.51kB
 => => sha256:a603fa5e3b4127f210503aaa6189abf6286ee5a73deeaab460f8f33ebc6b64e2 31.41MB / 31.41MB
 => => sha256:c428f1a494230852524a2a5957cc5199c36c8b403305e0e877d580bd0ec9e763 226B / 226B
 => => sha256:c9d7e608f73832673479770d66aacc8100011ec751d1905ff63fae3fe2e0ca6d 1.86kB / 1.86kB
 => => sha256:156740b07ef8a632f9f7bea4e57e4ee5541ade376adf9169351a1265382e39de 91.63MB / 91.63MB
 => => sha256:fb5a4c8af82f00730b7427e47bda7f76cea2e2b9aea421750bc9025aface98d8 270B / 270B
 => => sha256:25f85b498fd5bfc6cce951513219fe480850daba71e6e997741e984d18483971 19.25MB / 19.25MB
 => => extracting sha256:a603fa5e3b4127f210503aaa6189abf6286ee5a73deeaab460f8f33ebc6b64e2
 => => sha256:9b233e420ac7bbca645bb82c213029762acf1742400c076360dc303213c309d5 475B / 475B
 => => sha256:fe42347c4ecfc90333acd9cad13912387eea39d13827a25cfa78727fa5d200e9 514B / 514B
 => => sha256:d14eb2ed1e17ae00f5fcb44b0d562e2867c401c20372829e2cf443fc409342fa 10.76MB / 10.76MB
 => => sha256:66d98f73acb62e86c0c226f9eedcbc7eda305df0c1e171ca5caf81cb8b1c40cb 491B / 491B
 => => sha256:d2c43c5efbc861f83ee6565c7102ca660d6f35e158324fbb042de5017e43afe8 10.20MB / 10.20MB
 => => sha256:ab590b48ea476386dd7b07c34de9eff7cf2103c4668ade985fe31e59f15deee8 2.46kB / 2.46kB
 => => sha256:80692ae2d067c8358112c56490a2a97f69ef395fd8f7662a31498644c9a813ef 246B / 246B
 => => sha256:05e465aaa99a358add4acecdade8f39843089069f31fea0201533d3a09a98c9a 892B / 892B
 => => extracting sha256:c428f1a494230852524a2a5957cc5199c36c8b403305e0e877d580bd0ec9e763
 => => extracting sha256:156740b07ef8a632f9f7bea4e57e4ee5541ade376adf9169351a1265382e39de
 => => extracting sha256:fb5a4c8af82f00730b7427e47bda7f76cea2e2b9aea421750bc9025aface98d8
 => => extracting sha256:25f85b498fd5bfc6cce951513219fe480850daba71e6e997741e984d18483971
 => => extracting sha256:9b233e420ac7bbca645bb82c213029762acf1742400c076360dc303213c309d5
 => => extracting sha256:fe42347c4ecfc90333acd9cad13912387eea39d13827a25cfa78727fa5d200e9
 => => extracting sha256:d14eb2ed1e17ae00f5fcb44b0d562e2867c401c20372829e2cf443fc409342fa
 => => extracting sha256:66d98f73acb62e86c0c226f9eedcbc7eda305df0c1e171ca5caf81cb8b1c40cb
 => => extracting sha256:d2c43c5efbc861f83ee6565c7102ca660d6f35e158324fbb042de5017e43afe8
 => => extracting sha256:ab590b48ea476386dd7b07c34de9eff7cf2103c4668ade985fe31e59f15deee8
 => => extracting sha256:80692ae2d067c8358112c56490a2a97f69ef395fd8f7662a31498644c9a813ef
 => => extracting sha256:05e465aaa99a358add4acecdade8f39843089069f31fea0201533d3a09a98c9a
 => [2/2] COPY . /var/www/html/
 => exporting to image
 => => exporting layers
 => => writing image sha256:5773cefc25ad3e807cb2e6ca416193cbd36892071cfaeb84b924692b0b5fc5bb
 => => naming to docker.io/library/php-app
```

(Como paso adicional le añado permisos a mi usuario de manera que no requiera utilizar sudo cada vez que quiera utilizar docker)

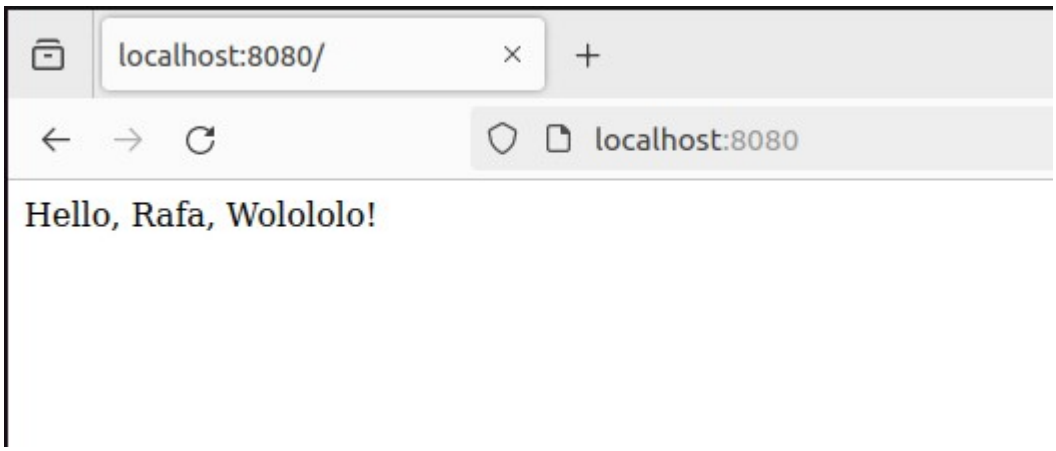
5. Ejecutar un contenedor a partir de la imagen

Ejecuta un contenedor usando la imagen recién creada. Mapea el puerto del contenedor al puerto de tu máquina anfitrión para que puedas acceder a la aplicación PHP a través de un navegador web.

```
joamft@joamft-VirtualBox:~/php-app$ docker run -d -p 8080:80 php-app
8b49897f67e8c749202614b88ec2d98f450b20033e5326662977ff74946e9ab5
```

6. Verificar que la aplicación funcione

Abre un navegador web y navega a la dirección local correspondiente para verificar que la aplicación PHP está funcionando correctamente.



Ejercicio 2

1. Crear un directorio para el proyecto

Similar al primer ejercicio, crea un nuevo directorio para organizar los archivos necesarios para este proyecto, incluyendo el Dockerfile.

```
joamft@joamft-VirtualBox:~$ sudo mkdir ubuntu-apache-php-app
joamft@joamft-VirtualBox:~$ cd ubuntu-apache-php-app/
```

2. Crear un Dockerfile

Escribe un Dockerfile que especifique las instrucciones para construir una imagen basada en Ubuntu. Este archivo debe incluir comandos para instalar Apache y PHP, así como para descargar y copiar una aplicación web PHP en el directorio de Apache.

```
  GNU nano 6.2                                                    Dockerfile
# Usar una imagen base de Ubuntu
FROM ubuntu:20.04

# Configurar la variable de entorno para desactivar la interacción durante la instalación de paquetes
ENV DEBIAN_FRONTEND=noninteractive

# Establecer la zona horaria para evitar la interacción durante la instalación de paquetes relacionados con el tiempo
ENV TZ=America/New_York

# Actualizar la lista de paquetes e instalar Apache y PHP
RUN apt-get update && \
    apt-get install -y apache2 php libapache2-mod-php tzdata

# Descargar la aplicación web PHP y copiarla al directorio de Apache
RUN mkdir -p /var/www/html && \
    echo "<?php phpinfo(); ?>" > /var/www/html/index.php

# Exponer el puerto 80
EXPOSE 80

# Comando para iniciar Apache en el contenedor
CMD ["apachectl", "-D", "FOREGROUND"]
```

3. Construir la imagen Docker

Ejecuta el comando de Docker para construir la imagen a partir del Dockerfile. Esta imagen incluirá un servidor web completo con Apache y PHP, así como la aplicación web PHP descargada.

```
joamft@joamft-VirtualBox:~/ubuntu-apache-php-app$ docker build -t ubuntu-apache-php-app .
[+] Building 106.0s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 799B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/3] FROM docker.io/library/ubuntu:20.04@sha256:874aca52f79ae5f8258faff03e10ce99ae836f6e7d2df6ecd3da5c1cad3a912b
 => [2/3] RUN apt-get update &&     apt-get install -y apache2 php libapache2-mod-php tzdata
 => [3/3] RUN mkdir -p /var/www/html &&     echo "<?php phpinfo(); ?>" > /var/www/html/index.php
 => exporting to image
 => => exporting layers
 => => writing image sha256:2bf68d03bb588d64de0e0c6137a51e18006ba7ccc41143fc911c2a69f2499ec5
 => => naming to docker.io/library/ubuntu-apache-php-app
```

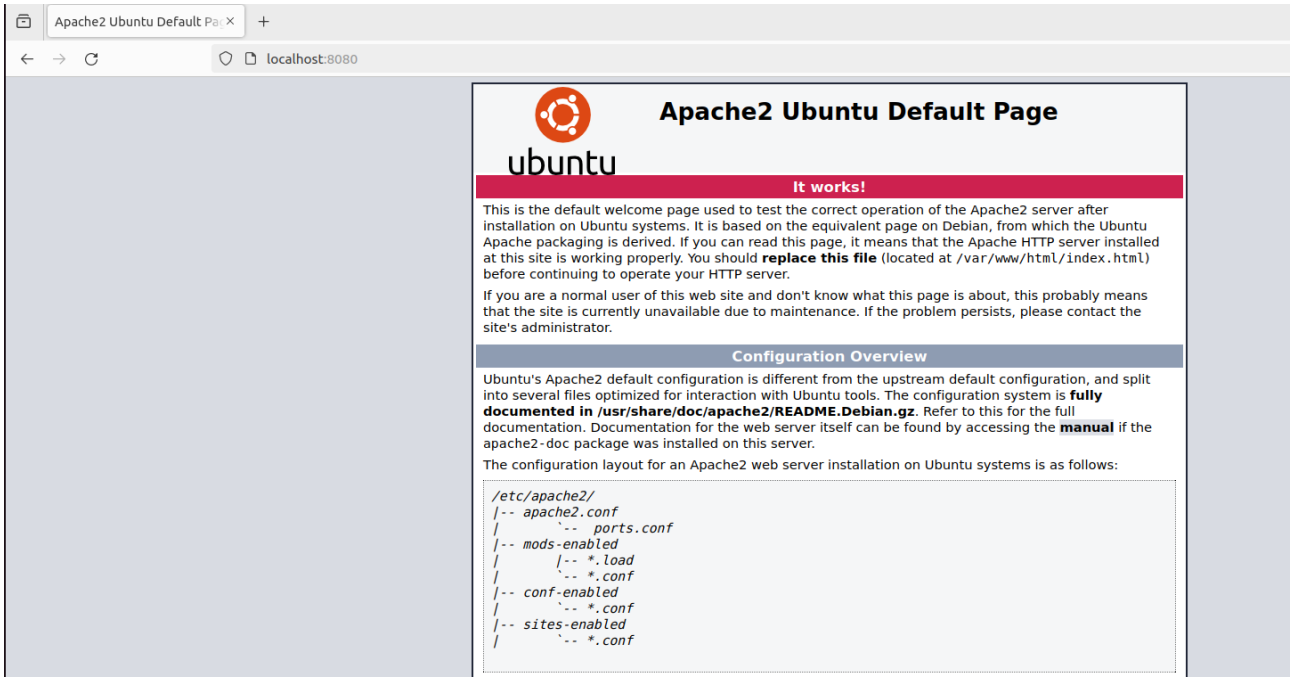
4. Ejecutar un contenedor a partir de la imagen

Inicia un contenedor desde la imagen recién creada, asegurándote de mapear el puerto del contenedor al puerto de tu máquina anfitrión para poder acceder a la aplicación web.

```
joamft@joamft-VirtualBox:~/ubuntu-apache-php-app$ docker run -d -p 8080:80 ubuntu-apache-php-app
9edb4970c7ddeff0c9c4b3bb42f90109238a59452de53c50384123bb06eecf3e
```

5. Verificar que la aplicación funcione

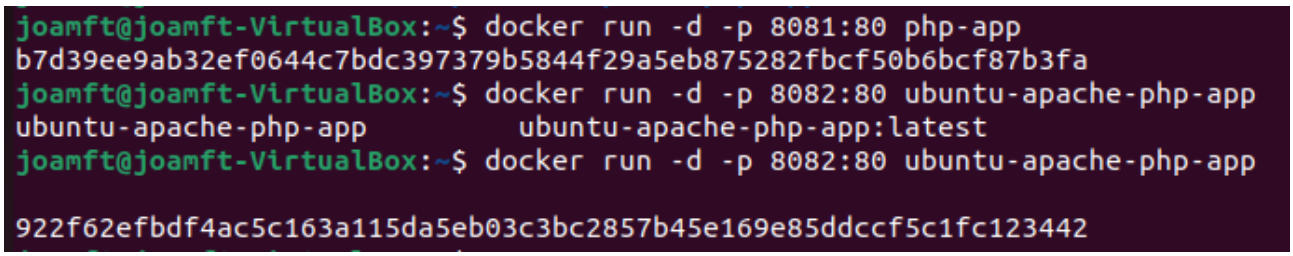
Abre un navegador web y navega a la dirección local correspondiente para asegurarte de que la aplicación web PHP se está sirviendo correctamente.

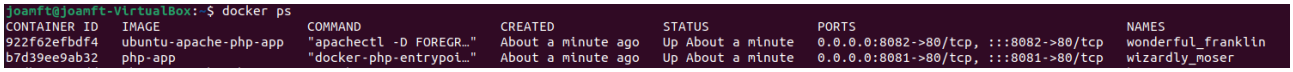


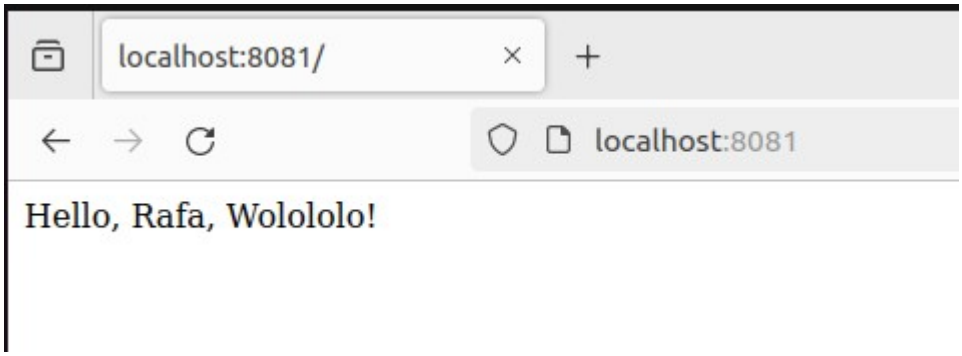
Ejercicio 3

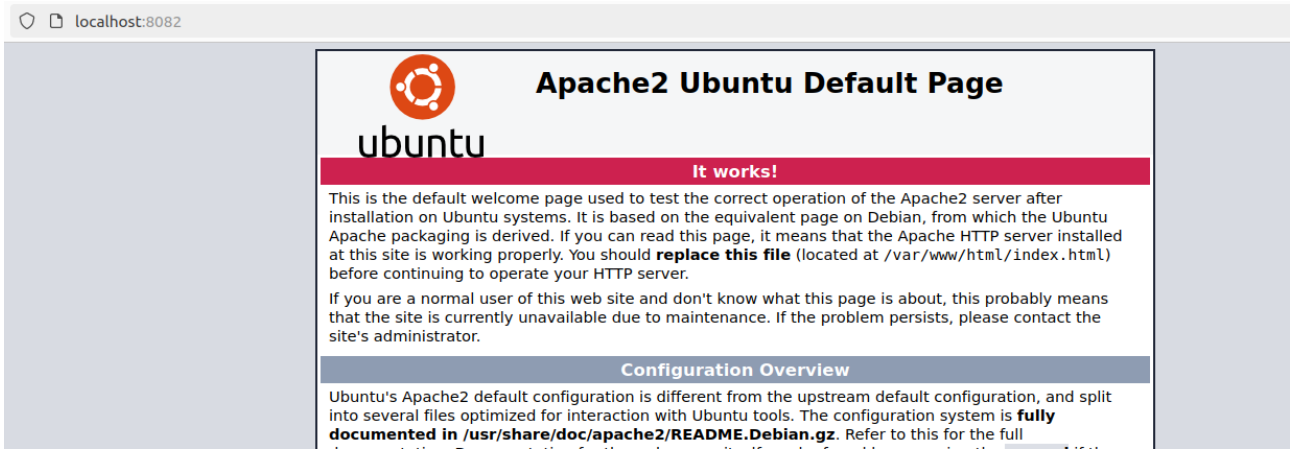
1. Crear un contenedor para cada imagen y verificar su funcionamiento

Inicia contenedores a partir de las imágenes creadas en los ejercicios anteriores y verifica que cada uno funcione correctamente mediante el comando docker ps y yendo a los servicios en el navegador









2. Detener y eliminar los contenedores

Usa comandos de Docker para detener y eliminar los contenedores que has creado. Esto libera recursos en tu sistema anfitrión y mantiene el entorno limpio.

```
joamft@joamft-VirtualBox:~$ docker stop w
wizardly_moser      wonderful_franklin
joamft@joamft-VirtualBox:~$ docker stop w
wizardly_moser      wonderful_franklin
joamft@joamft-VirtualBox:~$ docker stop wizardly_moser
wizardly_moser
joamft@joamft-VirtualBox:~$ docker stop wonderful_franklin
wonderful_franklin
joamft@joamft-VirtualBox:~$ docker rm
bold_easley         boring_moser        wizardly_moser      wonderful_franklin
joamft@joamft-VirtualBox:~$ docker rm
bold_easley         boring_moser        wizardly_moser      wonderful_franklin
joamft@joamft-VirtualBox:~$ docker rm
bold_easley         boring_moser        wizardly_moser      wonderful_franklin
joamft@joamft-VirtualBox:~$ docker rm wizardly_moser
wizardly_moser
joamft@joamft-VirtualBox:~$ docker rm wonderful_franklin
wonderful_franklin
joamft@joamft-VirtualBox:~$
```

(Los nombres son los que crea por defecto docker, se corresponden con los de la captura del docker ps)

Ejercicio 4

1. Lanzar 20 contenedores de la segunda imagen en diferentes puertos

Ejecuta un comando para iniciar 20 instancias de la imagen creada en el segundo ejercicio, cada una mapeada a un puerto diferente en tu máquina anfitrión. Esto demuestra la capacidad de Docker para manejar múltiples contenedores simultáneamente.

```
joamft@joamft-VirtualBox:~$ for i in {1..20}; do docker run -d -p $((8000 + i)):80 ubuntu-apache-php-app; done
f229327372288b053520ee328d8454ee08926066be344ce625fb290bc6b16ecc
fe45977651d9fb282aa8350c81bf55043300c2f1184d048928208b6df54d09bc
5c3b343c61d7cef9f808f90adf1baadb42340a10650fef3af9fd6e23ce5da723
ccf9a0ba954bd7d71d4635e1c1d3486e7bb936c5205e1ffc299b4f2344693c7e
adc8f9da6a67a95edb43e96be4b8045bc792316719634ae0713f5d4643cb0fed
a0707fdb1292c79b3f9f568dee38fde603d5851f2275bd3d69ca8d0f0f634583
a9077b9804da66e05c63c83794575719d96574573ebd6353d8970567bdf29a2b
19141fe5bc1490e80e1e2684cc2b094fa8a918f2708f38cdb58f4e39fe1aca08
8e58e4c99296227f354cbc714a98d9044110f220705fbaa79ede7d985bc21a84
092f99db0af6674203f5fd3496abc7c8260ff4522b675fca1d3c5374400d294c
2e6f06e5aa57503dec9c7a35383ae27c25a01252567b7f458cf2261ac75f175a
8e9e9592828e2979f99459a768c0d859b8575e69b05b1de3ef4b741d1d919646
b9cc53a96a87cfe1d460209604c31aabfb282ccb1711d619c59ea83167b0ce9f
73985b7735476db6b8b8b7a76f1c865fd23b08054c598c1c44691f41f3777202
f3d8836e4ef07d16194a76a22062971f008dcb73964105a5ea7dabfeb8159cff
716c318341e47747fbf28e9cc00d27ffe764c632af696741a9d6c1fcc9242555
671c03c828d9f0bd461546d810ef02e53c81ea707bd5a72fcaf881b233627456
031ded41ca626ccaad4297a7373dd8a7b372eadcd8aa11d51a176d19dbc6ea9f
8725c729e6ad9a33588e3d96e9532d9afe334c6d1551c929f36607e81f409c48
2f5cb64dd0fac80d3b37418d699aa3f0a67497f1c45bcdbb7aebe3e6fa4032f2
joamft@joamft-VirtualBox:~$
```

El comando consiste de un bucle for en Bash que itera desde 1 hasta 20. En cada iteración, la variable i toma el valor actual del contador del bucle (1, 2, 3, ..., 20). La segunda parte del comando ya es docker y consiste en:

docker run: Este comando se utiliza para ejecutar un contenedor nuevo.

-d: La opción -d indica que el contenedor debe ejecutarse en segundo plano (detached mode).

-p $((8000 + i)):80: Esta opción mapea un puerto de la máquina anfitriona a un puerto del contenedor.

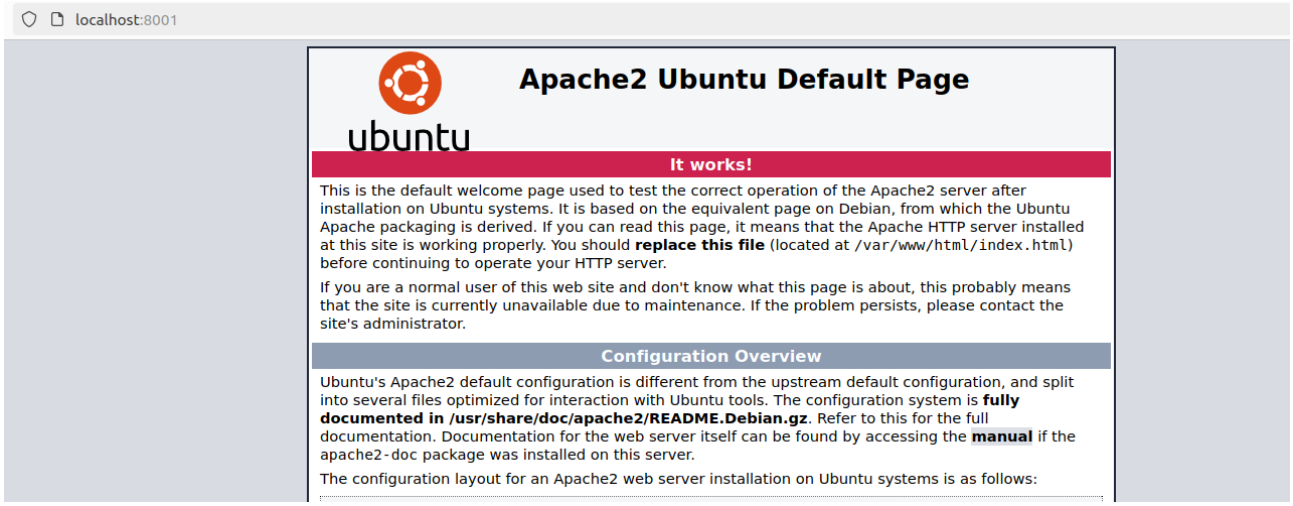
$((8000 + i)): Esta expresión aritmética en Bash calcula el puerto de la máquina anfitriona. Para la primera iteración (i=1), esto se evalúa como 8001, para la segunda iteración (i=2) se evalúa como 8002, y así sucesivamente hasta 8020.

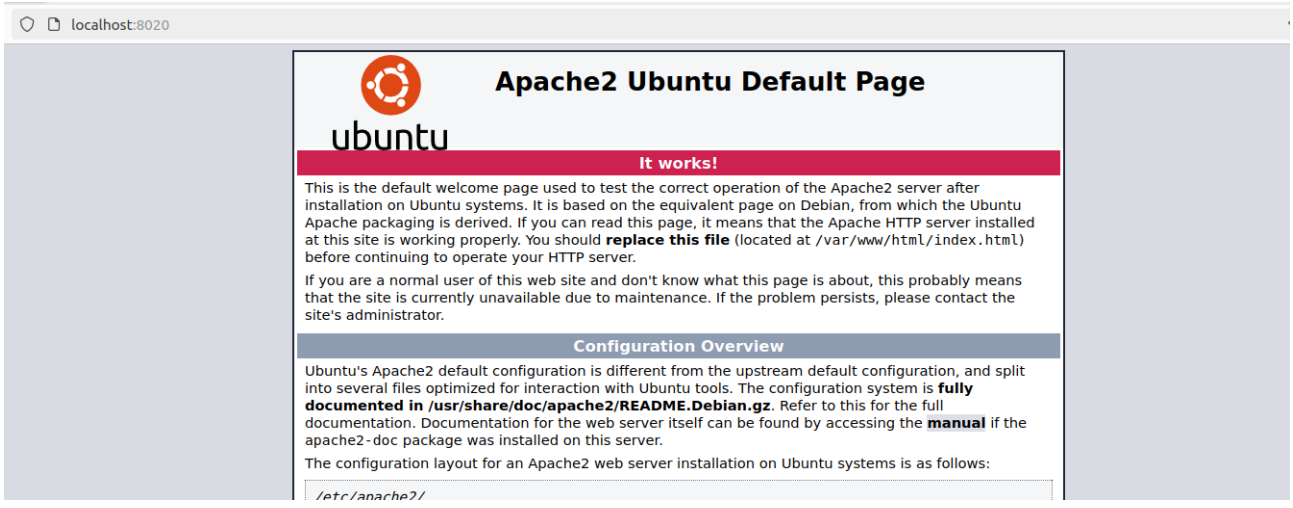
:80: Esto mapea el puerto 80 del contenedor (donde Apache normalmente escucha) al puerto calculado de la máquina anfitriona.

ubuntu-apache-php-app: Este es el nombre de la imagen Docker a partir de la cual se ejecuta el contenedor.

2. Verificar que funcionan

Asegúrate de que todos los contenedores estén funcionando correctamente accediendo a cada uno a través de los puertos mapeados en tu máquina anfitrión.





3. Detener y eliminar los 20 contenedores

Podemos simplificar nuesph tra tarea utilizando filtros de docker que permitan clasificar los contendores según su “padre” que sería la imagen de manera que todos los contenedores creados con la imagen del ejercicio anterior pueden ser parados y eliminados a la vez de la siguiente forma:

```
joamft@joamft-VirtualBox:~$ docker ps -q --filter "ancestor=ubuntu-apache-php-app" | xargs docker stop

2f5cb64dd0fa
8725c729e6ad
031ded41ca62
671c03c828d9
716c318341e4
f3d8836e4ef0
73985b773547
b9cc53a96a87
8e9e9592828e
2e6f06e5aa57
092f99db0af6
8e58e4c99296
19141fe5bc14
a9077b9804da
a0707fdb1292
adc8f9da6a67
ccf9a0ba954b
5c3b343c61d7
fe45977651d9
f22932737228
joamft@joamft-VirtualBox:~$
joamft@joamft-VirtualBox:~$ docker ps -aq --filter "ancestor=ubuntu-apache-php-app" | xargs docker rm
2f5cb64dd0fa
8725c729e6ad
031ded41ca62
671c03c828d9
716c318341e4
f3d8836e4ef0
73985b773547
b9cc53a96a87
8e9e9592828e
2e6f06e5aa57
092f99db0af6
8e58e4c99296
19141fe5bc14
a9077b9804da
a0707fdb1292
adc8f9da6a67
ccf9a0ba954b
5c3b343c61d7
fe45977651d9
f22932737228
9edb4970c7dd
joamft@joamft-VirtualBox:~$
```

Git

No tengo la captura de cada uno por separado pero he realizado una pequeña demostración de como sería la subida de los dos primeros ejercicios ya que considero que son los únicos que puedo subir en forma de repositorio ya que los otros 2 son comandos sueltos que ya están explicados en el documento.

```
joamft@joamft-VirtualBox:~$ git add php-app/
joamft@joamft-VirtualBox:~$ git commit -m "Ejercicio 1:Crear Dockerfile con una imagen PHP"
[main 10ab2c0] Ejercicio 1:Crear Dockerfile con una imagen PHP
 2 files changed, 8 insertions(+)
 create mode 100644 php-app/Dockerfile
 create mode 100644 php-app/index.php
joamft@joamft-VirtualBox:~$ git push
Enumerando objetos: 6, listo.
Contando objetos: 100% (6/6), listo.
Compresión delta usando hasta 2 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (5/5), 554 bytes | 554.00 KiB/s, listo.
Total 5 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/Whiterosiu/PPS-DevSecOps
   a5df5dc..10ab2c0  main -> main
joamft@joamft-VirtualBox:~$ git add ubuntu-apache-php-app/
joamft@joamft-VirtualBox:~$ git commit -m "Ejercicio 2: Crear Dockerfile con Ubuntu, Apache y PHP"
[main f8bac51] Ejercicio 2: Crear Dockerfile con Ubuntu, Apache y PHP
 1 file changed, 22 insertions(+)
 create mode 100644 ubuntu-apache-php-app/Dockerfile
joamft@joamft-VirtualBox:~$ git push
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 2 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (4/4), 856 bytes | 856.00 KiB/s, listo.
Total 4 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/Whiterosiu/PPS-DevSecOps
   10ab2c0..f8bac51  main -> main
joamft@joamft-VirtualBox:~$
```