ACTIVIDADES PRACTICANDO CON DOCKER

YERAY PADIAL

DESPLIEGUE DE APLICACIONES WEB

2ºDAW
I.E.S. LA MARISMA (HUELVA)
PROFESOR: IGNACIO

Índice

Actividad 1

En esta primera actividad vamos a profundizar en como instalar Docker en Ubuntu, para ello utilizaremos la documentación correspondiente de https://docs.docker.com/engine/install/ubuntu/ .

Lo primero que debemos de asegurarnos es de eliminar cualquier paquete o archivo que tengamos, que pueda ocasionar problemas, para ello ponemos este comando en la terminal "for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done" o tambien podemos con este " for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done".

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~$ for pkg in docker.io docker-doc
docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get
remove $pkg; done
[sudo] contraseña para usuario:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
```

Posteriormente haremos un set-up del repositorio de docker con los siguientes comandos:

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~$ # Add Docker's official GPG key:
```

"sudo apt-get update"

```
sudo apt-get update
```

" sudo apt-get install ca-certificates curl"

```
sudo apt-get install ca-certificates curl
```

"sudo install -m 0755 -d /etc/apt/keyrings"

```
sudo install -m 0755 -d /etc/apt/keyrings
```

"sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc"

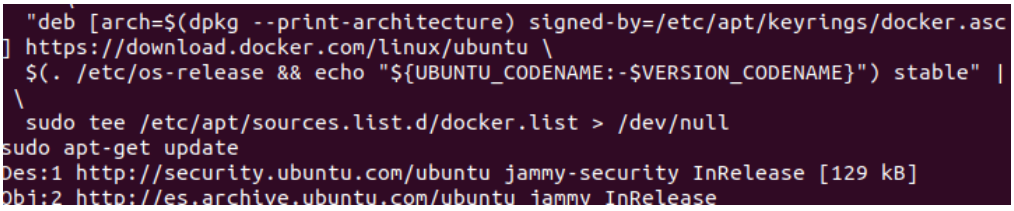
```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyring
s/docker.asc
```

"sudo chmod a+r /etc/apt/keyrings/docker.asc"

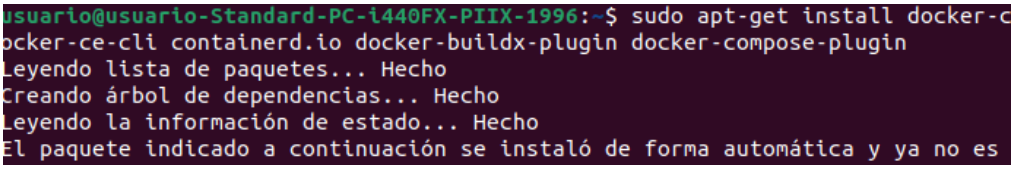
```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Después importante añadir el repositorio al Apt así:

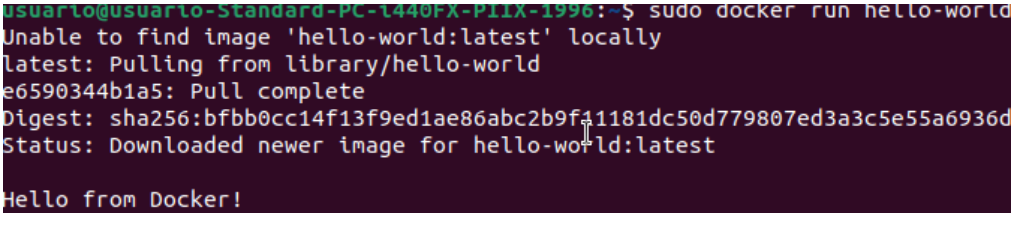
echo \"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \ https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null && sudo apt-get update



Tras hacer estos pasos previos debemos de instalar los paquetes de Docker:
“sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin”



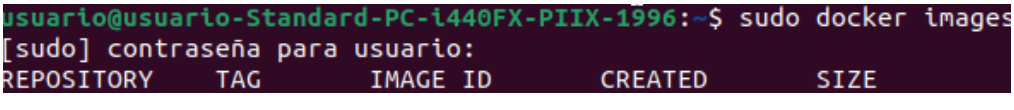
Y asi se confirmamos que todo se instaló correctamente:
“sudo docker run hello-world”



Actividad 2

Práctica primer artículo

En esta segunda actividad,tras comprobar que todo se hizo correctamente y se ejecutó la imagen “hello-world”, mostraremos imágenes Docker instaladas, para ello pondremos el siguiente comando:”sudo docker images”



Nos pide la contraseña, asi que la ponemos y nos muestra todos sus datos sin problemas, “Pd:Haciendo la documentación, me di cuenta que no hice la captura de los detalles”.

Si queremos mostrar todos los contenedores en docker escribiremos : “sudo docker ps -a ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~$ sudo docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS
     PORTS     NAMES
07a2a323ecba   hello-world   "/hello"   22 minutes ago   Exited (0)
go             intelligent_johnson
```

Práctica segundo artículo

Creamos un Dockerfile: “ mkdir mi_proyecto1”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~$ mkdir proyecto1
```

Y lo editamos “nano Dockerfile ”, poniendo algo como esto que nos da de ejemplo “# Indica la imagen base
FROM ubuntu:latest

Actualiza paquetes e instala, por ejemplo, curl
RUN apt-get update && apt-get install -y curl

Mensaje o comando que se ejecutará por defecto
CMD ["echo", "Hola desde mi contenedor!"]”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~$ cd proyecto1
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ nano Dockerfile
```

Después guardamos y listo.

Para crear un contenedor propio ponemos “docker build -t proyecto .”

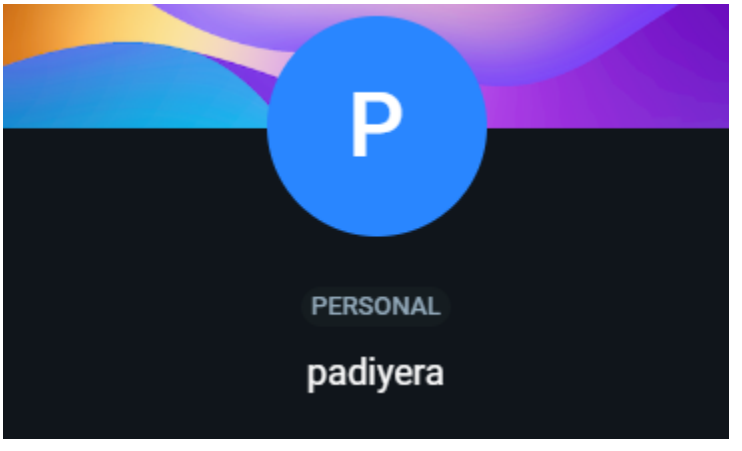
```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker build -t p
royecto .
```

```
=> => transferring dockerfile: 271B                                    0
=> [internal] load metadata for docker.io/library/ubuntu:latest        1
=> [internal] load .dockerignore                                       0
=> => transferring context: 2B                                         0
=> [1/2] FROM docker.io/library/ubuntu:latest@sha256:72297848456d5d37d12  4
=> => resolve docker.io/library/ubuntu:latest@sha256:72297848456d5d37d12  0
=> => sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c 6.69kB / 6.69kB  0
=> => sha256:3afff29dffbc200d202546dc6c4f614edc3b109691e7ab4 424B / 424B  0
=> => sha256:a04dc4851cbcbb42b54d1f52a41f5f9eca6a5fd0374 2.30kB / 2.30kB  0
=> => sha256:5a7813e071bfadf18aaa6ca8318be4824a9b6297b 29.75MB / 29.75MB  1
=> => extracting sha256:5a7813e071bfadf18aaa6ca8318be4824a9b6297b3240f2c  3
=> [2/2] RUN apt-get update && apt-get install -y curl                 17
=> exporting to image                                                  1
=> => exporting layers                                                 1
=> => writing image sha256:2a979f0b519013e451aa0a1405f55eba678d000be8082  0
=> => naming to docker.io/library/proyecto                             0
```

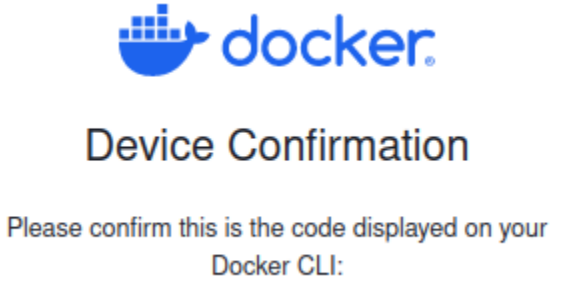
Posteriormente ponemos en funcionamiento con “docker run proyecto”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run proyec
to
Hola desde mi contenedor
```

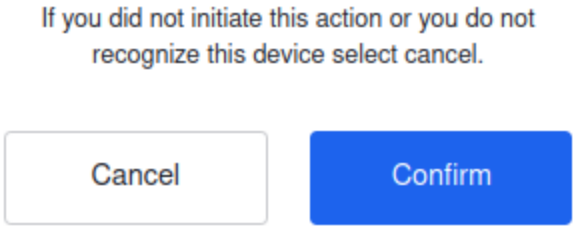
Importante crearse una cuenta de docker y crear el repositorio del proyecto.



Si queremos publicarlo, primero necesitamos poner “docker login”



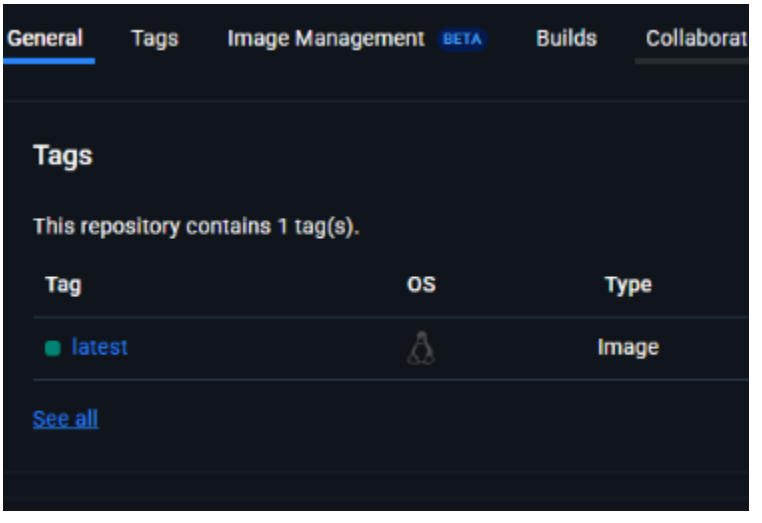
Presionamos confirm:



Ya para terminar tenemos que realizar un push en la terminal, obviamente debemos de tener claro que nuestra imagen debe de tener una etiqueta que es nuestro nombre de usuario.
Para hacer el push de nuestro contenedor haremos asi ”docker push padiyera/proyecto:latest”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker push
35be745f959: Pushed
b7c01ed0534: Mounted from library/ubuntu
atest: digest: sha256:6fe18a76dc32ac2e917381a00414e69beb51b0ac7d2daf23d
ff7c7b size: 741
```

Y en el repositorio se puede ver si salio bien:



Actividad 3

Para esta actividad, seguiremos practicando con Docker y haremos descarga de imágenes, mostraremos el listado de ellas, ejecutaremos contenedores, los mostraremos, editaremos y borraremos sobre ellos.

Comenzamos descargando la imagen de Ubuntu tal que así “docker pull ubuntu”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker pull ubunt
u
Using default tag: latest
latest: Pulling from library/ubuntu
5a7813e071bf: Already exists
Digest: sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
Status: Downloaded newer image for ubuntu:latest
```

Con la imagen de Nginx “docker pull nginx”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
7cf63256a31a: Pull complete
bf9acace214a: Pull complete
513c3649bb14: Pull complete
d014f92d532d: Pull complete
9dd21ad5a4a6: Pull complete
943ea0f0c2e4: Pull complete
103f50cb3e9f: Pull complete
```

y con la ultima “docker pull hello-world ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker pull hello
-world
Using default tag: latest
latest: Pulling from library/hello-world
```

Para ver el listado de ellas sería asi “docker images”.

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker images
```

nginx	latest
ubuntu	latest
hello-world	latest

Y para ejecutar cada contenedor (puede hacerse igual o no) así “docker run --name myhello2 hello-world ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run --name
 myhello2 hello-world

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
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

asi tambien “docker run --name myhello3 hello-world ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run --name
 myhello3 hello-world

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/
```

y con el último “docker run --name myhello1 hello-world” .

```
uario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run --name
yhello1 hello-world

llo from Docker!
is message shows that your installation appears to be working correctly.

 generate this message, Docker took the following steps:
. The Docker client contacted the Docker daemon.
. The Docker daemon pulled the "hello-world" image from the Docker Hub.
  (amd64)
. The Docker daemon created a new container from that image which runs the
  executable that produces the output you are currently reading.
. The Docker daemon streamed that output to the Docker client, which sent it
  to your terminal.

 try something more ambitious, you can run an Ubuntu container with:
docker run -it ubuntu bash
```

Para ver todos los contenedores ponemos “sudo docker -ps a” y vemos todos:

```
usuario@usuario-Standard-PC-i440FX-PIIX-199
CONTAINER ID   IMAGE             COMMAND
S                       PORTS     NAMES
85deba7112a8   hello-world       "/hello"
d (0) 2 minutes ago               myhello3
fc24b34684c0   hello-world       "/hello"
d (0) 2 minutes ago               myhello2
ba2d5a72126c   hello-world       "/hello"
d (0) 3 minutes ago               myhello1
```

Si queremos parar algún contenedor en específico utilizamos “docker stop myhello2” y así con cada contenedor que queramos pausar, tan solo cambiar el nombre correspondiente.

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker stop myhe
lo1
myhello1
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker stop myhe
lo2
myhello2
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker stop myhe
lo3
myhello3
```

De la misma manera se hace para borrarlos pero con este comando “docker rm myhello2 ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker rm myhello
2
myhello2
```

En caso de quererlos mostrar sería asi “sudo docker ps -a”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATU
S                     PORTS     NAMES
30bc5eac4b7a   proyecto:latest   "echo 'Hola desde mi…"   20 minutes ago   Exite
d (0) 20 minutes ago            proyecto
5f1c7f27a8d2   proyecto          "echo 'Hola desde mi…"   38 minutes ago   Exite
d (0) 38 minutes ago            agitated_wescoff
c0f613d8d62f   proyecto          "echo 'Hola desde mi…"   41 minutes ago   Exite
d (0) 41 minutes ago            vigorous_lalande
fe4c698aa3d7   proyecto          "echo 'Hola desde mi…"   41 minutes ago   Exite
d (0) 41 minutes ago            gifted_dirac
fafedd6776ed   proyecto          "echo 'Hola desde mi…"   3 hours ago      Exite
d (0) 3 hours ago               frosty_sutherland
17ab1a40665f   proyecto          "echo 'Hola desde mi…"   3 hours ago      Exite
d (0) 3 hours ago               elegant_cray
a48309279865   proyecto          "echo 'Hola desde mi…"   4 hours ago      Exite
d (0) 4 hours ago               vigorous_golick
07a2a343ecba   hello-world       "/hello"                 4 hours ago      Exite
d (0) 4 hours ago               intelligent_johnson
```

En caso de quererlos borrar todos de una, se hace así “docker rm -f $(docker ps -aq)”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker rm -f $(su
do docker ps -aq)
30bc5eac4b7a
5f1c7f27a8d2
c0f613d8d62f
fe4c698aa3d7
fafedd6776ed
```

Actividad 4

En esta actividad llevaré a cabo el ejemplo 1, 2 y 3 mostrados en el módulo.

Ejemplo 1

Como primer ejemplo tenemos que desplegar la aplicación Guestbook, para empezar nombraremos el contenedor como redis pues que es el valor por defecto para

conectarse con la base de datos. Ambos contenedores deben de estar en la misma red para ello pondremos “docker network create red_guestbook ”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker network cr
eate red_guestbook
18499d78c63dd2ddda19bde2994c1ada9d17784e37b57943e45d97e0dfb48a1f
```

Ejecutaremos los contenedores tal que así:
“docker run -d --name redis --network red_guestbook -v /opt/redis:/data redis redis-server --appendonly yes”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d --n
ame redis --network red_guestbook -v /opt/redis:/data redis redis-server --appen
donly yes
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
7cf63256a31a: Already exists
c8c62be273bb: Pull complete
d6c5e428cfd7: Pull complete
fc690ecf94f9: Pull complete
a37a0a824c7e: Pull complete
40836d0aa8f0: Pull complete
4f4fb700ef54: Pull complete
056ff5d77b71: Pull complete
Digest: sha256:6aafb7f25fc93c4ff74e99cff8e85899f03901bc96e61ba12cd3c39e95503c73
Status: Downloaded newer image for redis:latest
```

y para el otro

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d -p
80:5000 --name guestbook --network red_guestbook iesgn/guestbook
Unable to find image 'iesgn/guestbook:latest' locally
latest: Pulling from iesgn/guestbook
0ecb575e629c: Pull complete
7467d1831b69: Pull complete
feab2c490a3c: Pull complete
f15a0f46f8c3: Pull complete
937782447ff6: Pull complete
e78b7aaaab2c: Pull complete
dfce8611166c: Pull complete
3a6aeb6d9625: Pull complete
2b7e1323c92f: Pull complete
4bf029403d35: Pull complete
5e777db1f033: Pull complete
Digest: sha256:a1fc3e0652e324f643d1ebf8ab028d652aa10e13915bcc2c115d5dc9dd0ba7d2
Status: Downloaded newer image for iesgn/guestbook:latest
```

Comprobación: Ponemos Localhost en el navegador y se verá tal que esto.

Ejemplo 2

Para el ejemplo dos desplegamos la aplicación Temperaturas, es obligatorio poner un nombre adecuado en el dns como puede ser temperaturas_backend.

Dicho esto crearemos una red en la que conectamos los dos contenedores " docker network create red_temperaturas"

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker network cr
eate red_temperaturas
```

Ejecutamos los contenedores "docker run -d --name temperaturas-backend --network red_temperaturas iesgn/temperaturas_backend"

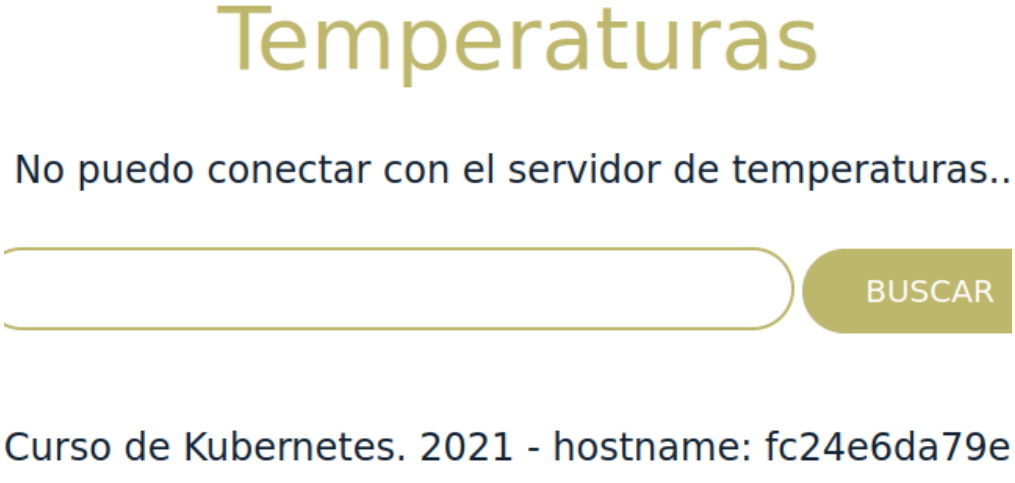
```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d --n
ame tempraturas-backend --network red_temperaturas iesgn/temperaturas_backend
Unable to find image 'iesgn/temperaturas_backend:latest' locally

latest: Pulling from iesgn/temperaturas_backend
32de3c850997: Pull complete
4a604c2354e7: Pull complete
f70ec8148670: Pull complete
7238ddc8ca51: Pull complete
Digest: sha256:ad95f9721e81ac004e258ff0241aa6417c58c5c4f32821e7ff06e92926aa0d3b
Status: Downloaded newer image for iesgn/temperaturas_backend:latest
```

y "docker run -d -p 80:3000 --name temperaturas-frontend --network red_temperaturas iesgn/temperaturas_frontend"

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d -p 80:3000 --name temperaturas-frontend --network red_t
emperaturas iesgn/temperaturas_frontend
```

Resultado desde localhost



Ejemplo 3

Para este último ejemplo, instalaremos Wordpress, necesitamos dos contenedores(base de datos y servidor web) y ambos estarán en la misma red.

Para ello en la terminal ponemos “docker network create red_wp”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker network create red_wp
08e9f97e16535be6552e1d6234672ca34ae6f9cbc05b79738eb6f8ce13a98377
```

Creamos los contenedores necesarios tal que así:

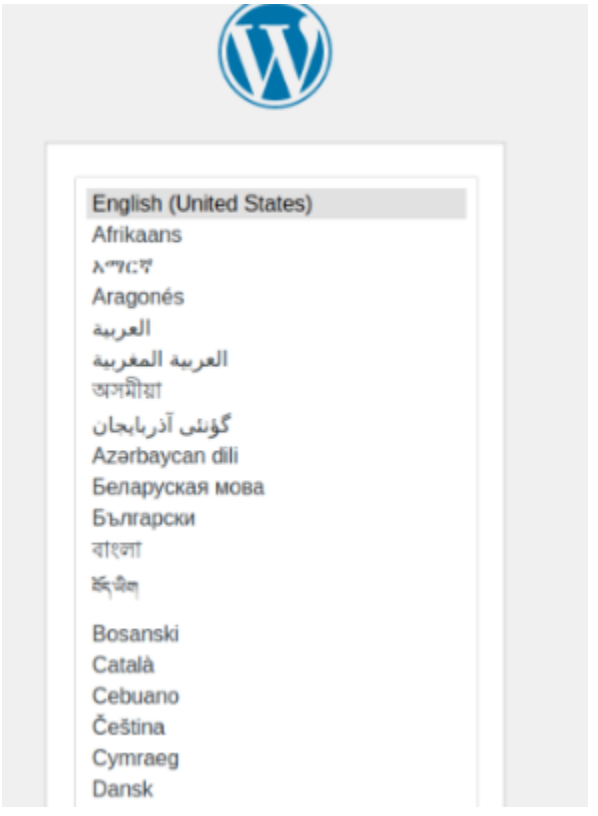
“docker run -d --name servidor_mysql \
--network red_wp \
-v /opt/mysql_wp:/var/lib/mysql \
-e MYSQL_DATABASE=bd_wp \
-e MYSQL_USER=user_wp \
-e MYSQL_PASSWORD=asdasd \
-e MYSQL_ROOT_PASSWORD=asdasd \
mariadb”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d --name servidor_mysql \
                --network red_wp \
                -v /opt/mysql_wp:/var/lib/mysql \
                -e MYSQL_DATABASE=bd_wp \
                -e MYSQL_USER=user_wp \
                -e MYSQL_PASSWORD=asdasd \
                -e MYSQL_ROOT_PASSWORD=asdasd \
                mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
5a7813e071bf: Already exists
bdecd990c29c: Pull complete
5db80086e4da: Pull complete
901fe9394c00: Pull complete
43eb19e1b102: Pull complete
597f7afe50fe: Pull complete
e1dede558384: Pull complete
5c3a22df929b: Pull complete
Digest: sha256:310d29fbb58169dcddb384b0ff138edb081e2773d6e2eceb976b3668089f2f84
Status: Downloaded newer image for mariadb:latest
```

y por otro lado “docker run -d --name servidor_wp \
--network red_wp \
-v /opt/wordpress:/var/www/html/wp-content \
-e WORDPRESS_DB_HOST=servidor_mysql \
-e WORDPRESS_DB_USER=user_wp \
-e WORDPRESS_DB_PASSWORD=asdasd \
-e WORDPRESS_DB_NAME=bd_wp \
-p 80:80 \
wordpress
”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker run -d --name servidor_wp
                --network red_wp \
                -v /opt/wordpress:/var/www/html/wp-content \
                -e WORDPRESS_DB_HOST=servidor_mysql \
                -e WORDPRESS_DB_USER=user_wp \
                -e WORDPRESS_DB_PASSWORD=asdasd \
                -e WORDPRESS_DB_NAME=bd_wp \
                -p 80:80 \
                wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
7cf63256a31a: Already exists
859c077b5003: Pull complete
59e01f001c00: Pull complete
7d7543348a2e: Pull complete
ee6fbc7f6010: Pull complete
7ac282ed1b18: Pull complete
ac27beeba4c1: Pull complete
848a107069e4: Pull complete
a02f50ccc1f1: Pull complete
897474ecb9dc: Pull complete
7b3a864a341f: Pull complete
5b3467e0601d: Pull complete
9e27623ff1e4: Pull complete
4f4fb700ef54: Pull complete
115a3dfab727: Pull complete
4cdbe039bbf5: Pull complete
0af0c10e3a6a: Pull complete
e9ab1aacf4f2: Pull complete
2e0cbc7f9407: Pull complete
9b2cf0bfcbcd: Pull complete
ecf9e78007a4: Pull complete
33569abf5deb: Pull complete
Digest: sha256:c31edd83f61ee9f524ff6a36357bd3bf6bdd4c397c32e15d7ce4708b717569e9
Status: Downloaded newer image for wordpress:latest
```

Y para comprobarlo, nos dirigimos a localhost y vemos lo siguiente:



Actividad 5

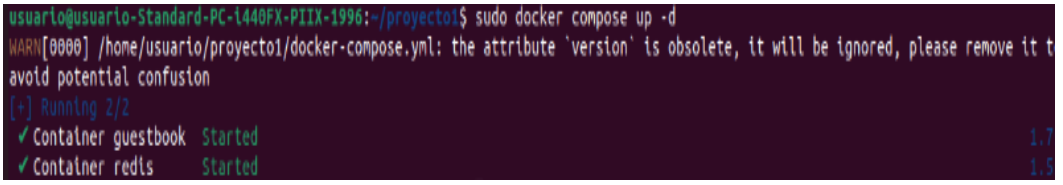
Nuevamente, seleccione 3 ejemplos de los que nos proporciona la documentación.

Ejemplo 1

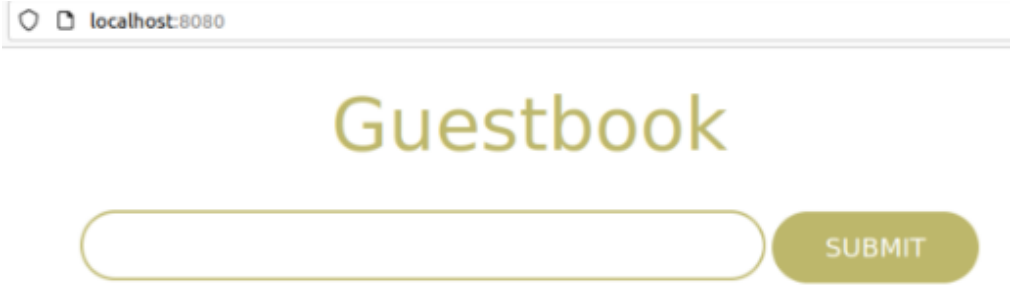
En este primer ejemplo crearemos nuestro docker-compose.yml “ version: '3.1'
services:
 app:
  container_name: guestbook
  image: iesgn/guestbook
  restart: always
  environment:
   REDIS_SERVER: redis
  ports:
   - 8080:5000
 db:
  container_name: redis
  image: redis
  restart: always
  command: redis-server --appendonly yes
  volumes:
   - redis:/data
volumes:
 redis:”

```
version: '3.1'
services:
  app:
    container_name: guestbook
    image: iesgn/guestbook
    restart: always
    environment:
      REDIS_SERVER: redis
    ports:
      - 8080:5000
  db:
    container_name: redis
    image: redis
    restart: always
    command: redis-server --appendonly yes
    volumes:
      - redis:/data
```

Creamos un escenario “docker compose up -d ”



Y vemos el resultado asi en localhost:8080 ya que pusimos ese puerto.



Ejemplo 2

En este ejemplo dos añadiremos a nuestro docker-compose.yml “nano docker-compose.yml

```
version: '3.1'
services:
  frontend:
    container_name: temperaturas-frontend
    image: iesgn/temperaturas_frontend
    restart: always
    ports:
      - 8081:3000
    environment:
      TEMP_SERVER: temperaturas-backend:5000
    depends_on:
      - backend
  backend:
    container_name: temperaturas-backend
    image: iesgn/temperaturas_backend
    restart: always”
```

```
version: '3.1'
services:
  frontend:
    container_name: temperaturas-frontend
    image: iesgn/temperaturas_frontend
    restart: always
    ports:
      - 8081:3000
    environment:
      TEMP_SERVER: temperaturas-backend:500
    depends_on:
      - backend
  backend:
    container_name: temperaturas-backend
    image: iesgn/temperaturas_backend
    restart: always
```

Posteriormente para terminar ponemos “docker compose up -d”

```
uario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker compose up -d
RN[0000] /home/usuario/proyecto1/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it
oid potential confusion
RN[0000] Found orphan containers ([guestbook redis]) for this project. If you removed or renamed this service in your compose
you can run this command with the --remove-orphans flag to clean it up.
] Running 2/2
✔ Container temperaturas-frontend  Started
✔ Container temperaturas-backend   Started
```

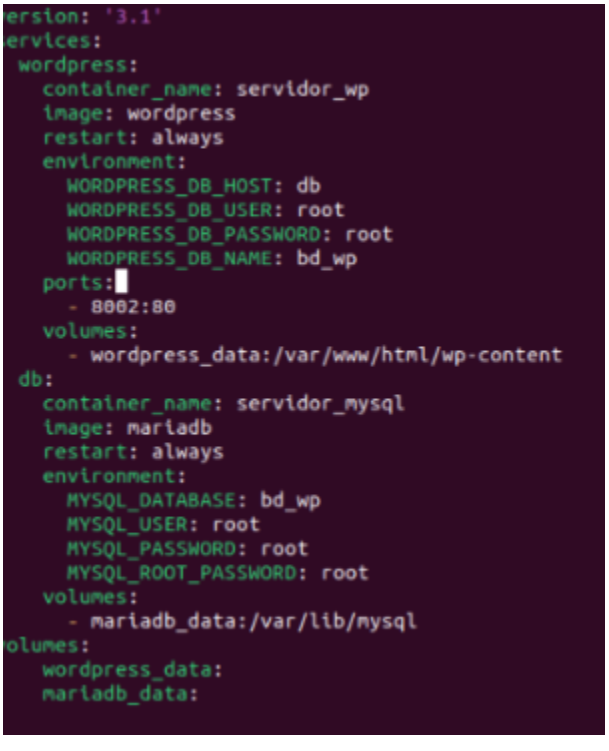
Y el resultado sería el siguiente:

Ejemplo 3

Para este ejemplo al igual que el anterior modificamos con “nano docker-compose.yml

```
version: '3.1'
services:
  wordpress:
    container_name: servidor_wp
    image: wordpress
    restart: always
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: root
      WORDPRESS_DB_PASSWORD: root
      WORDPRESS_DB_NAME: root
```

```
    ports:
      - 8002:80
    volumes:
      - wordpress_data:/var/www/html/wp-content
  db:
    container_name: servidor_mysql
    image: mariadb
    restart: always
    environment:
      MYSQL_DATABASE: bd_wp
      MYSQL_USER: root
      MYSQL_PASSWORD: root
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - mariadb_data:/var/lib/mysql
volumes:
    wordpress_data:
    mariadb_data:”
```



Ponemos “docker compose up -d”

```
usuario@usuario-Standard-PC-i440FX-PIIX-1996:~/proyecto1$ sudo docker compose up -d
WARN[0000] /home/usuario/proyecto1/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to
avoid potential confusion
WARN[0000] Found orphan containers ([temperaturas-frontend temperaturas-backend guestbook]) for this project. If you removed or ren
med this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
[+] Running 3/3
 ✓ Container servidor_wp                Started                                                                          2.0s
 ✓ Container redis                      Removed                                                                          0.5s
 ✓ Container 453d41e1dc6f_servidor_mysql Started                                                                         1.7s
```

Escenario creado.

Y el resultado en local puerto 8002 es

Actividad 6

Al igual que hemos estado realizando las actividades anteriores, seleccionamos 3 ejemplos.

Ejemplo 1

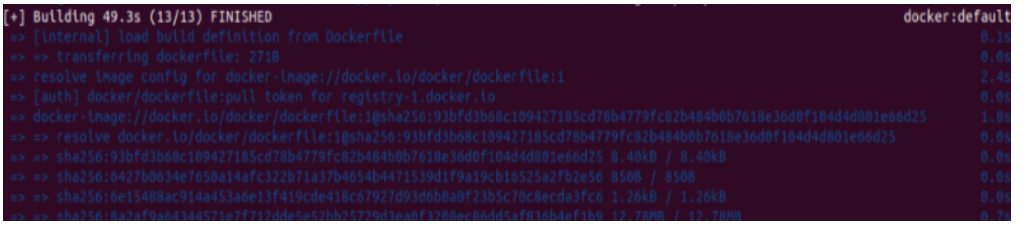
Crearemos imagen con página estática en diferentes versiones.
En Docker tenemos un directorio que se denomina contexto, donde tenemos el fichero Dockerfile y un directorio, llamado public_html. Una vez contextualizado ponemos “ls Dockerfile public_html”

Imagen de un sistema operativo sin ningun servicio. El fichero dockerfile ponemos
“# syntax=docker/dockerfile:1
FROM debian:stable-slim
RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
WORKDIR /var/www/html/
COPY public_html .
EXPOSE 80
CMD apache2ctl -D FOREGROUND”

Instalamos los paquetes necesarios para tener el servidor web, en este acaso apache2. Procedemos a añadir el contenido del directorio public_html al directorio /var/www/html/ del contenedor y finalmente iniciamos el servidor web en segundo plano.

Para ello creamos la imagen “docker build -t padiyera/ejemplo1:v1 .”

Creamos un contenedor “docker run -d -p 80:80 --name ejemplo1 padiyera/ejemplo1:v1”

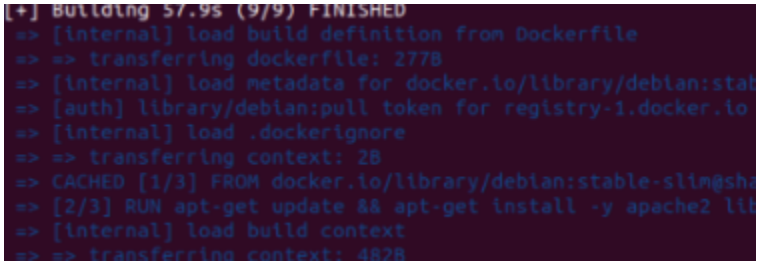


Y ya podemos acceder desde google por ejemplo:

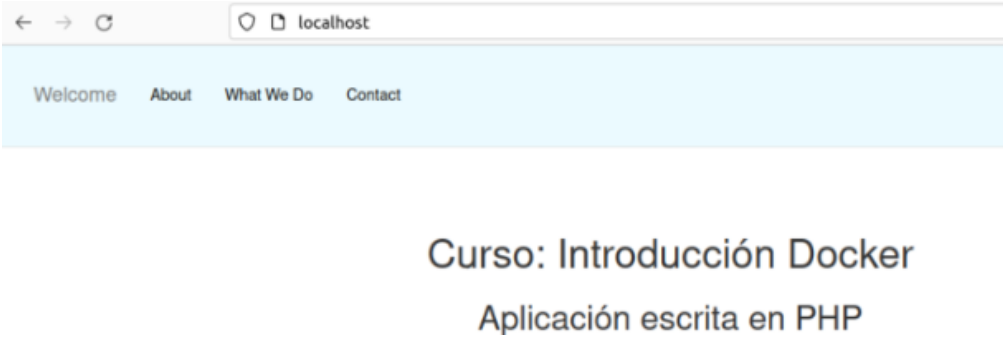
Ejemplo 2

Ya para este ejemplo crearemos la imagen desde una app de php, para ello ponemos en el dockerfile "# syntax=docker/dockerfile:1
FROM debian:stable-slim
RUN apt-get update && apt-get install -y apache2 libapache2-mod-php php && apt-get clean && rm -rf /var/lib/apt/lists/* && rm /var/www/html/index.html
COPY app /var/www/html/
EXPOSE 80
CMD apache2ctl -D FOREGROUND"

Posteriormente de preparar todo, creamos la imagen con "docker build -t padiyera/ejemplo2:v1 ."

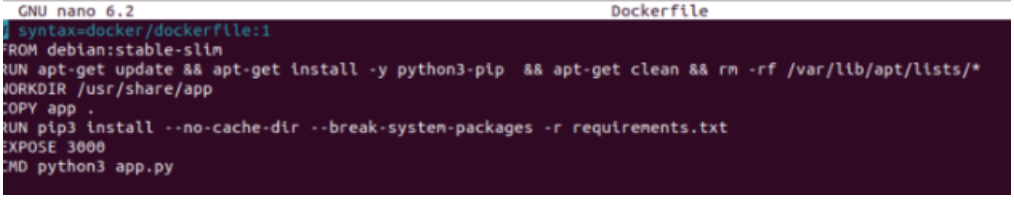


Creo el contenedor y ya despues vemos resultados para comprobar:
"docker run -d -p 80:80 --name ejemplo2padiyera/ejemplo2:v1"



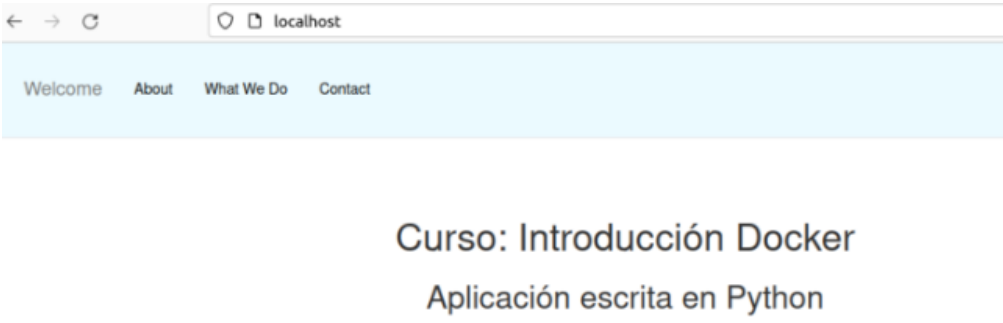
Ejemplo 3

Ahora tendremos el mismo objetivo solo que en vez de a partir de una app de php lo haremos con python, tal que asi:
ponemos “# syntax=docker/dockerfile:1
FROM debian:12
RUN apt-get update && apt-get install -y python3-pip && apt-get clean && rm -rf /var/lib/apt/lists/*
WORKDIR /usr/share/app
COPY app .
RUN pip3 install --no-cache-dir --break-system-packages -r requirements.txt
EXPOSE 3000
CMD python3 app.py” Esto se modifica en el dockerfile.



Creo la imagen con “docker build -t padiyera/ejemplo3:v1 .” Tambien creo el contenedor con “docker run -d -p 80:3000 --name ejemplo2 padiyera/ejemplo3:v1”

Y como resultado:


localhost
Welcome
About
What We Do
Contact
Curso: Introducción Docker
Aplicación escrita en Python