



Documentación de Docker



Arquitectura Web

Profesor Antonio Cardeña

García Ramos, Bernardo

Primavera 2021



ÍNDICE

INTRODUCCIÓN	3
DESARROLLO	4
Teoría	4
¿Qué es Docker?	4
Diferencias entre Contenedores y Máquinas Virtuales	4
Imagen Docker	5
Contenedor Docker	6
Comandos comunes de Docker	6
Práctica	7
Instalación de Docker	7
Ejemplo de implementación con NGINX y VSCode	8
Ejemplo de implementación con NGINX, PHP, HTTPS y HTTP/2 (en base a la práctica).....	18
CONCLUSIONES	20
BIBLIOGRAFÍA/REFERENCIAS	21



INTRODUCCIÓN

El objetivo de este reto y documentación es que quién la lea pueda entender la importancia de los contenedores, sepa cómo implementarla y comprenda qué cosas debería de investigar por su cuenta si quiere saber más de contenedores. De igual manera, con esta documentación se espera que sean capaces de poder implementar Docker con sus funcionalidades básicas y tengan un pequeño manual personalizado sobre el mismo.

Es por lo anterior que en esta documentación se encuentra todo lo necesario para entender los contenedores y empezar a implementarlos con la ayuda de Docker. Se va a ir detalle a detalle de todos los conceptos necesarios para entender lo que es Docker. De igual manera, se va a ilustrar de por qué Docker es una herramienta que nos sirve mucho como desarrolladores de software y/o web.



DESARROLLO

Teoría

¿Qué es Docker?

Docker es una herramienta de creación y administración de contenedores. Básicamente lo que hace es encapsula tu aplicación en desarrollo junto con todas sus dependencias (ya sean bibliotecas, bases de datos, programas auxiliares, etc.) para que funcione en cualquier entorno. Generalmente se usan en el despliegue o hasta en el desarrollo para cuando queremos compartir un entorno estable con otros desarrolladores. De igual forma, los contenedores se usan cuando queremos modificar/mejorar nuestra aplicación a que se adapte a cualquier entorno.

La idea es ejecutar CUALQUIER software en CUALQUIER hardware. Esto significa que tienen el mismo propósito que las máquinas virtuales, sin embargo, los contenedores tienden a tener mejor rendimiento, ya que las máquinas virtuales consumen muchos recursos debido a que tienen que levantar su propio sistema operativo, mientras que los contenedores no.

Diferencias entre Contenedores y Máquinas Virtuales

Como podemos observar en la figura 1, con una máquina virtual se tiene un *hypervisor* (ya sea tipo VirtualBox) que te va a permitir descargar tantos sistemas operativos para tus respectivas aplicaciones. Los contenedores, así como con Docker, van a permitirte administrar los recursos correctamente a tus distintas aplicaciones en desarrollo en base a tu sistema operativo original. En sí lo que terminas teniendo con una máquina virtual son entornos totalmente aislados entre tus aplicaciones para que



no interfieran unas entre otras, mientras que en Docker todos se basa en tu propia máquina.

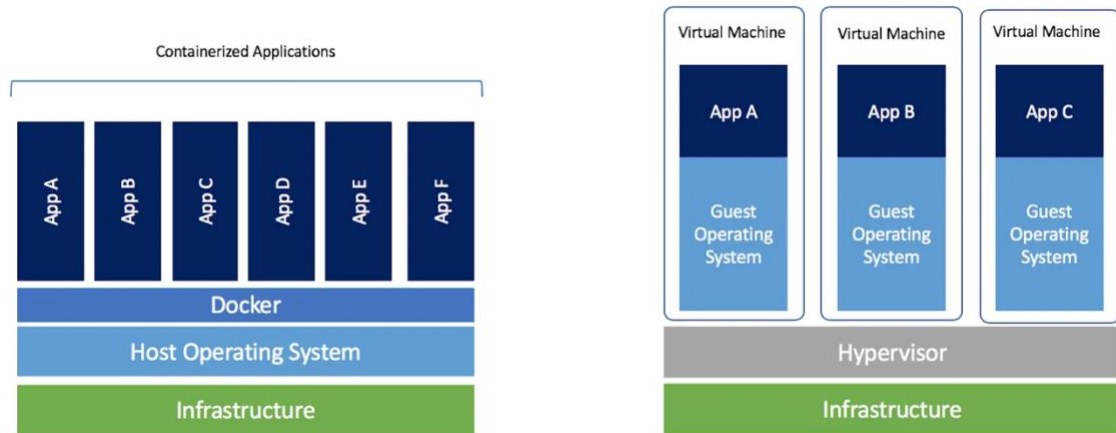


FIGURA 1: Diferencia entre Docker y Máquinas Virtuales respectivamente

Máquina Virtual	Contenedores
Cada MV corre su propio S.O.	Todos los contenedores comparten el mismo S.O.
Rendimiento limitado	Rendimiento nativo
Totalmente aislada = más seguro	Aislamiento a nivel de proceso = menos seguro
Pesado	Ligero

TABLA 1: Ventajas y desventajas clave entre máquinas virtuales y contenedores

Imagen Docker

Es una plantilla para crear un ambiente de desarrollo. Contiene todos los elementos necesarios para tu app, como librerías, código, configuraciones, etc. Estas



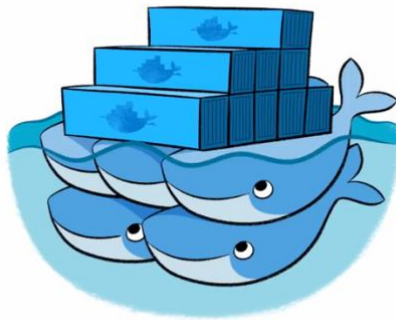
imágenes nos las ofrece Docker, lo cual quiere decir que nosotros no vamos a estar instalando programas, sino que Docker nos va a proveer instaladores de los programas que necesitemos. Con estas imágenes ya no vamos a necesitar descargar MySQL o Python, por ejemplo, sino que con una de estas imágenes vamos a poder trabajar con ellas de manera nativa dentro de nuestros contenedores o ambientes de trabajo.

Para buscar e instalar imágenes, ingresar a la página siguiente:

<https://hub.docker.com/>

Contenedor Docker

Un contenedor es una instancia en ejecución de una imagen, la cual permite un rendimiento nativo de dicho sistema/programa. Es decir, cuando ejecutamos procesos a través de una imagen se les conoce como contenedores.



Comandos comunes en Docker

Para usar los comandos de Docker a través de tu terminal, recuerda siempre escribir *docker* antes.

- ps: Te dice qué contenedores se están ejecutando en este momento.



- `ps -a`: Te dice qué contenedores se han ejecutado y cuando terminaron (si es el caso).
- `images`: Vemos las imágenes instaladas.
- `start [contenedor]`: Podemos volver a ejecutar un contenedor que esté en espera.
- `stop [contenedor]`: Podemos detener un contenedor que esté en ejecución.
- `rm [contenedor]`: Podemos borrar contenedores.
- `run [imagen]`: ejecuta imágenes descargadas para tener contenedores activos
- `run -p [puerto local]:[puerto representativo] -d [imagen]`: el `-p` te permite tener control de tus puertos a la hora de usar tus contenedores. Por otro lado, `-d` te permite tener activo el contenedor en un segundo plano.

Práctica

Instalación de Docker

Lo primero que vamos a necesitar es instalar Docker. Existen diferentes opciones de descarga dependiendo el sistema operativo que manejes, por lo que recomendamos visitar la siguiente página para descargarlo sin más problemas: <https://docs.docker.com/get-docker/>

Si existe la versión de Desktop para tu sistema operativo de preferencia, recomendamos que lo descargues, ya que automáticamente te va a instalar Docker y vas a tener herramientas visuales que te van a poder ayudar (las cuales no se van a ver en esta documentación).

Una vez descargado, en tu terminal escribir el comando *docker*. Si la instalación fue correcta, te desplegará los comandos y configuraciones que puedes hacer. De igual forma, si escribes *docker versión* y le das enter se espera que te aparezca la captura de pantalla 1:



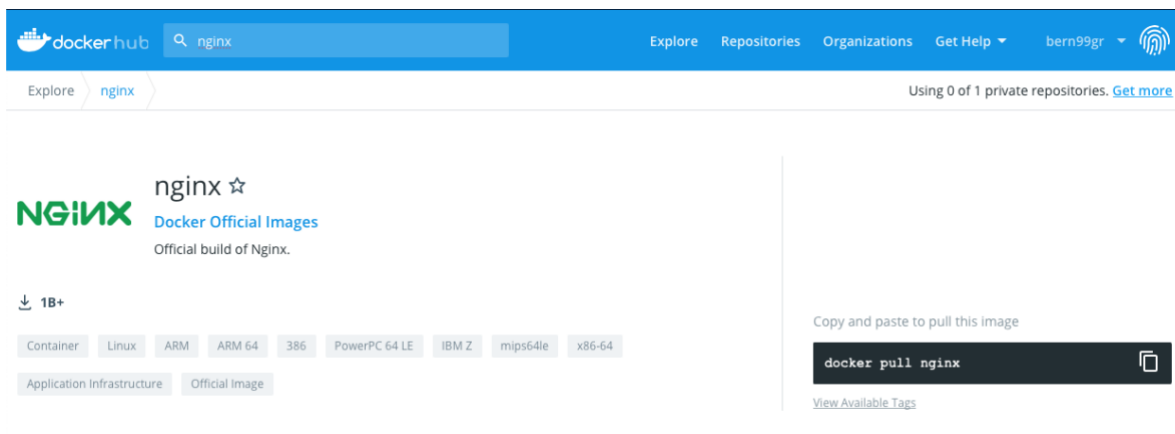
```
((base) MacBook-Pro-de-Mac:~ bernardo$ docker version
Client: Docker Engine - Community
Cloud integration: 1.0.7
Version: 20.10.2
API version: 1.41
Go version: go1.13.15
Git commit: 2291f61
Built: Mon Dec 28 16:12:42 2020
OS/Arch: darwin/amd64
Context: default
Experimental: true

Server: Docker Engine - Community
Engine:
Version: 20.10.2
API version: 1.41 (minimum version 1.12)
Go version: go1.13.15
Git commit: 8891c58
Built: Mon Dec 28 16:15:28 2020
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.4.3
GitCommit: 269548fa27e0089a8b8278fc4fc781d7f65a939b
runc:
Version: 1.0.0-rc92
GitCommit: ff819c7e9184c13b7c2607fe6c30ae19403a7aff
docker-init:
Version: 0.19.0
GitCommit: de40ad0
(base) MacBook-Pro-de-Mac:~ bernardo$
```

CAPTURA DE PANTALLA: Resultado de *docker versión*

Ejemplo de implementación con NGINX y VSCode

1. Lo primero que vamos a necesitar es descargar la imagen de NGINX de Docker. Para eso visitamos la página https://hub.docker.com/_/nginx y búscanos “nginx”.



2. Para instalar la imagen Docker, copiar el comando que se encuentra a la derecha y pegarlo en tu terminal.



```
((base) MacBook-Pro-de-Mac:~ bernardo$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:10b8cc432d56da8b61b070f4c7d2543a9ed17c2b23010b43af434fd40e2ca4aa
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
(base) MacBook-Pro-de-Mac:~ bernardo$
```

3. Para comprobar que lo descargaste correctamente, ingresar el comando *docker images* para ver todas tus imágenes Docker descargadas.

```
((base) MacBook-Pro-de-Mac:~ bernardo$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest   f6d0b4767a6c   3 weeks ago   133MB
(base) MacBook-Pro-de-Mac:~ bernardo$
```

4. Para ahora ejecutar esa imagen, correr el siguiente comando:

```
((base) MacBook-Pro-de-Mac:~ bernardo$ docker run -d -p 3000:80 --name server nginx
efa2fd275c88305cb87907afa6f477292962e8d5d969d0a98640007a837ddd37
(base) MacBook-Pro-de-Mac:~ bernardo$
```

Lo que hace run es ejecutar nuestra imagen, *-d* hace que podamos ejecutarla en segundo plano y no ocupe la presencia de nuestra terminal, *-p [puerto local]:[puerto que requiere]* nos sirve para cambiarle un puerto local del default, *--name [nombre]* nos permite darle nombre a nuestro contenedor y finalmente *nginx* es el nombre de la imagen a ejecutar.

Como podemos observar, este comando nos regresa una cadena de caracteres muy compleja. Esta termina siendo el id de nuestro contenedor.

5. Para revisar que nuestro contenedor se está ejecutando, escribir el comando *docker*

ps:

```
((base) MacBook-Pro-de-Mac:~ bernardo$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS               NAMES
efa2fd275c88   nginx    "/docker-entrypoint..." 5 minutes ago Up 5 minutes    0.0.0.0:3000->80/tcp   server
(base) MacBook-Pro-de-Mac:~ bernardo$
```

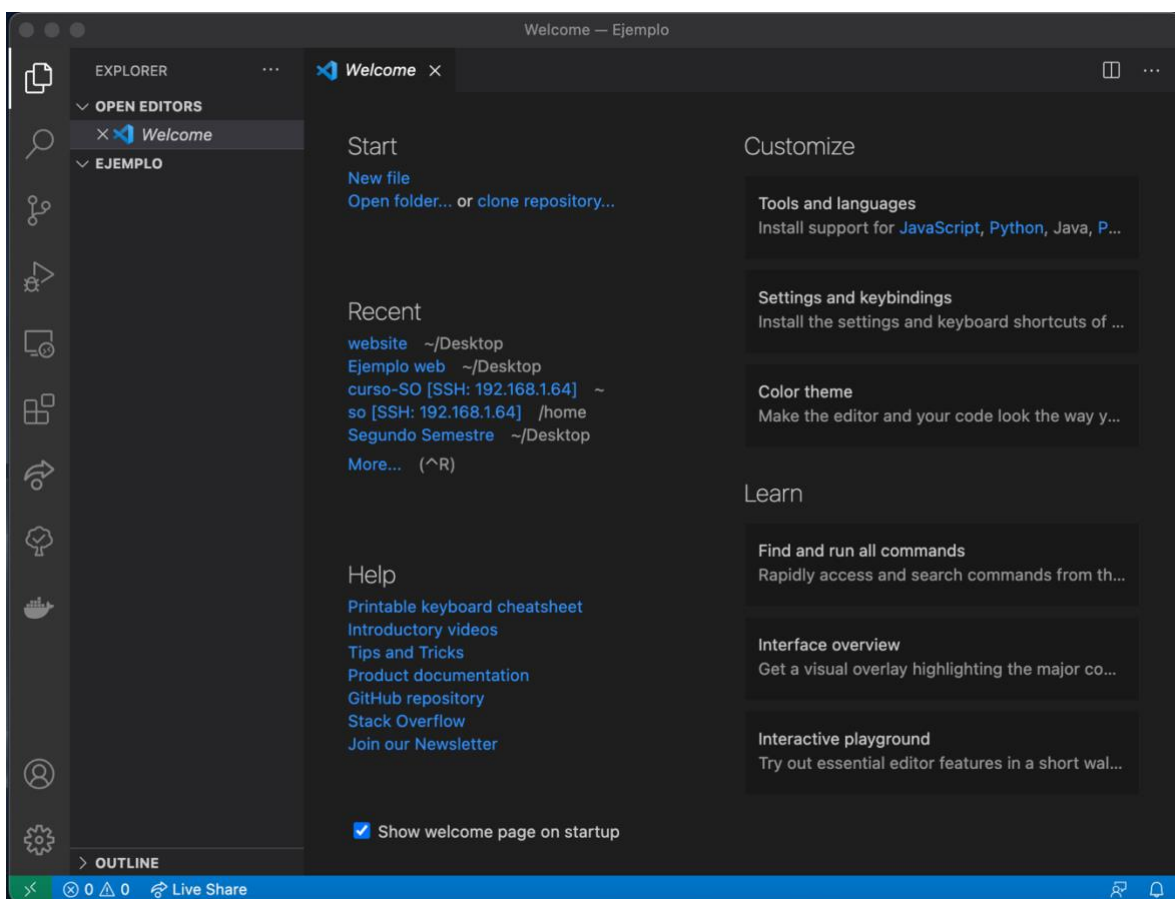


6. De igual manera, para ver todos los contenedores que estén activos o detenidos, ingresar el mismo comando con la extensión -a:

```
(base) MacBook-Pro-de-Mac:~ bernardo$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
efa2fd275c88   nginx    "/docker-entrypoint..." 6 minutes ago Up 6 minutes    0.0.0.0:3000->80/tcp    server
(base) MacBook-Pro-de-Mac:~ bernardo$
```

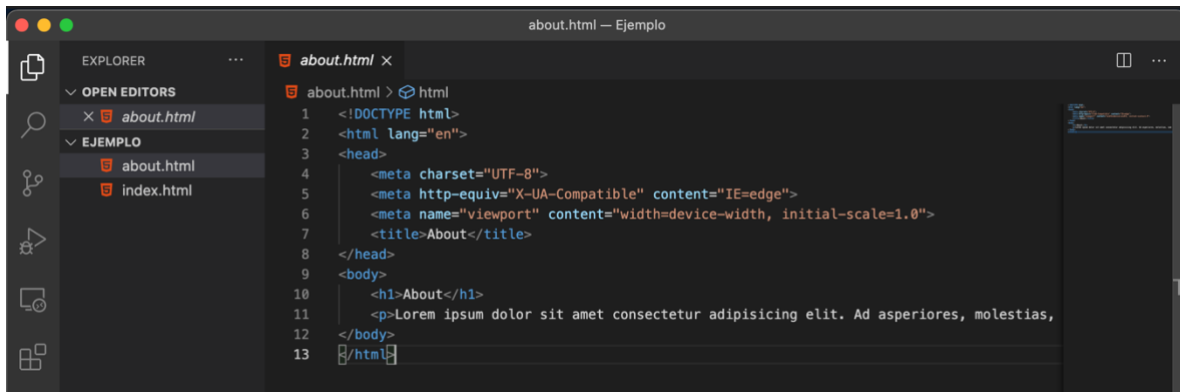
Entonces, ya tenemos NGINX corriendo en nuestro sistema. ¿Pero cómo le hacemos para ponerlo a prueba? Para eso utilizaremos VSCode.

7. Crear una carpeta donde quieras guardar tus archivos (tipo html o php) ya abrirla en VSCode.





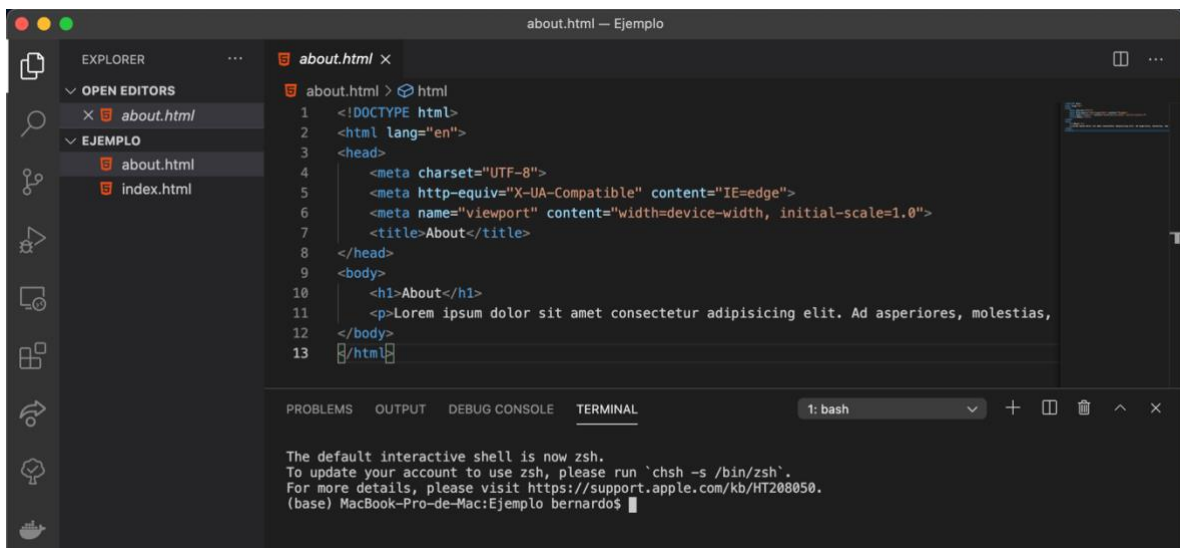
8. Crear un archivo `index.html` y otro `about.html`. Llénalos con la información que quieras.



The screenshot shows the VS Code editor with the file `about.html` open. The Explorer sidebar on the left shows the project structure with `about.html` and `index.html` under the `EJEMPLO` folder. The editor window displays the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>About</title>
8 </head>
9 <body>
10  <h1>About</h1>
11  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Ad asperiores, molestias,
12 </body>
13 </html>
```

9. Abrir una terminal integrada en VSCode con `ctrl+shift+p`.



The screenshot shows the VS Code editor with the file `about.html` open. The Explorer sidebar on the left shows the project structure with `about.html` and `index.html` under the `EJEMPLO` folder. The editor window displays the same HTML code as in the previous screenshot. At the bottom, the integrated terminal is open, showing the following text:

```
1: bash
The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) MacBook-Pro-de-Mac-Ejemplo bernardo$
```

10. Para agregarle la dirección de esta carpeta al servidor de NGINX, vamos a tener que crear un nuevo contenedor (no se pueden modificar los contenedores creados). Para ello, vamos a ingresar el siguiente comando:



```
about.html X
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>About</title>
8 </head>
9 <body>
10  <h1>About</h1>
11  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Ad asperiores, molestias, nam nostrum
12 </body>
13 </html>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$ docker run -d -p 7000:80 -v $(pwd):/usr/share/nginx/html --name website nginx
32d1ee83efc067bfa47852acc70e95fe955028753f27dd6a8a625b3bbf10bde8
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$
```

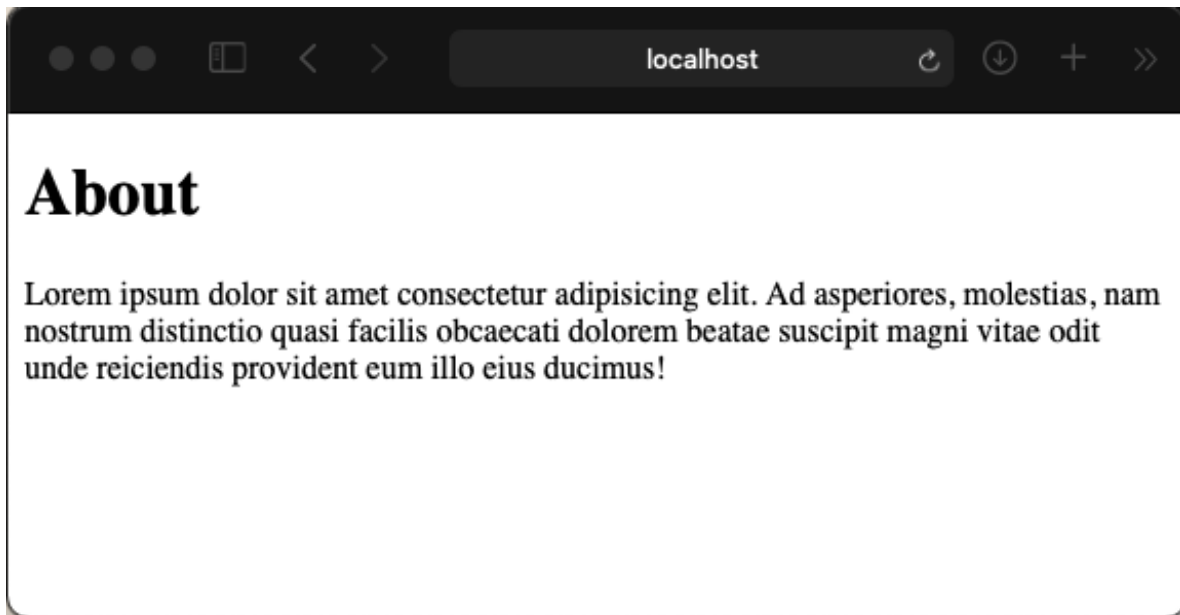
Nota: Hice un cambio en el puerto porque no podíamos repetir el mismo ya que actualmente tenemos otro contenedor corriendo en el puerto 3000. De igual manera, le pusimos otro nombre para distinguirlo del anterior. El argumento que le agregamos fue `-v [dirección original]:[dirección destino]`, así como lo especifica la documentación de NGINX. Recordemos que `$(pwd)` nos regresa el directorio actual.

11. Si en uno de nuestros navegadores buscamos `localhost:7000`, vamos a encontrar nuestra página html creada:

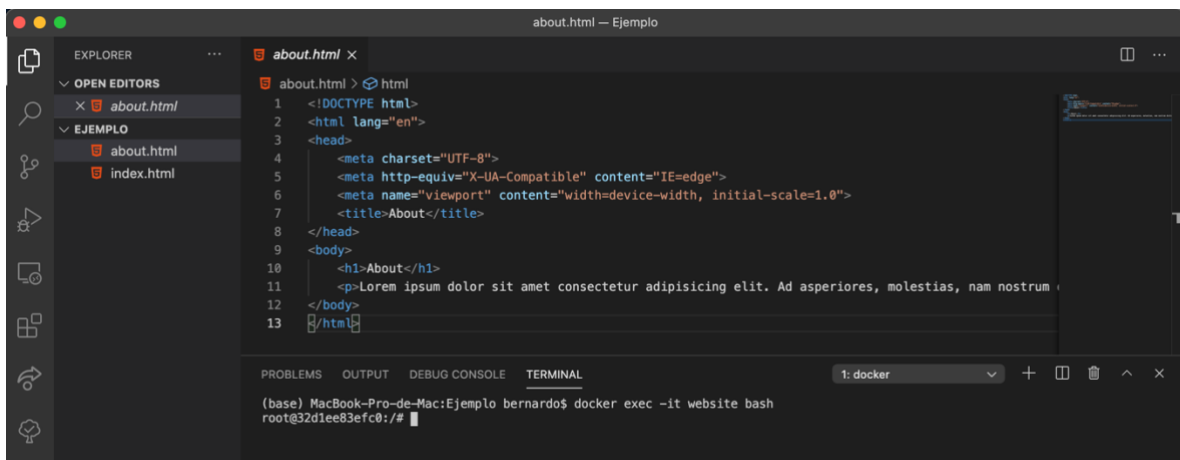


Hola mundo

12. De igual manera, si buscamos `localhost:7000/about.html`, vamos a encontrar nuestra otra página:



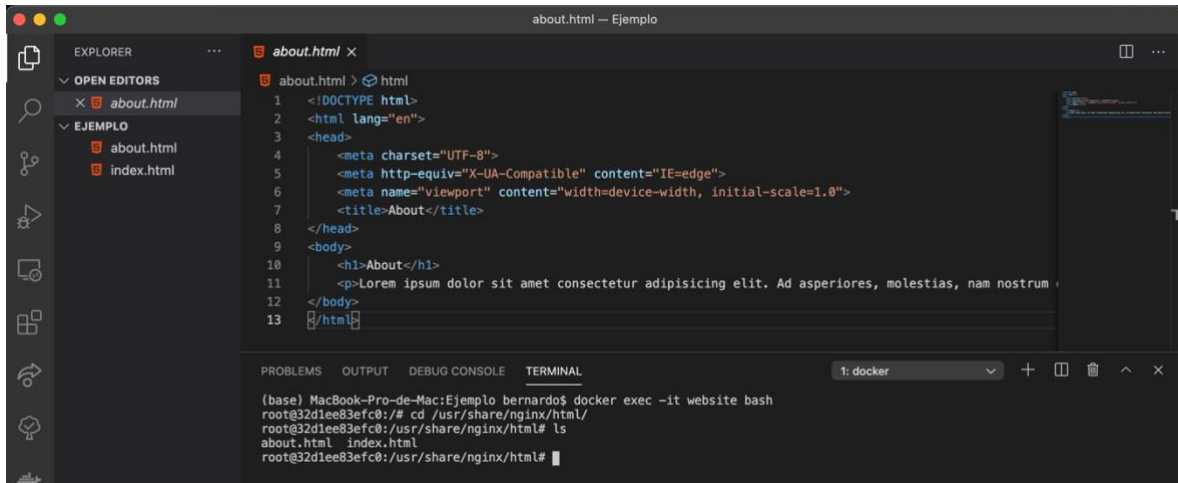
13. Ahora, si queremos ingresar a nuestro servidor y ver lo que tiene adentro, podemos ingresar el siguiente comando:



Como podemos observar, esto nos va a abrir un acceso directo a NGINX, de tal forma que podemos navegar a través del servidor, para buscar nuestros archivos que fueron copiados.



14. Para navegar por NGINX y ver el contenido que fue copiado, ingresar los siguientes dos comandos:



The screenshot shows a VS Code editor with a Docker container terminal at the bottom. The terminal shows the following commands and output:

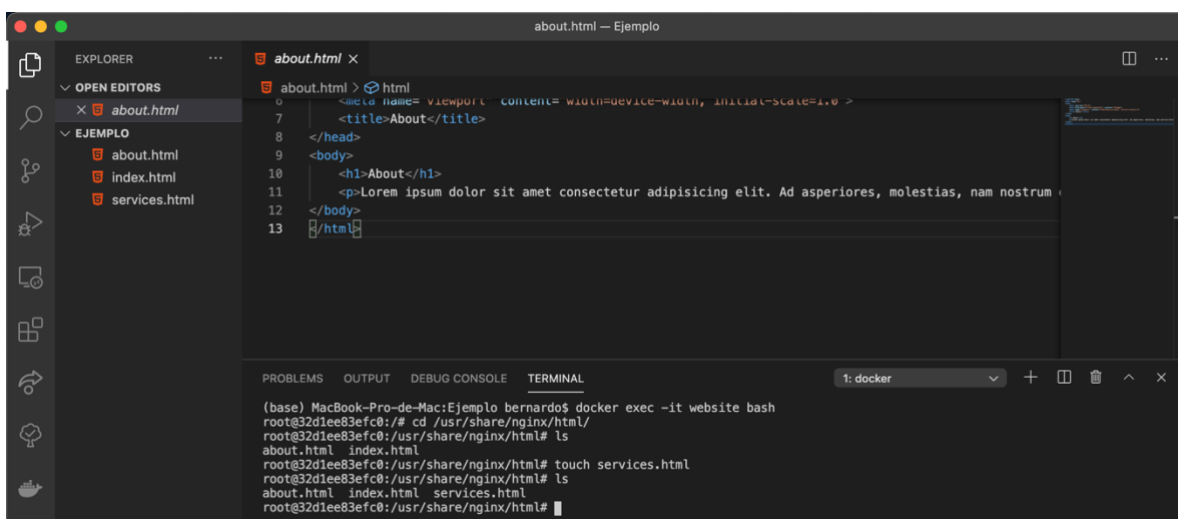
```
(base) MacBook-Pro-de-Mac:Ejemplo bernardos$ docker exec -it website bash
root@32d1ee83efc0:/# cd /usr/share/nginx/html/
root@32d1ee83efc0:/usr/share/nginx/html# ls
about.html index.html
root@32d1ee83efc0:/usr/share/nginx/html#
```

The editor shows the content of `about.html`:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>About</title>
8 </head>
9 <body>
10   <h1>About</h1>
11   <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Ad asperiores, molestias, nam nostrum
12 </body>
13 </html>
```

Con esto podemos comprobar que nuestros archivos efectivamente fueron copiados al servidor.

15. Si quisiéramos agregar un archivo nuevo desde aquí, podríamos simplemente crearlo con el comando *touch* y se vería reflejado también en nuestra carpeta original:



The screenshot shows a VS Code editor with a Docker container terminal at the bottom. The terminal shows the following commands and output:

```
(base) MacBook-Pro-de-Mac:Ejemplo bernardos$ docker exec -it website bash
root@32d1ee83efc0:/# cd /usr/share/nginx/html/
root@32d1ee83efc0:/usr/share/nginx/html# ls
about.html index.html
root@32d1ee83efc0:/usr/share/nginx/html# touch services.html
root@32d1ee83efc0:/usr/share/nginx/html# ls
about.html index.html services.html
root@32d1ee83efc0:/usr/share/nginx/html#
```

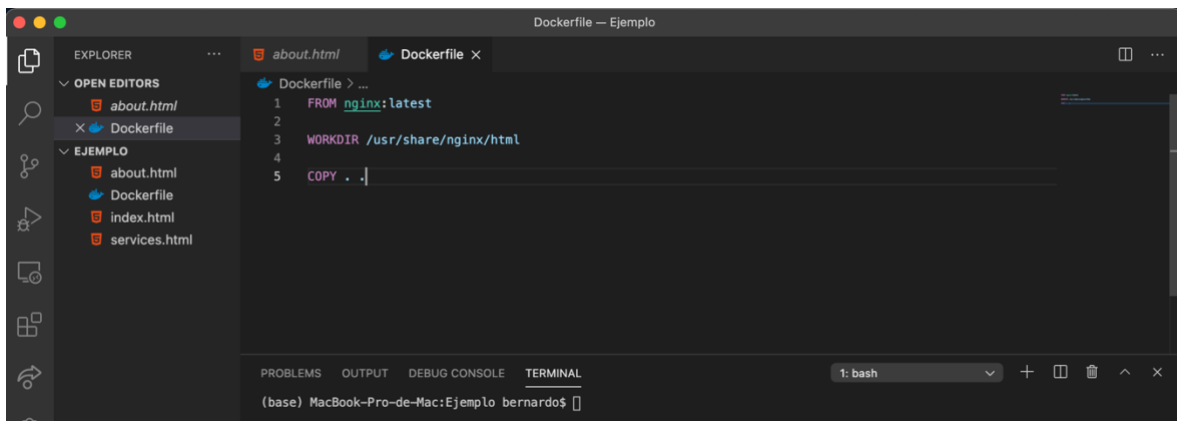
The editor shows the content of `about.html` and a new file `services.html` has been added to the Explorer view.



Esto también lo podríamos ver reflejado si buscáramos en el navegador *localhost:7000/services.html*. Para salir, escribir *exit* y darle enter.

16. Si quisiéramos crear una imagen a partir de lo que acabamos de crear, lo primero que vamos a tener que hacer es ingresar a tu cuenta de Docker Hub desde tu terminal con *docker login*. Con esto puede que te pida tu usuario y contraseña.

17. Lo siguiente sería crear un archivo tipo Dockerfile e ingresar los siguientes datos dentro del archivo:



```
Dockerfile — Ejemplo
1 FROM nginx:latest
2
3 WORKDIR /usr/share/nginx/html
4
5 COPY . .|
```

FROM nos dice qué imagen requiere. WORKDIR es la dirección en donde estamos trabajando, y COPY . . dice que estamos copiando este mismo directorio al directorio especificado en WORKDIR.

18. Ingresar el siguiente comando para crear la imagen de manera local:



The screenshot shows a VS Code editor with a file explorer on the left containing 'about.html', 'Dockerfile', 'index.html', and 'services.html'. The main editor displays a Dockerfile with the following content:

```
1 FROM nginx:latest
2
3 WORKDIR /usr/share/nginx/html
4
5 COPY . .
```

The terminal at the bottom shows the output of the command `docker build -t bern99gr/prueba .`:

```
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$ docker build -t bern99gr/prueba .
[+] Building 0.2s (8/8) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 36B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/nginx:latest                 0.0s
=> [1/3] FROM docker.io/library/nginx:latest                                  0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 121B                                               0.0s
=> CACHED [2/3] WORKDIR /usr/share/nginx/html                                  0.0s
=> CACHED [3/3] COPY . .                                                        0.0s
=> => exporting layers                                                         0.0s
=> => writing image sha256:63342792c6264edae15b481861834c8f0288ec6ab3f43cd9e2dc8cf877949d14 0.0s
=> => naming to docker.io/bern99gr/prueba                                     0.0s
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$
```

-t [pruebas] nos dice que le damos nombre, mientras que el . dice que la imagen va a ser una copia de este mismo directorio. El nombre tiene que tener el nombre de usuario de Docker Hub previo a una diagonal para que funcione correctamente.

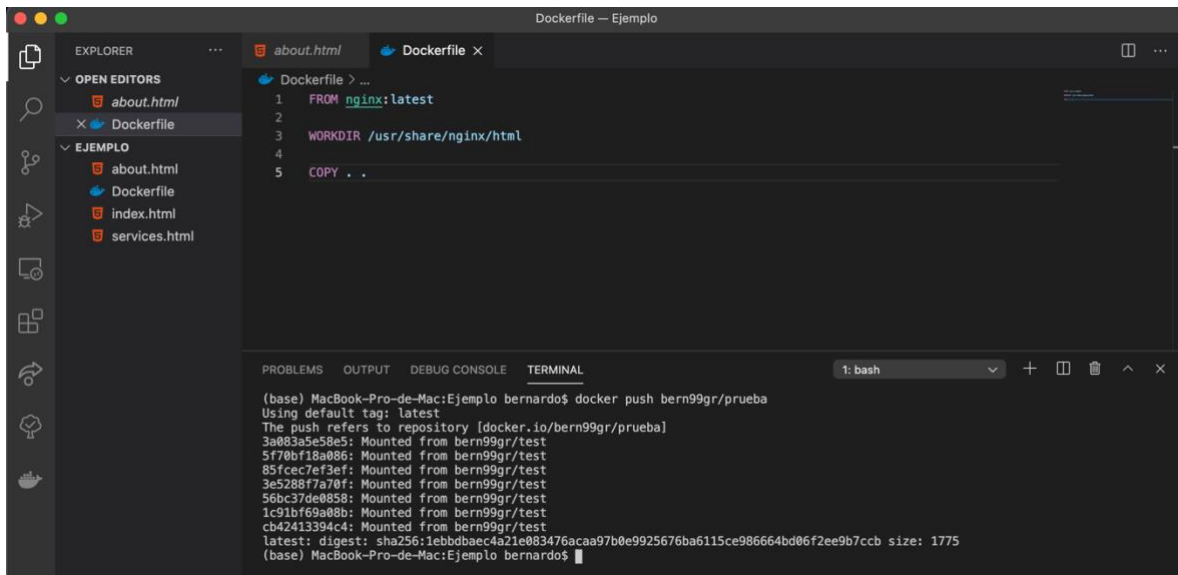
19. Si ingresamos el comando `docker images`, deberíamos de ser capaces de ver nuestra nueva imagen creada.

The screenshot shows the same VS Code editor as before, but the terminal now displays the output of the command `docker images`:

```
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
bern99gr/prueba     latest     63342792c626  3 hours ago  133MB
nginx               latest     f6d8b4767a6c  3 weeks ago  133MB
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$
```




20. Finalmente, para publicarlo en Docker Hub ingresar el siguiente comando:



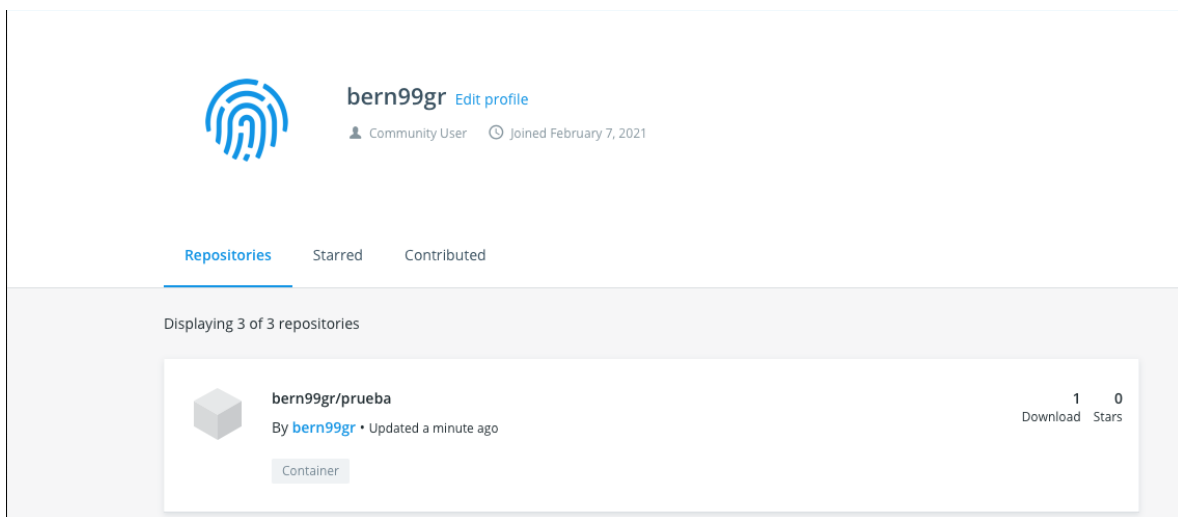
The screenshot shows the VS Code interface. The Explorer panel on the left shows a project named 'EJEMPLO' with files 'about.html', 'Dockerfile', 'index.html', and 'services.html'. The Dockerfile is open in the editor, showing the following content:

```
Dockerfile > ...
1 FROM nginx:latest
2
3 WORKDIR /usr/share/nginx/html
4
5 COPY . .
```

The terminal window at the bottom shows the command `docker push bern99gr/prueba` being executed. The output indicates that the image was successfully pushed to the repository `docker.io/bern99gr/prueba`. The terminal output includes the following information:

```
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$ docker push bern99gr/prueba
Using default tag: latest
The push refers to repository [docker.io/bern99gr/prueba]
3a083a5e58e5: Mounted from bern99gr/test
5f70bf18a086: Mounted from bern99gr/test
85fcec7ef3ef: Mounted from bern99gr/test
3e5288f7a70f: Mounted from bern99gr/test
50bc37de0858: Mounted from bern99gr/test
1c91b169a08b: Mounted from bern99gr/test
cb42413394c4: Mounted from bern99gr/test
latest: digest: sha256:1ebbd8aec4a21e883476acaa97b0e9925676ba6115ce986664bd06f2ee9b7ccb size: 1775
(base) MacBook-Pro-de-Mac:Ejemplo bernardo$
```

21. Para revisar que se hizo exitosamente, revisar tu perfil de DockerHub.



Y listo. Ya otras personas pueden descargar esa misma imagen que puede contener eso y hasta más dependencias a parte de NGINX.

Si quieres poder contar con ayuda de Docker en VSCode, descargar el plugin de Docker.



Ejemplo de implementación con NGINX, PHP, HTTPS y HTTP/2 (en base a la práctica)

1. Crear un archivo llamado *docker-compose.yml* que contenga la siguiente información:

```
web:
  image: nginx
  container_name: servidores
  ports:
    - "3000:80"
    - "5000:443"
  volumes:
    - ./code:/etc/nginx/html
    - ./site.conf:/etc/nginx/conf.d/site.conf
    - ./Certificados:/etc/nginx/private
  links:
    - php
php:
  image: php:7-fpm
  container_name: phpServer
  volumes:
    - ./code:/etc/nginx/html
db:
  image: mariadb
  container_name: db
  restart: always
  ports:
    - "7000:3306"
  environment:
    MYSQL_DATABASE: bernardo
    MYSQL_ROOT_PASSWORD: 12345
    SERVICE_TAGS: dev
    SERVICE_NAME: mysql
```

Aquí podemos ver que tenemos 3 servicios, uno llamado *web* (con la imagen de nginx), otro *php* (con la imagen de php:7-fpm) y otro *db* con la imagen: mariadb. Cada uno con sus respectivas configuraciones que se pueden determinar en la documentación oficial de DockerHub.

2. De igual forma, contar con un archivo *.conf* para poder hacer ajustes directamente en tu imagen de NGINX. El archivo a continuación está nombrado *site.conf*.



```
1 server {
2     index index.html;
3     listen 80 http2;
4     listen 443 ssl http2;
5     server_name bernardo.dominio;
6     ssl_certificate /etc/nginx/private/bernardo.dominio.crt;
7     ssl_certificate_key /etc/nginx/private/bernardo.dominio.key;
8
9     location ~ /\.php$ {
10         try_files $uri =404;
11         fastcgi_split_path_info ^(.+\.php)(/.+)$;
12         fastcgi_pass php:9000;
13         fastcgi_index index.php;
14         include fastcgi_params;
15         fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
16         fastcgi_param PATH_INFO $fastcgi_path_info;
17     }
18 }
```

Como pueden observar, los certificados fueron creados y asignados a dicha carpeta.

La asignación de dicho lugar se encuentra en el apartado *volumes* de la imagen de NGINX en el archivo *docker-compose.yml*.

Nota: En ejemplos como esos cuando se usa PATH1:PATH2, PATH1 hace alusión a la dirección real de tu computadora, es decir, donde tú tienes acceso a tus archivos. PATH2 hace alusión a la dirección del respectivo servicio. En este caso, mis certificados que están guardados en una carpeta nombrada *Certificados* la estoy copiando a la dirección */etc/nginx/private*.

3. Correr el siguiente comando para tener funcionando tus servidores:

```
(base) MacBook-Pro-de-Mac:website bernardo$ docker-compose up
```



CONCLUSIONES

Las ventajas de Docker son muy claras. A parte de que terminan siendo una mejor opción que máquinas virtuales para el desarrollo en diferentes ambientes debido a su mejor rendimiento, también lo hace muy fácil para nosotros. Gracias a que Docker es más un servicio de contenedores que ha estado evolucionando con los años, se ve que pueden simplificar mucho la vida de los desarrolladores.

Pareciera que los contenedores y servicios como Docker posiblemente van a terminar evolucionando a algo como GitHub, una de las herramientas más conocidas y usadas por desarrolladores. Lo importante aquí es que si te piensas convertir en un desarrollador profesional, conocer herramientas como Docker van a ser clave e indispensables para que puedas tener mejores oportunidades en el ámbito laboral, ya sea porque sea requisito conocerla o porque puedas proponer nuevas soluciones con estas nuevas herramientas que acabas de conocer.



BIBLIOGRAFÍA/REFERENCIAS

Fazt Code. (2020). Docker, Curso Práctico para principiantes (desde Linux) [Archivo de video]. Recuperado de:

<https://www.youtube.com/watch?v=NVvZNmfgg6M&list=PLo5lAe9kQrwpmayOSKOTfHv70t8NHXa8x&index=8>

Docker (2021). What is a Container. Recuperado de:

<https://www.docker.com/resources/what-container>

Red Hat (2020). ¿Qué es DOCKER?. Recuperado de:

<https://www.redhat.com/es/topics/containers/what-is-docker>