

# Montar Servidor Apache2 en Contenedor Docker

Acosta Rosales Jair Sebastián

1 de enero de 2020

## 1. Introducción

Docker es una herramienta de virtualización basado en el despliegue de contenedores, comenzó como un proyecto de dotCloud por Solomon Hykes y Sebastien Pahl debutando finalmente para todo el público en PyCon Santa Clara en el año 2013.

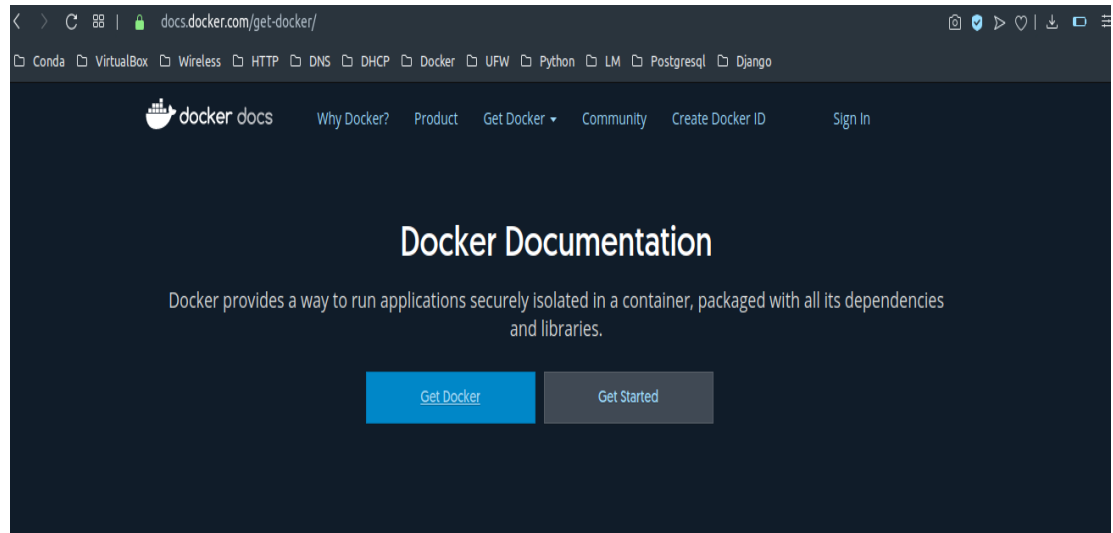
Esta herramienta permite llevar a cabo la virtualización de un entorno apto para alguna aplicación requerida, sin la necesidad de usar las máquinas virtuales tradicionales, las cuales necesitaban ocupar una cantidad considerable de recursos para ser montadas en un servidor, con Docker no se necesita un nuevo sistema operativo para que las aplicaciones montadas en los contenedores puedan funcionar, si bien si es necesario que corran en un ambiente base, este solo usará lo necesario para que la aplicación pueda ser ejecutada sin ningún tipo de problema, manteniendo la estabilidad de la aplicación, la portabilidad para ser ejecutada en otra computadora y el rápido despliegue de esta para poder ofrecer sus servicios lo más pronto posible.

Con docker como herramienta de virtualización se pretende que un mismo equipo de trabajo pueda trabajar sobre la misma versión de la aplicación sin necesidad de preocuparse de las dependencias que se necesitan en cada una de las computadoras de trabajo, pues la imagen(nombre dado al archivo creado específicamente para la aplicación) ya contiene todas esas dependencias requeridas, por lo que permite aprovechar ese tiempo requerido en la instalación de dependencias en el desarrollo y pruebas de la aplicación.

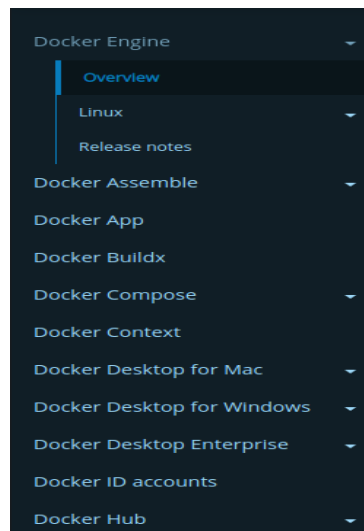
Otra de las grandes ventajas de docker es el uso de una memoria cache que permite crear nuevas imágenes a partir de los paquetes que ya se instalaron en otras aplicaciones, es decir que reusa los paquetes para ahorrar espacio.

## 2. Instalación de Docker en SO Linux Mint 19.2 Tina

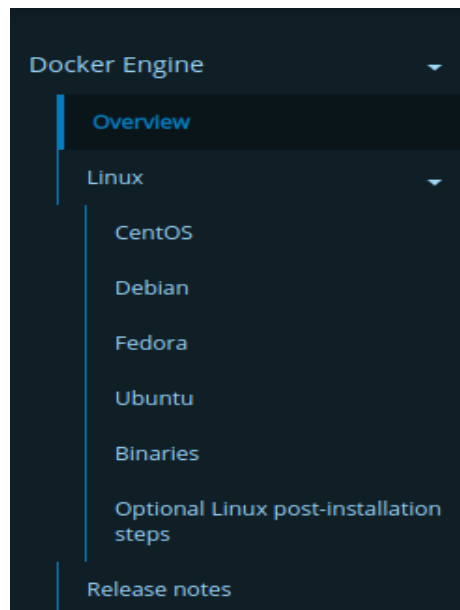
1. Primeramente debemos acceder a la url: <https://docs.docker.com/>, allí debemos ir al apartado "Get Docker".



2. Se desplegará un menú a la izquierda, en el que debemos seleccionar el apartado "Linux".



3. Una vez desplegado el menú de Linux, seleccionaremos el S.O más parecido a Linux Mint, es decir Ubuntu, dado que ambos están basados en Debian.



4. A nuestra derecha se abrirá el apartado de instrucciones para instalar Docker en diferentes versiones de Ubuntu, en este caso la versión de Linux Mint 19.2 Tina usada es bastante similar a Ubuntu Bionic 18.04(LTS), ambas son las versiones recientes más estables respectivamente, basadas en el Sistema Operativo Debian.

## OS requirements

To install Docker Engine - Community, you need the 64-bit version of one of these Ubuntu versions:

- Disco 19.04
- Cosmic 18.10
- Bionic 18.04 (LTS)
- Xenial 16.04 (LTS)

5. Nos aseguraremos de no tener una versión de docker ya instalada, ejecutando en consola el siguiente comando:

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

De este modo eliminamos alguna versión antigua o nos aseguramos de no tenerlo instalado.

6. Bajamos hasta el apartado denominado 'Install using the convenience script'.

**Install using the convenience script**

Docker provides convenience scripts at [get.docker.com](https://get.docker.com) and [test.docker.com](https://test.docker.com) for installing edge and testing versions of Docker Engine - Community into development environments quickly and non-interactively. The source code for the scripts is in the `docker-install` repository. **Using these scripts is not recommended for production environments**, and you should understand the potential risks before you use them:

- The scripts require `root` or `sudo` privileges to run. Therefore, you should carefully examine and audit the scripts before running them.
- The scripts attempt to detect your Linux distribution and version and configure your package management system for you. In addition, the scripts do not allow you to customize any installation parameters. This may lead to an unsupported configuration, either from Docker's point of view or from your own organization's guidelines and standards.
- The scripts install all dependencies and recommendations of the package manager without asking for confirmation. This may install a large number of packages, depending on the current configuration of your host machine.
- The script does not provide options to specify which version of Docker to install, and installs the latest version that is released in the "edge" channel.
- Do not use the convenience script if Docker has already been installed on the host machine using another mechanism.

This example uses the script at [get.docker.com](https://get.docker.com) to install the latest release of Docker Engine - Community on Linux. To install the latest testing version, use [test.docker.com](https://test.docker.com) instead. In each of the commands below, replace each occurrence of `get` with `test`.

Justamente en este apartado se encuentran dos comandos que permiten la descarga y la ejecución del instalador para docker, el primero:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
```

De modo que podemos observar la descarga de un archivo llamado `get-docker.sh`

```
jsebastian-ar@jSebastian-AR:~/Escritorio$ ls
000-default.conf  bind 'Cuentas gmail HDS'  SitioWeb_1  SitioWeb_2  station.png
jsebastian-ar@jSebastian-AR:~/Escritorio$ curl -fsSL https://get.docker.com -o get-docker.sh
jsebastian-ar@jSebastian-AR:~/Escritorio$ ls
000-default.conf  bind 'Cuentas gmail HDS'  get-docker.sh  SitioWeb_1  SitioWeb_2  station.png
jsebastian-ar@jSebastian-AR:~/Escritorio$
```

Una vez descargados los paquetes, procedemos a instalarlos con el comando:

```
$ sudo sh get-docker.sh
```

7. Para finalizar procedemos a revisar la versión instalada con el comando:

```
$ sudo docker version
```

```
jsebastian-ar@jsebastian-ar:~/Escritorio$ sudo docker version
[sudo] password for jsebastian-ar:
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:25:38 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:           19.03.5
  API version:       1.40 (minimum version 1.12)
  Go version:        go1.12.12
  Git commit:        633a0ea838
  Built:             Wed Nov 13 07:24:09 2019
  OS/Arch:           linux/amd64
  Experimental:      false
 containerd:
  Version:           1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:           1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f80e6d94a8c831b9d5aa481657
 docker-init:
  Version:           0.18.0
  GitCommit:        fec3683
```

### 3. Creación de imagen para servidor Apache2

Para la creación de la imagen en donde nuestro servidor Apache2 estará dando servicio, es necesario tener los archivos de configuración del servicio(Apache2), además de los archivos de cada uno de las páginas web que tenemos, un archivo que llamaremos Dockerfile el cual contiene todas las instrucciones necesarias para crear nuestra imagen con la configuración requerida para el servidor, como primer paso crearemos una carpeta donde reuniremos todos los archivos para construir la imagen, esta carpeta será llamada apache2, dentro de esta carpeta comenzaremos por guardar los sitios web con los que contamos.

1. Se tienen 3 sitios web distintos, cada uno de ellos sera guardado de forma anidada en dos carpetas, de modo que al guardarlo dentro de la carpeta apache2 anteriormente creada la jerarquía de carpetas queda de la siguiente forma:

- apache2/Sitio1/helloworld/
- apache2/Sitio2/lapalma/
- apache2/Sitio3/nightbeach/

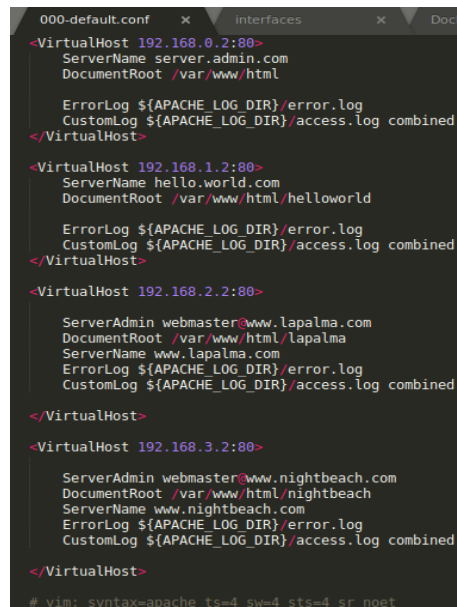
El porqué de esta forma de guardar los archivos en dos carpetas en lugar de una será explicado más adelante al momento de crear la imagen con todos los archivos reunidos.

2. Es momento de obtener los archivos de configuración de nuestro servidor para que este otorgue el servicio del modo que nosotros deseamos, en este caso para el servidor apache2, nos centraremos en dos archivos específicos, los archivos son:

- **apache2.conf**, ubicado en la carpeta `/etc/apache2/`
- **000-default.conf**, ubicado en la carpeta `/etc/apache2/sites-available`

Para este punto es recomendable tener una base de estos archivos, por lo que si se tiene el servidor apache2 instalado en el sistema operativo, se puede acceder a estas rutas y copiar y pegar estos archivos dentro de la carpeta apache2 creada con anterioridad, en este caso se tenía esta base de los archivos, por lo que se procedió a copiar y pegar y realizar algunas configuraciones en ellos, particularmente en el archivo 000-default.conf, el archivo apache2.conf se dejo exactamente con la configuración por default con la que se instala pero se agrego igualmente por si se desea realizar una modificación a este más adelante.

Procedemos a configurar el archivo **000-default.conf** de modo que deseamos que cada página este disponible en una dirección ip distinta a la de las demás páginas web, de modo que nuestro archivo de configuración queda de la siguiente forma:



```
000-default.conf x interfaces x Dock
<VirtualHost 192.168.0.2:80>
  ServerName server.admin.com
  DocumentRoot /var/www/html

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost 192.168.1.2:80>
  ServerName hello.world.com
  DocumentRoot /var/www/html/helloworld

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost 192.168.2.2:80>
  ServerAdmin webmaster@www.lapalma.com
  DocumentRoot /var/www/html/lapalma
  ServerName www.lapalma.com
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost 192.168.3.2:80>
  ServerAdmin webmaster@www.nightbeach.com
  DocumentRoot /var/www/html/nightbeach
  ServerName www.nightbeach.com
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

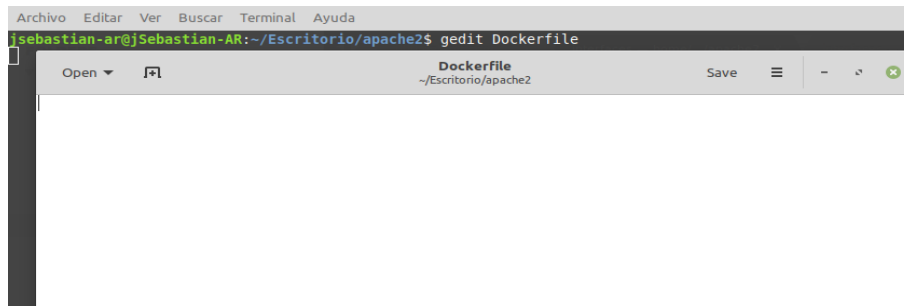
De este modo definimos un VirtualHost para cada uno de los sitios(incluyendo la página por default) cada una en una ip distinta, por lo que resumiendo cada sitio tendrá la siguiente ip:

- **default** ⇒ 192.168.0.2
- **helloworld** ⇒ 192.168.1.2
- **lapalma** ⇒ 192.168.2.2
- **nightbeach** ⇒ 192.168.3.2

3. Finalmente procedemos a crear el archivo Dockerfile, desde la terminal nos situamos en la carpeta apache2 donde hemos estado guardando los archivos y ejecutamos el comando:

```
$ gedit Dockerfile
```

Inmediatamente el editor de texto se abrirá, presionaremos save y cerraremos el archivo.



Procederemos a abrir el archivo con el editor de texto sublimetext, el archivo en estos momentos esta vacío, pero procederemos a escribir diferentes instrucciones de modo que el archivo quede de la siguiente manera:

```
Dockerfile  X  000-default.conf  X  untitled
FROM ubuntu:18.04

#Se descarga el servidor http apache2
RUN apt-get update && apt-get -y install apache2
#Se descarga la paqueteria de redes que permite visualizar las interfaces
RUN apt-get install -y net-tools
#Editor de texto de consola
RUN apt-get install -y nano

#El puerto 80 queda abierto para que otros contenedores puedan conectarse al servicio
EXPOSE 80

#Se copian los archivos de las páginas web
COPY Sitio1 /var/www/html/
COPY Sitio2 /var/www/html/
COPY Sitio3 /var/www/html/

#Se copian los archivos de configuración deseados para customizar nuestro servidor al modo deseado
COPY apache2.conf /etc/apache2/
COPY 000-default.conf /etc/apache2/sites-available

#Se ejecuta apache en primer plano
CMD ["apachectl", "-D", "FOREGROUND"]
```

Como se puede observar en cada línea hay distintas instrucciones, con un significado particular:



- a) **FROM ubuntu:18.04** establece el S.O en el que la imagen va a estar basada para crear los archivos y directorios respectivos de cada contenedor basado en esa imagen.
- b) **RUN apt-get update && apt-get -y install apache2** ejecuta el comando dado como si un usuario desde consola lo estuviera ejecutando para actualizar e instalar el servidor apache2, notesé que la bandera **-y** es agregada para que se acepte por default instalar los archivos sin necesidad de responder manualmente a la pregunta.
- c) **RUN apt-get install -y net-tools** este paquete permitirá hacer uso de comandos como **ifconfig** para visualizar cada una de las interfaces de red con las que cuenta el contenedor, algo que será muy importante más adelante.
- d) **RUN apt-get install -y nano** El editor de texto que permitirá visualizar como es que nuestros archivos de configuración customizados son agregados al contenedor.
- e) **EXPOSE 80**, permite que el contenedor pueda comunicarse con otros contenedores a través de ese puerto.
- f) **COPY Sitio1 /var/www/html/**  
**COPY Sitio2 /var/www/html/**  
**COPY Sitio3 /var/www/html/**

Copia cada una de las carpetas que contienen las páginas web a desplegar, es en este punto donde es toma importancia que las carpetas de las páginas web hayan sido anidadas, dado que el comando **COPY** entiende que hay que copiar todo lo que esta dentro de la carpeta específica, por lo que la carpeta anidada en cada sitio(helloworld, la-palma, nightbeach) serán copiadas por completo con su respectivo contenido dentro de ellas, si en cambio intentáramos copiarlas directamente, es decir **COPY helloworld /var/www/html**, etc. solo el contenido de éstas será copiada por lo que puede provocar un desorden de archivos de distintos sitios web.

- g) **COPY apache2.conf /etc/apache2/**  
**COPY 000-default.conf /etc/apache2/sites-available**

Estos comandos permiten agregar los archivos de configuración del servidor apache en los respectivos directorios del servidor Apache2 que se encuentran en la imagen(por eso es importante que el servidor Apache2 ya este instalado), de modo que el servidor de la imagen

leerá la configuración de estos archivos customizados por nosotros y dará el servicio tal y como lo deseamos.

*h)* **CMD** [“**apachectl**”, “**-D**”, “**FOREGROUND**”]

Este comando ejecutará el servidor Apache2 en primer plano, de modo que sea el proceso que se estará ejecutando durante toda la vida del contenedor, esto es importante dado que la vida de un contenedor de docker depende del proceso que se esté ejecutando, por lo que un proceso que no conlleve estar en constante ejecución y termine pronto hará que el contenedor de docker también termine su ejecución, es gracias a este comando que el contenedor de docker estará ejecutando el servicio de Apache2 por lo tanto el contenedor vivirá todo el tiempo que el mismo proceso del servicio viva.

4. Como último paso crearemos la imagen para nuestro servicio, para ello debemos estar una carpeta antes de la carpeta apache2 donde guardamos todos los archivos de configuración, de modo que si ejecutamos el comando **ls** la única carpeta que debe aparecer debe ser **apache2**:

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ ls
apache2
```

Procedemos a ejecutar el comando con el que la imagen se construirá, el cual es:

**\$ sudo docker build apache2 -t apache2:ips**

Los parámetros de este comando son:

- a)* **build** el cual indica a docker que se construirá una imagen.
- b)* **apache2** el nombre de la carpeta donde se encuentra nuestro Dockerfile y el resto de archivos necesarios para nuestra imagen.
- c)* **-t** indica que lo siguiente será el nombre de la imagen así como un respectivo tag de identificación.
- d)* **apache2:ips** como se explico en el anterior punto, este parámetro consta de dos partes, el primero es **apache2** el cual es el nombre de la imagen que se creará, finalmente y separado por un **:** está el tag (opcional, si se omite, docker por default agregará el tag latest) el cual es **ips**.

Ejecutado este comando comenzará el proceso para crear la imagen, donde podemos observar como los comandos definidos en el dockerfile son ejecutados uno por uno y se comienzan a realizar la instalación de los paquetes requeridos.

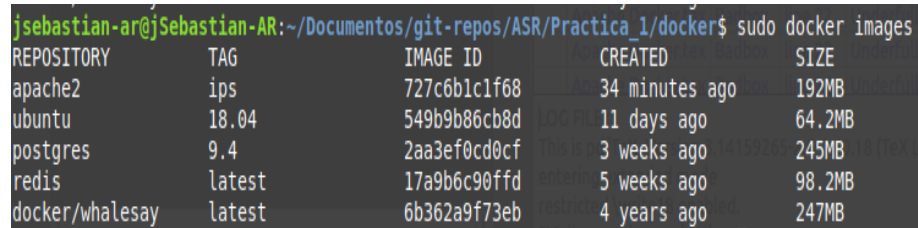
```

jsebastian-ar@Sebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker build apache2 -t apache2:ips
Sending build context to Docker daemon  580.6kB
Step 1/11 : FROM ubuntu:18.04
----> 549b9b86cb8d
Step 2/11 : RUN apt-get update && apt-get -y install apache2
----> Running in 9bdf6ef82858
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [795 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:8 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [19.2 kB]
Get:9 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [761 kB]
Get:10 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [6781 B]
Get:11 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Step 3/11 : RUN apt-get install -y net-tools
----> Running in 09fc133eab4b
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  net-tools
Step 4/11 : RUN apt-get install -y nano
----> Running in c0c2c6642321
Reading package lists...
Building dependency tree...
Reading state information...
Suggested packages:
  spell
The following NEW packages will be installed:
  nano
Step 5/11 : EXPOSE 80
----> Running in 5c9ce803324f
Removing intermediate container 5c9ce803324f
----> 92d501ca8828
Step 6/11 : COPY Sitio1 /var/www/html/
----> c64bc7a32988
Step 7/11 : COPY Sitio2 /var/www/html/
----> 44feb5b35406
Step 8/11 : COPY Sitio3 /var/www/html/
----> 3cef532c7cde
Step 9/11 : COPY apache2.conf /etc/apache2/
----> 6ac5b93fdf4b
Step 10/11 : COPY 000-default.conf /etc/apache2/sites-available
----> 1a2185240983
Step 11/11 : CMD ["apachectl","-D","FOREGROUND"]
----> Running in c07a36b4160f
Removing intermediate container c07a36b4160f
----> 727c6b1c1f68
Successfully built 727c6b1c1f68
Successfully tagged apache2:ips

```

Ejecutar comando para visualizar la imagen recientemente creada podremos ver la lista de imagenes disponibles que tenemos, de modo que podremos observar la imagen creada recientemente:

**\$ sudo docker images**



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
apache2	ips	727c6b1c1f68	34 minutes ago	192MB
ubuntu	18.04	549b9b86cb8d	11 days ago	64.2MB
postgres	9.4	2aa3ef0cd0cf	3 weeks ago	245MB
redis	latest	17a9b6c90ffd	5 weeks ago	98.2MB
docker/whalesay	latest	6b362a9f73eb	4 years ago	247MB

Figura 1: Se observa como primer resultado la imagen recién creada.

## 4. Creación de subredes en Docker

La creación de subredes en Docker permite a un contenedor estar conectado a múltiples redes a las cuales puede ofrecer el servicio por el cual es creado, de esta forma es posible la comunicación entre otros contenedores o también es posible que el servicio este disponible fuera del mismo entorno de Docker, es decir para que usuarios accedan al servicio.

Para la creación de subredes hay que tener en cuenta las direcciones ip en donde nuestras páginas web van a dar su servicio, el cual como se menciono con anterioridad quedo de la siguiente forma:

- **default**  $\Rightarrow$  192.168.0.2
- **helloworld**  $\Rightarrow$  192.168.1.2
- **lapalma**  $\Rightarrow$  192.168.2.2
- **nightbeach**  $\Rightarrow$  192.168.3.2

Observamos que cada una de las ip's pertenecen a una subred distinta, esto es porque al momento de conectar un contenedor a una subred, se crea una interfaz de red específica para esa red dentro del contenedor y se le asigna una ip nueva al contenedor para esa interfaz de ese segmento de red al que se conecto, esto es necesario, ya que no es posible crear interfaces virtuales (como normalmente sería) de un mismo segmento de red en un contenedor, por lo tanto, para tener varias interfaces en un mismo contenedor es necesario tener varios segmentos de red (que no se repitan) para que al conectar el contenedor a estas subredes se puedan tener varias interfaces con una ip específica en donde las páginas web estarán alojadas.

Una vez entendido esto, procedemos a crear cada uno de los segmentos de red que proporcionaran las ip's para nuestras páginas web, usando el comando:

```
$ sudo docker network create --driver param0 --subnet param1  
--gateway param2 network_name
```

Donde:

- **network create**: network indica una operación para redes, mientras que create indica que la operación será la creación de una nueva red
- **--driver param0**: con la bandera --driver podremos indicar el tipo de subred que deseamos crear, hay tres tipos, bridge: el cual permite la conectividad entre contenedores, así como al equipo externo donde se ejecuta el contenedor, none: que aísla totalmente de comunicación al contenedor y host: que permite heredar todas las interfaces que contiene el equipo host sobre el cual el contenedor se está ejecutando, en nuestro caso el valor de **param0** será **bridge**.

- **–subnet param1**: indicara el segmento de red(ya sea de tipo A,B ó C), en este parámetro debemos tener cuidado ya que este segmento de red debe ser el que contenga a la ip de nuestras páginas web.
- **–gateway param2**: este parámetro indica la dirección ip por la cual nuestra subred mandará paquetes fuera de la misma para conectarse con otras redes.
- **network\_name**: este parámetro será el nombre de nuestra subred.

Ahora procedemos a crear las subredes ejecutando el comando anterior, pero con los respectivos valores para cada una de las subredes que convengan a nuestras páginas web:

- a) **default**: para esta subred el comando es: `$sudo docker network create –driver bridge –subnet 192.168.0.0/24 –gateway 192.168.0.1 subnet0`

De modo que podemos ver que el segmento de red que contiene la ip elegida(192.168.0.2) es 192.168.0.0/24 a la cual se denomino subnet1.

- b) **helloworld**: para esta subred el comando es: `$ sudo docker network create –driver bridge –subnet 192.168.1.0/24 –gateway 192.168.1.1 subnet1`

Este segmento de red que contiene la ip elegida(192.168.1.2) para helloworld.

- c) **lapalma**: para esta subred el comando es: `$ sudo docker network create –driver bridge –subnet 192.168.2.0/24 –gateway 192.168.2.1 subnet2`

- d) **helloworld**: para esta subred el comando es: `$ sudo docker network create –driver bridge –subnet 192.168.3.0/24 –gateway 192.168.3.1 subnet3`

En la siguiente imagen se puede apreciar el listado de las redes una vez creadas con el comando **\$ sudo docker network ls**

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9319ff854878        bridge             bridge              local
eef0b09685ef        host              host               local
f6d85fba7579        none              null               local
ef6941f02cf2        subnet0           bridge             local
5c3c85f55baf        subnet1           bridge             local
61e663c147be        subnet2           bridge             local
3902cdb3d31f        subnet3           bridge             local
```

## 5. Creación de contenedor

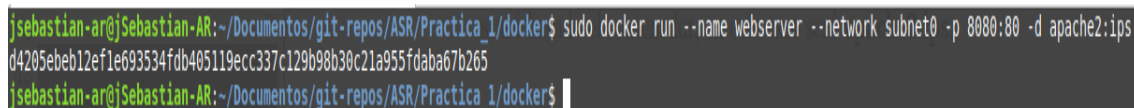
Para la creación de un nuevo contenedor basado en la imagen recientemente creada ejecutaremos el siguiente comando:

```
$ sudo docker run --name name_container --network network_name  
-p host_port:container_port -d name_image
```

Donde:

- **run**: indica la creación de un contenedor.
- **--name name\_container**: El valor de `name_container` indicará el nombre del contenedor.
- **--network network\_name**: El valor de `network_name` indicará el nombre de la red a la que está conectado el contenedor desde su creación, solo es posible conectarlo a una red en un inicio, posteriormente se pueden hacer las conexiones del contenedor con varias subredes.
- **-p host\_port:container\_port**: esta bandera permitirá al contenedor conectar uno de sus puertos con uno de los puertos(libres) del host, para que de esta manera sea accesible el servicio desde fuera de la red interna de docker, el valor `host_port` indica el puerto usado por el host mientras que `container_port` es el puerto usado por el contenedor.
- **-d**: indica que la ejecución del contenedor no será explícitamente visible desde consola, aunque el servicio sí haya sido iniciado.
- **name\_image**: el nombre de la imagen(incluyendo tag) de la cual estará basado el contenedor, en este caso será **apache2:ips**.

Una vez ejecutado el comando aparecerá una clave hash, que indicará el ID del contenedor:



```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker run --name webserver --network subnet0 -p 8080:80 -d apache2:ips  
d4205eb12ef1e693534fdb405119ecc337c129b98b30c21a955fdaba67b265  
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$
```

Figura 2: Podemos observar que el contenedor se llama **webserver**, la subred a la que se conecta de inicio es **subnet0** y el puerto **80**(puerto para http) del contenedor estará dando servicio en el puerto **8080** del host

En este momento nuestro contenedor ya esta en ejecución(se puede comprobar con el comando **\$sudo docker ps**), por lo tanto el servidor apache ya está dando servicio, sin embargo, dado que el contenedor solo fue conectado a una red(subnet0) la única página disponible es la página **default**, esto se puede comprobar si entramos directamente a la consola del contenedor con el comando:

### **\$ sudo docker exec -it webserver bash**

El comando **exec** nos permite ejecutar un comando de consola en un contenedor de docker, mientras que la bandera **-it** hará que la ejecución se haga de forma interactiva, es decir nosotros podremos ver el resultado del comando una vez que lo ejecutemos, posteriormente se agrega el nombre del contenedor en ejecución y el comando deseado, en este caso será el comando **bash**, dado que deseamos desplegar la consola del contenedor.

```
jsebastian-ar@jsebastian-AR:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS               NAMES
d4205eb12e    apache2:ips "apachectl -D FOREGR..." 16 minutes ago Up 16 minutes    0.0.0.0:8080->80/tcp webserver
jsebastian-ar@jsebastian-AR:~$ sudo docker exec -it webserver bash
root@d4205eb12e:/#
```

Si ejecutamos el comando **ifconfig** se desplegarán cada una de las interfaces de red disponibles.

```
jsebastian-ar@jsebastian-AR:~$ sudo docker exec -it webserver bash
root@d4205eb12e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255
    ether 02:42:c0:a8:00:02 txqueuelen 0 (Ethernet)
    RX packets 94 bytes 15201 (15.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 3: Observamos la interfaz referente a subnet0, donde nos otorgo la ip 192.168.0.2(una después de gateway)



## 6. Conectar contenedor a diferentes subredes

Una vez creado nuestro contenedor y estando en ejecución es posible conectarlo a las subredes restantes que permitirán que las otras páginas web estén disponibles, para ello se usará el comando:

```
$ sudo docker network connect network_name name_container
```

Donde:

- **network connect**: network indica que es una operación que involucra una red, mientras que connect explícitamente indica que la operación será la conexión entre una subred y un contenedor.
- **network\_name**: indica el nombre de la subred a conectar.
- **name\_container**: indica el nombre del contenedor a conectar.

De modo que debemos hacer la conexión entre nuestro contenedor llamado **webserver** y las subredes **subnet1,subnet2,subnet3**, de la siguiente manera:

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker network connect subnet1 webserver
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker network connect subnet2 webserver
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_1/docker$ sudo docker network connect subnet3 webserver
```

Volviendo a la consola del contenedor, notamos que si ejecutamos el comando **ifconfig** el resto de interfaces que otorgan las ip's para el resto de sitios web ahora están disponibles.

```
root@d4205eb12e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255
    ether 02:42:c0:a8:00:02 txqueuelen 0 (Ethernet)
    RX packets 168 bytes 28991 (28.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.2 netmask 255.255.255.0 broadcast 192.168.1.255
    ether 02:42:c0:a8:01:02 txqueuelen 0 (Ethernet)
    RX packets 63 bytes 9050 (9.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0


eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.2 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 02:42:c0:a8:02:02 txqueuelen 0 (Ethernet)
    RX packets 69 bytes 9021 (9.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.2 netmask 255.255.255.0 broadcast 192.168.3.255
    ether 02:42:c0:a8:03:02 txqueuelen 0 (Ethernet)
    RX packets 61 bytes 7849 (7.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

## 7. Pruebas

Una vez finalizada la configuración podemos acceder a los sitios web, desde el navegador usando las ip's definidas para cada uno de ellos.

### Página default



# ubuntu

## Apache2 Ubuntu Default Page

**It works!**

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

### Configuration Overview

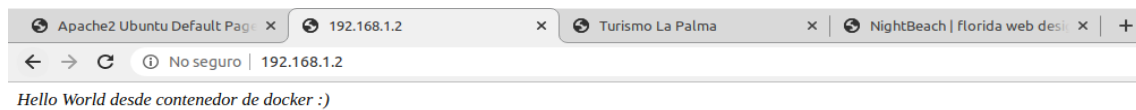
Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in** [/usr/share/doc/apache2/README.Debian.gz](#). Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for

## Página helloworld



## Página lapalma

Ubuntu Default Page x 192.168.1.2 x Turismo La Palma x NightBeach | florida web desi x +

No seguro | 192.168.2.2

Turismo La Palma Información Lugares Vídeos Astrofísica

Search

# La Palma.

La Palma es una isla del océano Atlántico perteneciente al archipiélago de Canarias (España). La Palma tiene una población de 81.486 habitantes y ocupa el quinto lugar en extensión y es, tras Tenerife, la segunda de Canarias en altitud con los 2.426 metros del Roque de los Muchachos ([vía wikipedia](#)).

### Capital

Santa Cruz de La Palma es un municipio y localidad española, capital de la isla de La Palma, adscrita y perteneciente administrativamente a la provincia de Santa Cruz de Tenerife (Canarias).

Detalles »

### Naturaleza

Debido a su formación y localización, La Palma presenta una gran variedad de paisajes, debido a la diversidad de ecosistemas que presenta, desde los áridos costeros hasta la muy húmeda formación boscosa de la laurisilva, además de bosques de pinares y un ecosistema de alta montaña

Detalles »

### Historia

A lo largo de la historia La Palma ha recibido numerosos nombres. Actualmente son muy populares los sobrenombres de: La Isla Bonita, La Isla Verde o La Isla Corazón.

Detalles »

## Página nightbeach

