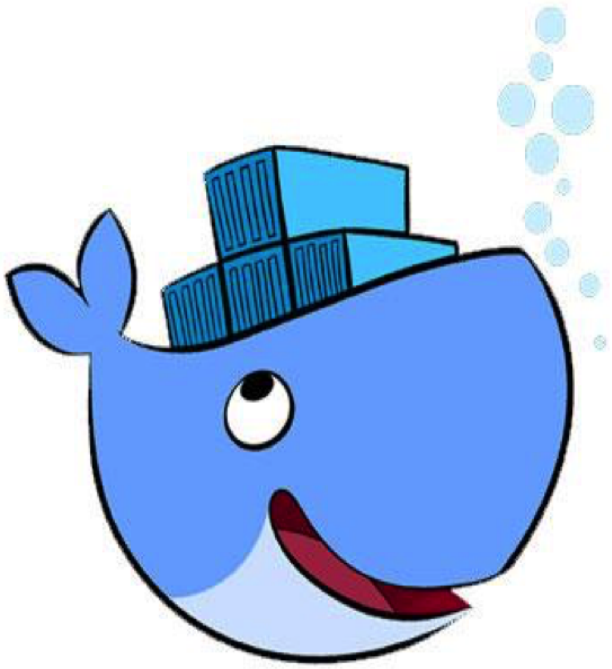
PROYECTO FINAL

Hosting Hostcker



Realizado por: Daniel Bascón Arenas

Índice

1. Introducción

Como selección de proyecto, he elegido la creación de un Hosting con Docker, utilizando una api creado por mi escrita en python, donde un usuario se dará de alta en mi hosting y se le proporcionará una base de datos y un espacio en memoria para poder desplegar alguna aplicación o algún CMS.

La elección de Hosting es por la razón de que tocamos varias ramas de un perfil de un Administrador de Sistemas, y la elección de Docker, por lo que nos ofrece, principalmente el “aislamiento” que nos ofrece.

Iremos explicando poco a poco que es docker y como he realizado mi aplicación.

2. Desarrollo

2.1. ¿Qué es un Hosting?

Hosting o alojamiento web es el servicio que provee a los usuarios de Internet un sistema para poder almacenar información, imágenes, vídeo, o cualquier contenido accesible vía web.

Las compañías que proporcionan espacio de un servidor a sus clientes se suelen denominar con el término en inglés web host.

Las compañías cuando nos damos de alta nos proporciona un nombre de dominio único en internet para poder acceder, esto se realiza a través de DNS.

No todos los servicios de alojamiento web sirven para todos los proyectos, por tanto, es necesario examinar una serie de criterios en la contratación del servicio. Entre los criterios más importantes se encuentran los siguientes:

- Los dominios: se pueden adquirir por separado (con otro proveedor) o con el mismo proveedor de alojamiento web.
- Previsión de tráfico: Los hospedajes web muy baratos suelen tener muchas una capacidad de tráfico limitada (menos de 1.000 visitas diarias). Especialmente a la hora de migrar una web de un alojamiento a otro hay que tener cuidado con esto.
- La capacidad del servidor: Capacidad de proceso (capacidad de CPU), espacio en disco y ancho de banda disponible.
- El tipo de tecnología que utilizará la web: páginas estáticas HTML o aplicaciones PHP. El caso de usar aplicaciones, el servidor de alojamiento soportar su tecnología.
- Capacidad de hosting multidominio: es decir, que el alojamiento soporte una sola web (un único dominio) o varias webs con diferentes dominios.
- Seguridad de la sala de servidores o centro de procesamiento de datos: se trata de averiguar si el vendedor de hosting tiene un centro de procesamiento de datos que este protegido contra hackers o desastres naturales y que tengan sistemas de recuperación de datos confiables.

Un alojamiento web se puede diferenciar de otro por el tipo de sistema operativo, bases de datos y motor de generación de sitios web que exista en él. La combinación más conocida y extendida es la del tipo LAMP (Linux, Apache, MySQL y PHP), aunque se está comenzando a usar una combinación con Java.

Los servicios más comunes que se pueden incluidos en un alojamiento son los siguientes:

- Alojamiento de ficheros y acceso vía web a los ficheros para subidas, descargas, edición, borrado, etc.
- Acceso a ficheros vía FTP.
- Creación de bases de datos, típicamente MySQL en el caso de alojamientos basados en Linux y administración vía web de las basea de datos con herramientas web como phpMyAdmin.
- Cuentas de correo electrónico con dominio propio, gestión de listas de correo, acceso vía clientes de sobremesa (tipo MS Outlook, etc.) y acceso vía webmail a estas cuentas. Reenvía del correo a otras cuentas (incluso externas).
- Discos duros virtuales que se pueden configurar como unidad de red en un equipo local vía protocolos como WebDav.
- Copias de seguridad.
- Gestión de dominios y subdominios.
- Estadísticas de tráfico.

- Asistentes para la instalación rápida de paquetes software libre populares como WordPress, Joomla, etc.

2.2. ¿Qué ofrece mi Hosting actualmente?

Mi Hosting actualmente tiene las siguientes características:

- Creación de usuarios.
- Eliminación de usuarios.
- Servidor FTP.
- Servidor Apache.
- Servidor MySQL.
- Servidor net2ftp.
- PhpMyAdmin,

Más adelante explicaremos como funciona todo esto.

2.3. ¿Qué es Docker?

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux.

La idea de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado. Para ello utiliza una plataforma donde se puede compartir imágenes entre la comunidad.

Docker es una plataforma para desarrolladores y administradores de sistemas para desarrollar, implementar y ejecutar aplicaciones con contenedores.

La tecnología Docker usa el kernel de Linux y las funciones de este, como es Cgroups y namespaces, para que puedan crear procesos de forma independientes.

Esta independencia es lo que le da la propiedad de contenedor (aislamiento), la capacidad de ejecutar varios procesos y aplicaciones separados unos de otros para mejor uso de las infraestructuras, y mantener la seguridad.

2.4. Historia de Docker.

Su creador fue Salomón Hykes, junto a otros contribuyentes, que lo comenzó como proyecto interno dentro de dotCloud, empresa dedicada a Paas. Docker fue liberado en marzo de 2013, y un año más tarde dejo de utilizar LXC como el entorno de ejecución por defecto y lo reemplazó con su propia biblioteca “libcontainer”, escrito en Go.

2.5. ¿En que se diferencia Docker de Máquinas Virtuales?

Realmente el concepto es algo similar el de Docker con respecto a lo que se define como máquina virtual, pero no es lo mismo. Un contenedor es más ligero, ya que a una máquina de virtual hay que instalarle un sistema operativo para funcionar, en cambio, un contenedor Docker, utiliza el sistema operativo que tiene la máquina en la que se esta ejecutando el contenedor.

Un contenedor se ejecuta nativamente en Linux y comparte el kernel de la máquina host con otros contenedores. Se ejecuta un proceso discreto, no teniendo más memoria que cualquier otro ejecutable, por lo que es ligero.

Por el contrario, una máquina virtual (MV) ejecuta un sistema operativo completo con acceso virtual a recursos de host a través de un hipervisor. En general, las máquinas virtuales proporcionan un entorno con más recursos de los que la mayoría de las aplicaciones necesitan.

A la hora de seleccionar MV o Docker para ejecutar una aplicación es más viable la opción de Docker por lo dicho, menos consumo, además la facilidad de exportarlo.

También podemos destacar la velocidad de iniciarlo, una MV tiene que cargar el sistema operativo completo, mientras que el contenedor con un simple comando ya lo tenemos arrancado.

2.6. ¿Qué nos proporciona Docker como contenedor?

Dentro de las características que nos proporciona Docker podemos reflejar varias características, en las cuales podemos indicar:

- Flexibilidad: incluso las aplicaciones más complejas pueden contenerse.
- Ligero: los contenedores aprovechan y comparten el kernel de host.
- Actualizable: puede implementar actualizaciones y actualizaciones sobre la marcha.
- Portable: puedes construir localmente, implementar en la nube y ejecutar en cualquier lugar.
- Escalable: puede aumentar y distribuir automáticamente réplicas de contenedores.
- Apilable: puede apilar servicios y sobre la marcha.

Un contenedor se inicia ejecutando una imagen creada o descargada. Una imagen es un paquete ejecutable que incluye todo lo necesario para ejecutar una aplicación (el código, un tiempo de ejecución, bibliotecas, variables de entorno y archivos de configuración).

Cuando lanzamos un contenedor, este se convierte en memoria cuando se ejecuta. Podemos ver el proceso con las herramientas que nos proporciona Docker (“$ sudo docker ps -a”). Esto permite compartir una aplicación una aplicación, un servicio o conjuntos de estos, con tosas sus dependencias.

Estas herramientas desarrolladas en contenedores de Linux, es lo que hace a Docker fácil de usar y único, les otorgan a los usuarios un acceso sin precedentes a aplicaciones, la capacidad de implementar rápidamente, y controlar las versiones y su distribución. Esta ultima parte es muy importante, “el control de versiones”, nos proporciona la capacidad de añadir capas a nuestro contenedor e ir mejorándolo.

Como entorno de prueba, Docker es genial, ya que podemos usarlo como entorno de pruebas tantas veces como queramos, y crearnos tantos contenedores como queramos. Tirarlo y volverlos a levantar. Por ejemplo, podemos asociar un volumen a un contenedor en particular, eliminar el contenedor y volver a levantar otro contenedor con ese volumen, y los datos seguirían, hay persistencia.

Docker implementa una API de alto nivel para proporcionar contenedores livianos que ejecutan procesos de manera aislada. Construido sobre las facilidades proporcionadas por el kernel Linux (cgroups y namespaces), un contenedor Docker, a diferencia de una máquina virtual, no requiere

incluir un sistema operativo independiente. En su lugar, se basa en las funcionalidades del kernel y utiliza el aislamiento de recursos (CPU, RAM, E/S, red, etc.) y namespaces separados para aislar la vista de una aplicación del kernel Linux, ya sea a través de la librería de libcontainer o directamente de otras, como son libvirt, LXC o systemd-nspawn.

Usando Docker para crear y gestionar contenedores puede simplificar la creación de sistemas altamente distribuidos, permitiendo múltiples de aplicaciones, las tareas de los trabajadores y otros procesos para funcionar de forma autónoma en una única máquina física o en varias. Esto permite que el despliegue de nodos se realice a medida que se dispone de recursos o cuando se necesiten más nodos, lo que permite una plataforma como servicio (Paas) de estilo desplegable y ampliación de los sistemas como Apache, Cassandra, MongoDB o Riak. Docker también simplifica la creación y el funcionamiento de las tareas de carga de trabajo o las colas y otros sistemas distribuidos.

2.7. Integración de Docker.

Podemos integrar docker con diferentes herramientas de infraestructura:

- Amazon Web Services
- Ansible
- Chef
- Google Cloud Platform
- DigitalOcean
- Microsoft Azure
- OpenStack Nova
- Salt
- Puppent
- Vagrant
- Jenkins

2.8. Sistemas Operativos que soportan Docker.

Actualmente docker, es soportado por una amplía variedad de sistemas operativos, ya se han de escritorios, servidores e incluso cloud.

Puedes utilizar sistemas operativos de escritorio como:

- Linux
- Mac
- Windows 10

Además de versiones para servidores:

- Ubuntu Server
- Fedora
- CentOS
- Debian
- Otros

También en sistemas Cloud como:

- Amazon Web Services
- Azure
- Google Cloud

2.9. Componentes de Docker.

2.9.1. Docker Engine

Es un demonio que lanza docker y se ejecuta dentro del sistema operativo, puesto que Docker está basado en Linux (aunque actualmente hay versiones desarrollándose para Windows). A Docker Engine también se le denomina Docker Cliente y Servidor.

Lo que hace en realidad las instalaciones no Linux es levantar una máquina virtual con un Linux muy básico que contiene el demonio Docker, que es el que hace de servidor, y la parte cliente de Docker sí que puede ejecutar en otros sistemas operativos.

El Demonio o servidor es el proceso principal de gestión del engine que corre en la máquina anfitriona. El usuario nunca interactúa con el demonio directamente, sino que lo hace por medio del cliente.

2.9.2 Docker Hub

El Docker Hub es un servicio de registro basado en la nube para la construcción y envío de aplicaciones o servicios de contenedores. Proporciona un recurso centralizado para el descubrimiento de la imagen de contenedores , la distribución y la gestión del cambio , el usuario y la colaboración en equipo , y la automatización del flujo de trabajo a lo largo de la línea de desarrollo .

Específicamente , Docker Hub ofrece las siguientes características y funciones principales :

- Repositorio de imágenes: Buscar, administrar y enviar y obtener imágenes oficiales de la comunidad y bibliotecas de imágenes privadas.
- Construcciones automatizadas : Creación automática de nuevas imágenes cuando realiza cambios en una fuente GitHub o un repositorio Bitbucket.
- WebHooks : Una característica para automatizar construcciones, WebHooks permite desencadenar acciones después de enviar con éxito a un repositorio.
- Organizaciones: Crear grupos de trabajo para gestionar el acceso de usuarios a los repositorios de imágenes.
- Integración GitHub y Bitbucket: Añadir las imágenes alojadas en Docker Hub a sus flujos de trabajo.

2.9.3. Docker Machine

Docker Machine permite crear máquinas virtuales en diferentes proveedores, en una máquina anfitrión con VirtualBox hasta Amazon EC2 o Digital Ocean. La lista de controladores soportados es bastante amplia.

Para empezar a utilizar Docker, primero debemos configurar un demonio de Docker. Docker Machine configura automáticamente Docker en su ordenador, en los proveedores en la nube, y dentro de su centro de proceso de datos. Docker machine instala el demonio Docker (también llamado Docker engine) en los anfitriones y luego configura el cliente Docker para que se comunique con los demonios de Docker.

Docker machine automatiza todas las tareas de aprovisionamiento e instalación de un único host Docker ($ docker-machine create -d virtualbox dev).

2.9.4. Docker Compose

Nos permite definir aplicaciones de varios contenedores en un fichero con las mismas propiedades que indicaríamos con el comando “Docker run” individualmente. Con un único comando, podremos iniciar todos los contenedores y en el orden que especifiquemos.

El fichero nos puede servir no sólo como forma de iniciar los contenedores en un entorno de desarrollo, sino de documentación de la aplicación en la que veremos claramente que contenedores, imágenes, volúmenes, enlaces, variables, puertos, etc.

El fichero docker-compose es un archivo de texto con formato yaml en la que especificamos los diferentes contenedores y sus propiedades. El fichero debemos llamarlo docker-compose.yml, en función de lo que queremos hacer podemos realizar las siguientes instrucciones:

- $ docker-compose up: estamos arrancando en orden los contenedores definidos.
- $ docker-compose ps: podremos ver el estado de los contenedores y de cuales está compuesta la aplicación.
- $ docker-compose stop: podremos pararlos.
- $ docker-compose restart: podremos reiniciarlos.
- $ docker-compose rm: eliminamos completamente los contenedores.
- $ docker-compose logs: veremos los logs de los servicios.

2.9.5. Kitematic

Es un proyecto open source (de código abierto) creado para simplificar y racionalizar el uso de Docker en Mac o Windows. Kitematic automatiza el proceso de instalación y configuración de Docker y proporciona una interfaz intuitiva gráfica de usuario para el funcionamiento de los contenedores Docker. Kitematic se integra con Docker-machine para provisionar máquinas virtuales en VirtualBox e instalar el demonio Docker en su máquina.

Una vez instalada la interfaz gráfica de usuario, Kitematic lanza y permite desde la pantalla de inicio en la que muestra las imágenes disponibles que se puedan ejecutar al instante. Podemos buscar cualquier imagen pública sobre Docker Hub en Kitematic

Podemos utilizar la interfaz gráfica de usuario para crear, ejecutar y administrar sus propios contenedores. Kitematic también automatiza características avanzadas, tales como la gestión de puertos y configuración de volúmenes. Podemos utilizar Kitematic para cambiar las variables de entorno, registros de log, y hacer clic en un sólo terminal en su contenedor Docker, todo desde la interfaz gráfica de usuario.

2.9.6. Docker Swarm

Swarm es la forma más fácil de ejecutar un contenedor Docker en producción. Permite que una aplicación que está en desarrollo implementarlo en un clúster de servidores.
Debido a que Docker Swarm utiliza el API estándar Docker, cualquier herramienta que se comunique con el demonio Docker puede utilizar Swarm para escalar de forma transparente a múltiples host.

Algunas de las herramientas soportadas serían:

- Dokku
- Docker Compose
- Krane
- Jenkins
- Cliente Docker.

El primer paso para el uso de swarm en nuestra red es obtener la imagen de Docker Swarm. Después, usando Docker configurar el gestor de swarm y todos los nodos ejecutarán Docker Swarm, esto requiere abrir un puerto TCP en cada nodo para la comunicación con el gestor Swarm, tener Docker instalado en cada nodo y crear y administrar certificados TLS para garantizar la seguridad del clúster.

2.9.7. Docker Registry

Es un repositorio que nos permite almacenar nuestras propias imágenes. Por defecto, el registro se levanta sobre una conexión http en claro y sin autenticación.

Para registrarnos tendremos que lanzar el siguiente comando:

```
$ sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

2.10. Instalación de Docker en Ubuntu 16.04.5

Comenzaremos con la instalación de Docker en Ubuntu 16.04.5, que es la versión que tengo instalada.

1. Comenzamos añadiendo la clave GPG en nuestro equipo:

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

2. Añadimos ahora el repositorio de docker actualizamos:

```
$ sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
$ sudo apt update
```

3. Comprobamos que esta todo ok para la instalación de docker:

```
$ sudo apt-cache policy docker-engine
```

4. Procedemos a instalar docker

```
$ sudo apt install docker-engine -y
```

2.11. Expresiones de Docker

Introducción

En este apartado vamos a explicar alguna acciones típicas de docker, como arrancar un contenedor con una imagen, como crear una imagen, como subirla, etc.

Comenzaremos con descargarnos una imagen docker, concretamente debian jessie por ejemplo. El comando es muy simple:

```
root@ubuntu:/home/dani# docker pull debian:jessie
jessie: Pulling from library/debian
3d77ce4481b1: Pull complete
Digest: sha256:f29d0c98d94d6b2169c740d498091a9a8545fabfa37f2072b43a4361c10064fc
Status: Downloaded newer image for debian:jessie

root@ubuntu:/home/dani# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              jessie              4eb8376dc2a3        6 weeks ago         127MB
```

Como podemos ver hemos utilizado dos expresiones:

- **docker pull <imagen:version>**: con esto nos descargamos una imagen del repositorio docker hub.
- **Docker images**: lo utilizamos para visualizar las imágenes que tenemos.

Ahora vamos a proceder a arrancar un contenedor accediendo a el:

```
root@ubuntu:/home/dani# docker run -it --name maq1 debian:jessie /bin/bash
root@bfa783d590a9:/# ls
bin   dev  home  lib64 mnt  proc  run  srv  tmp  var
boot  etc  lib   media opt  root  sbin  sys  usr


root@bfa783d590a9:/# free
             total       used       free     shared    buffers     cached
Mem:       2018076    1859736     158340       9760     174136     588376
```

```
-/+ buffers/cache:    1097224     920852
Swap:      2094076         12    2094064


root@bfa783d590a9:/# exit
exit


root@ubuntu:/home/dani# free
              total        used        free      shared  buff/cache   available
Mem:        2018076      925292      164660        9628      928124      827552
Swap:       2094076          12     2094064
root@ubuntu:/home/dani#
```

Como vemos hemos utilizando el comando “docker run -it --name maq1 debian:jessie /bin/bash”, que hace cada elemento:

- **docker run**: nos arrancar un contenedor.
- **-i**: nos abre una sesión interactiva.
- **-t:** nos crea una terminal.
- **-- name maq1**: es el nombre que recibirá nuestro conetenedor.
- **debian:jessie**: es la imagen de la cual partimos
- **/bin/bash**: es el comando que vamos a utilizar.

Ademas pode ver que he utilizado el comando “ls” dentro del contenedor para ver su contenido, igual que el comando “free” para ver los recursos de memoria, como vemos, son igual al del host.

Ahora pasamos a como ver el estado de un contenedor, si esta o no apagado, para ello empleamos el comando “**docker ps -a**”. Si solo queremos ver los encendidos seria “docker ps”

```
root@ubuntu:/home/dani# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6a60d5124f82        debian:jessie       "/bin/bash"         26 seconds ago      Up 24 seconds                           maq1

root@ubuntu:/home/dani# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6a60d5124f82        debian:jessie       "/bin/bash"         37 seconds ago      Exited (0) 1 second ago                 maq1
```

```
root@ubuntu:/home/dani# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
root@ubuntu:/home/dani#
```

Como vemos al principio el contenedor estaba arrancado, luego hemos salido y vemos la diferencia que hay entre uno y otro, es que “docker ps” solo muestra los arrancados, mientras que los que están apagados no los muestra.

Ahora vamos a proceder a eliminar un contenedor la sintaxis es la siguiente, “docker rm <contenedor>”, como hemos visto antes, cuando hacemos un “docker ps -a” por ejemplo, vemos que hay 7 columnas, cada una muestra diferentes cosas:

- **CONTAINER ID**: muestra el id del contenedor.
- **IMAGE**: es la imagen de la cual partimos.
- **COMMAND**: es el comando con el que se arranca.
- **CREATED**: el momento que se creó.
- **STATUS**: nos muestra en el estado en el que se encuentra el contenedor.
- **PORTS**: puerto por el cual esta escuchando.
- **NAMES**: es el nombre que le hemos dado al contenedor.

Pues bien, como iba diciendo un contenedor se puede borrar bien sea con el nombre o con el id del contenedor.

Despliegue de demonio con servicio

Ahora pasamos a arrancar un servicio, vamos a utilizar nginx por ejemplo, y vamos a dejar un demonio para que el contenedor siga en pie. Para ello hay que seguir varios pasos:

Primero. Debemos de arrancar un contenedor debian en el cual vamos a instalar nginx, antes que nada debemos actualizar y luego instalar. Pero la novedad de esto es que vamos a arrancar el contenedor de una forma distinta, vamos a asignarle un puerto, para que sea accesible desde nuestro host:

```
root@ubuntu:/home/dani# docker run -it -p 81:80  --name maq1 debian:jessie /bin/bash
```

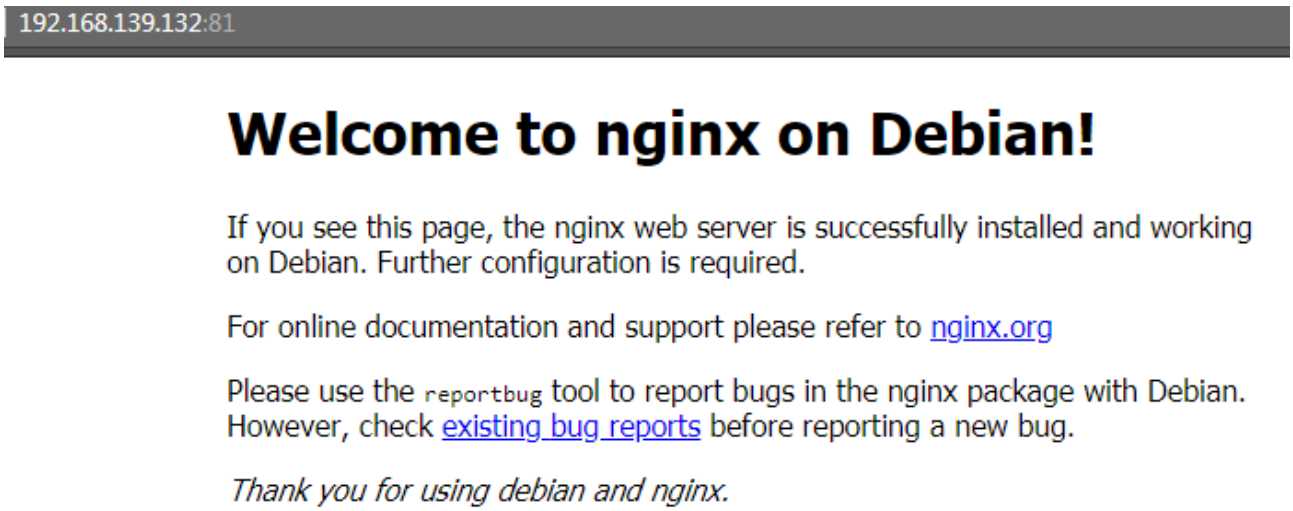
Vamos ahora a ver los procesos que existen para ver que viene en la columna “PORTS”:

```
root@ubuntu:/home/dani# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

```
PORTS               NAMES
f0f8f0691e33        debian:jessie       "/bin/bash"         27 seconds ago      Up 26 seconds
0.0.0.0:81->80/tcp  maq1
```

Como vemos el contenedor esta escuchando por el puerto y el host redireccioná lo que vaya en el puerto 81 de su host al 80 del contenedor.

Una vez instalado accedemos al al puerto 81 para verlo, podemos poner la ip del host (si queremos verlo desde otra máquina dentro de la misma red) o localhost:



Segundo. Una vez definido el contenedor debemos guardarlo y ejecutarlo con el demonio para que se inicie, antes que nada debemos crear una imagen de nuestro contenedor, para realizarlo deberemos realizar mientras el contenedor este creado.

La sintaxis es la siguiente “docker commit -m "Comentario" -a "Propietario" <contenedor> <imagen>”:

- **docker commit**: es el comando para realizar el guardado o creación de la imagen.
- **-m**: añadimos un comentario.
- **-a**: el propietario de la imagen.
- **contenedor**: es el container que hemos creado y al cual queremos crearle una imagen.
- **imagen**: es el nombre que queremos darle a la imagen.

```
root@ubuntu:/home/dani# docker commit -m "Imagen nginx" -a "Dani Bascon" maq1 nginx
sha256:08337d70ea99ff2570841f1fe991d23dd986e8a4254508f337530ac794b90745

root@ubuntu:/home/dani# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              08337d70ea99        About a minute ago   207MB
debian              jessie              4eb8376dc2a3        6 weeks ago         127MB
root@ubuntu:/home/dani#
```

Como vemos, hemos creado la imagen y se nos ha guardado. También podemos ver cuando se nos creo, el tamaño que tiene y la versión que posee:

- REPOSITORY: es nombre de la imagen.
- TAG: es la versión que posee la imagen.
- IMAGE ID: es el identificado de la imagen.
- CREATED: es cuando se creo la imagen.
- SIZE: tamaño de la imagen.

Tercero. Por último vamos ahora a crear un contenedor con el servicio nginx activo, para ello debemos de desplegarlo con un demonio y un comando que ejecute el servicio para que la máquina no se paré.

La sintaxis para lanzar el contenedor sería la siguiente “docker run -d -p 82:80 --name <nombre> <imagen> <servicio>”, esto nos inicia el contenedor con ese servicio:

- **docker run**: nos arrancar un contenedor.
- **-e:** es el demonio docker.
- **nombre**: es el nombre que recibirá nuestro conetenedor.
- **imagen**: es la imagen de la cual partimos
- **servicio**: es el servicio que lanza el contenedor y que esta activo por el demonio.

```
root@ubuntu:/home/dani# docker run -d -p 82:80  --name maq2 nginx nginx -g 'daemon off;
```

Como vemos, aunque no estemos conectado al contenedor, el contenedor sigue en pie

Creación de Dockerfile

Como ya hemos explicado anteriormente, Dockerfile es un fichero en el cual defines una imagen paso a paso, paquetes que vas a instalar, fichero que vas a incluir, volúmenes, variables….

Pues bien, vamos a definir un Dockerfile simple, en el cual levantemos simplemente el servicio de apache por ejemplo, para empezar debemos de crear el fichero y definirlo todo:

```
root@ubuntu:/home/dani# cat Dockerfile

FROM debian:jesie
MAINTAINER Daniel Bascon Arenas "bascon1991@hotmail.es"

RUN \
	apt update && \
	apt install -y apache2

EXPOSE 80
CMD apachectl -D FOREGROUND
```

Ya hemos definido el Dockerfile, ahora explicamos que significa cada línea:

- FROM: es de donde parte la imagen.
- MAINTAINER: es quien mantiene el contenedor.
- RUN: como definimos es la ejecución, es como si estuviésemos interactuando directamente con la terminal.
- EXPOSE: es el puerto al que esta escuchando el contenedor.
- CMD: es el comando que se ejecuta en el contenedor.

Ahora pasamos a crear la imagen, pero esta vez no lo hacemos con un commit si no con build, la sintaxis sería de la siguiente forma “docker build -t <imagen> .”, es muy importante añadir el punto a la hora de crear la imagen:

- **build**: es el comando que se utiliza para crear la imagen.
- **-t**: es par añadir el nombre de la imagen.
- **imagen**: nombre que recibirá la imagen.
- **.**: el punto es donde se encuentra el Dockerfile, en este caso en el mismo directorio.

```
root@ubuntu:/home/dani# docker build -t apache-prueba .
Sending build context to Docker daemon  2.048kB
Step 1/5 : FROM debian:jessie
 ---> 4eb8376dc2a3
Step 2/5 : MAINTAINER Daniel Bascon Arenas "bascon1991@hotmail.es"
 ---> Running in 66f4fc50fabe
 ---> 189632713fa4
Removing intermediate container 66f4fc50fabe
Step 3/5 : RUN apt update &&         apt install -y apache2
 ---> Running in 3b3a96663b4b

WARNING: apt does not have a stable CLI interface yet. Use with caution in scripts.

Get:1 http://security.debian.org jessie/updates InRelease [94.4 kB]
Ign http://deb.debian.org jessie InRelease
Get:2 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:3 http://deb.debian.org jessie Release.gpg [2434 B]
Get:4 http://deb.debian.org jessie Release [148 kB]
Get:5 http://security.debian.org jessie/updates/main amd64 Packages [624 kB]
Get:6 http://deb.debian.org jessie-updates/main amd64 Packages [23.0 kB]
Get:7 http://deb.debian.org jessie/main amd64 Packages [9064 kB]
Fetched 10.1 MB in 23s (437 kB/s)
Reading package lists...
Building dependency tree...
Reading state information…
……..
6 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

```
 ---> 268057172d5d
Removing intermediate container 3b3a96663b4b
Step 4/5 : EXPOSE 80
 ---> Running in 64db34c93cde
 ---> 58084b690822
Removing intermediate container 64db34c93cde
Step 5/5 : CMD apachectl -D FOREGROUND
 ---> Running in 6d9a1b9f2c7a
 ---> ac94831d656a
Removing intermediate container 6d9a1b9f2c7a
Successfully built ac94831d656a
Successfully tagged apache-prueba:latest
```

Como vemos no ha habido problemas a la hora de crear la imagen, ahora veamos si se ha creado:

```
root@ubuntu:/home/dani#  docker images | grep apache-prueba
apache-prueba               latest              ac94831d656a        4 minutes ago       200MB
```

Ahora vamos a lanzarla:

- docker run -d -p 81:80 --name maq1 apache-prueba

```
root@ubuntu:/home/dani#  docker run -d -p 81:80 --name maq1 apache-prueba
1cbbdb545feeaa0d657a498fb9d16b180a8c31824b63c14d59f1395bae98c22f
root@msi:/home/dani/Documentos/pruebas# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
1cbbdb545fee        apache-prueba       "/bin/sh -c 'apach..."   6 seconds ago       Up 3 seconds
```

Como vemos tanto el contenedor como el servicio apache esta funcionando plenamente.

Creación de contenedor con variables

Las variables son muy útiles para gestionar contenedores, en mi caso, todo el proyecto lleva variables que van cambiando en función del usuario.

Vamos a definir ahora un Dockerfile el cual tenga variables y luego lo levantaremos pasándole las variables:

```
root@ubuntu:/home/dani# cat Dockerfile

FROM debian
MAINTAINER Daniel Bascon Arenas "bascon1991@hotmail.es"

RUN \
	apt-get update && \
	apt-get install -y nginx && \
	apt-get clean && \
	rm -rf /var/lib/apt/lists/* && \
	rm -r /etc/nginx/sites-available/default /etc/nginx/sites-enabled/default

ADD index.html /home/index.html
ADD run.sh /home/run.sh
RUN chmod +x /home/run.sh


EXPOSE 80
CMD ["/home/run.sh"]
```

Como vemos partimos de la imagen que creamos anteriormente, cargamos las variables necesarios para arrancar el servicio apache2, de ahí, todas esas variables “ENV”.

Lo siguiente es añadir el fichero ejecutable y darle los permisos de ejecución:

- ADD: crea el fichero en la ruta y con el nombre especifico.
- RUN: ejecuta el comando.

Por último ejecutamos el script run.sh:

```
root@ubuntu:/home/dani# cat run.sh

#!/bin/bash

set -e

NGINX_SERVER_NAME=${NGINX_SERVER_NAME:-""}
NGINX_DOCUMENTROOT=${NGINX_DOCUMENTROOT:-""}

if [[ $NGINX_DOCUMENTROOT = "" ]] ;then
        NGINX_DOCUMENTROOT="/var/www/html"
fi



echo "server {
        listen 80;
        listen [::]:80;

        root $NGINX_DOCUMENTROOT;
        index index.html index.php;
        server_name $NGINX_SERVER_NAME;
}" > /etc/nginx/sites-available/default


ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/

if [[ ! -d $NGINX_DOCUMENTROOT ]] ;then
        mkdir -p $NGINX_DOCUMENTROOT
fi

cp /home/index.html $NGINX_DOCUMENTROOT
chown -R www-data:www-data $NGINX_DOCUMENTROOT

exec /usr/sbin/nginx -g 'daemon off;'
```

Ahora volvamos a construir la imagen, pero esta vez vamos a añadirle una versión:

- docker build -t apache-prueba:2 .

```
root@ubuntu:/home/dani# docker build -t nginx-prueba:2 .
Sending build context to Docker daemon  6.144kB
Step 1/8 : FROM debian
```

```
 ---> 8626492fecd3
Step 2/8 : MAINTAINER Daniel Bascon Arenas "bascon1991@hotmail.es"
 ---> Using cache
 ---> 8d1eb98d85c4
Step 3/8 : RUN apt-get update &&         apt-get install -y nginx &&         apt-get clean &&         rm
-rf /var/lib/apt/lists/* &&         rm -r /etc/nginx/sites-available/default /etc/nginx/sites-
enabled/default
 ---> Using cache
 ---> a37aef195411
Step 4/8 : ADD index.html /home/index.html
 ---> f5508ed65b9c
Removing intermediate container 111b755d8c7b
Step 5/8 : ADD run.sh /home/run.sh
 ---> b58096922b5b
Removing intermediate container 6c8cb48f1694
Step 6/8 : RUN chmod +x /home/run.sh
 ---> Running in f3866d545cbd
 ---> 82a355dd7b17
Removing intermediate container f3866d545cbd
Step 7/8 : EXPOSE 80
 ---> Running in e21c4d1ea623
 ---> 00a485078f18
Removing intermediate container e21c4d1ea623
Step 8/8 : CMD /home/run.sh
 ---> Running in d60fa2d5604d
 ---> 871468a010ce
Removing intermediate container d60fa2d5604d
Successfully built 871468a010ce
Successfully tagged nginx-prueba:2
```

Arrancamos ahora el contenedor definiendo la variable en el comando:

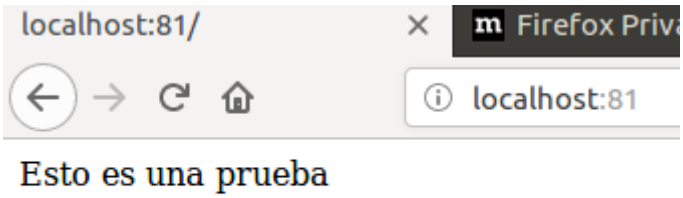
- docker run -d -p 81:80 --env NGINX_SERVER_NAME=www-bascon-nginx.org --env NGINX_DOCUMENTROOT=/srv/dani --name maq1 nginx-prueba:2

```
root@ubuntu:/home/dani/# docker run -d -p 81:80 --env NGINX_SERVER_NAME=www-
bascon-nginx.org --env NGINX_DOCUMENTROOT=/srv/dani --name maq1 nginx-prueba:2
7bdaa6ca5c421e0b5bcb053c26d2b4247ab4d3514c2401076ce52a9de661b539

root@ubuntu:/home/dani/# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

```
7bdaa6ca5c42        nginx-prueba:2      "/home/run.sh"      5 seconds ago       Up 3 seconds
0.0.0.0:81->80/tcp   maq1
```

Como vemos se ha iniciado sin problemas el contenedor:



Creación de contenedor con volumen

Este caso vamos a crear un contenedor con un volumen, la cuestión es como actuá el volumen de un contenedor.

Bien, el volumen es un lugar de almacenamiento donde definimos en el Dockerfile, debemos de indicar que directorio va a ser añadido al volumen y posteriormente, cuando arranquemos el contenedor, indicarlo nuevamente, además incluyendo con se llamará el volumen.

La sintaxis es la siguiente “docker run -it --name <nombre> -v <nombre_vol>:<ruta> <imagen>:

- **-v**: indicamos que vamos a añadir un volumen.
- **nombre_vol**: el nombre que recibirá el nombre si existe, si no existe lo creará
  - Si queremos crear un volumen “docker volume create <nombre_vol>”.
- <ruta>: es la ruta que tomará, es decir, el directorio definido se incluirá en volumen

Vamos a hacer la prueba con dos contenedores asociados a un volumen:

```
root@ubuntu:/home/dani# docker run -it --name maq1 -v vol:/prueba debian:jessie /bin/bash
root@3fd9eed7447b:/# touch prueba/fichero1
root@3fd9eed7447b:/#


root@ubuntu:/home/dani# docker run -it --name maq2 -v vol:/prueba debian:jessie ls /prueba
fichero1
```

Como vemos, hemos creado un fichero dentro de un volumen asociado a dos contenedores, así que todo lo que se cree dentro de ese directorio se quedará guardado aunque borremos el contenedor, el volumen mientras que no se borre no perderemos la información.

Creación de contenedor con link

Un link es simplemente una unión entre contenedores, para que puedan ver entre ellos, la sintaxis en sencilla “docker run -it --name <contenedor2> --link <contenedor1>:<contenedor1> <imagen> <comando>”:

- **--link <contenedor1>:<contenedor1>** : nos creará un enlace entre el contenedor actual con el otro contenedor

```
root@ubuntu:/home/dani# docker run -it --name maq1 debian:jessie  /bin/bash
root@3fd9eed7447b:/#

root@ubuntu:/home/dani# docker run -it --name maq2  debian:jessie /bin/bash
root@42b0f5525146:/# ping maq1
ping: unknown host maq1
root@42b0f5525146:/# exit

root@ubuntu:/home/ubuntu# docker run -it --name maq3  --link maq1:maq1 debian:jessie
/bin/bash
root@768bcd8e56cb:/# ping maq1
PING maq1 (172.17.0.2) 56(84) bytes of data.
64 bytes from maq1 (172.17.0.2): icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from maq1 (172.17.0.2): icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from maq1 (172.17.0.2): icmp_seq=3 ttl=64 time=0.048 ms
```

Como vemos la maq3 si ha tenido conexión con la maq1, mientras que la maquina 2, no ha tenido, y es por el link.

Ya hemos dado un pequeño repaso a lo más esencial de docker, pero hay muchos más, algunos de ellos son:

- docker logs <contenedor>: muestra los logs de un contenedor.
- docker attach <contenedor>: nos conectamos directamente a un contenedor.
- Docker exec -it <contenedor> <comando>: nos conectamos a un contenedor, inyectándole un comando. Este ya lo hemos explicado.
- COPY: a diferencia de ADD, este copia el fichero a una ruta, no hace falta añadirlo.
- Diff: diferencia entre contenedores.

- History, nos muestra el historial de la imagen.
- Inspect: nos inspecciona el contenido de un contenedor.
- push: nos sube una imagen a docker hub, primero hay que hacer un login.
- tag: etiqueta una imagen.

Docker-Compose

Docker-compose es una herramienta de docker la cual nos permite lanzar varios contenedores definidos en un fichero yaml de forma ordenada.

Para instalar docker-compose tenemos que hacerlo de la siguiente forma:

- curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
- chmod +x /usr/local/bin/docker-compose

Vamos arrancar los siguientes contenedores:

- docker run -d --name servidor danibascon/proftpd
- docker run -d -p 21:21 -p 22:22 -p 81:80 --name servidor_apache --link servidor:servidor danibascon/apache2:1

Una vez definido como lineas de comando los contenedores que vamos a crear, ahora vamos hacerlo en el docker-compose

- nano docker-compose.yml

```
servidor:
  image: danibascon/proftpd
  environment:
    USER: usuario
    PASS: contra
  volumes:
   - bookmedik:/var/www/html/usuario

servidor_apache:
  image:  danibascon/apache2:1
  links:
    - servidor
  ports:
    - "81:80"
    - "21:21"
    - "22:22"
```

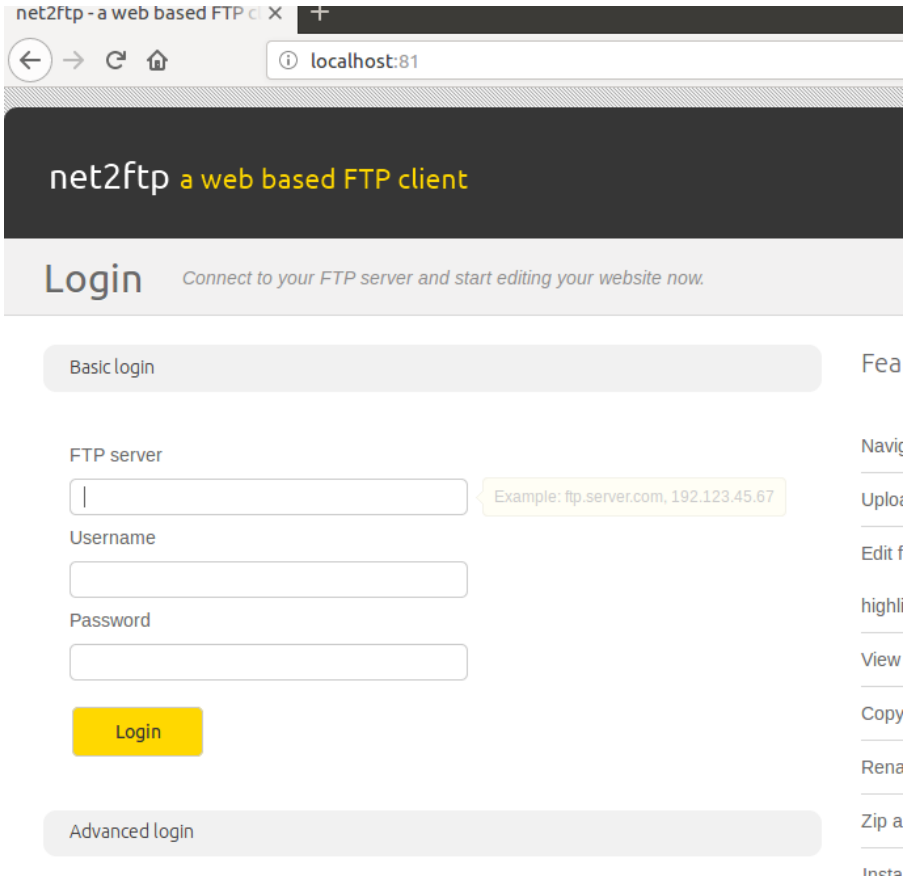
Procedemos ahora a levantar el servicio:

- docker-compose up -d

```
root@ubuntu:/home/dani# nano docker-compose.yml
Creating pruebas_servidor_1        ... done
Creating pruebas_servidor_1        ...
Creating pruebas_servidor_apache_1 ... done

root@ubuntu:/home/dani/# docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
PORTS                                  NAMES
3aad17e239da        danibascon/apache2:1   "/bin/sh -c 'apach..."   2 minutes ago       Up 2 minutes
0.0.0.0:21-22->21-22/tcp, 0.0.0.0:81->80/tcp   pruebas_servidor_apache_1
834d9cb40665        danibascon/proftpd     "/bin/sh -c /cmd.sh"     2 minutes ago       Up 2 minutes
21-22/tcp                              pruebas_servidor_1
```

Como vemos el servicio se ha iniciado sin problema y estado todo ok.



Una vez estando todo levantado estaría bien, parar o eliminar los contenedores, en este caso, son 2, pero y si fuese más de 10, sería una perdida de tiempo, hacerlo uno a uno, por lo que docker-compose tiene la función de real izarlo todos a la vez, igual que al crearlos.

Con la opción kill o stop, los paramos y con rm los eliminamos:

```
root@ubuntu:/home/dani/# docker-compose kill
Killing pruebas_servidor_apache_1 ... done
Killing pruebas_servidor_1        ... done


root@ubuntu:/home/dani/# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED           STATUS
PORTS               NAMES
3aad17e239da        danibascon/apache2:1   "/bin/sh -c 'apach..."   6 minutes ago       Exited (137) 3
seconds ago                 pruebas_servidor_apache_1
834d9cb40665        danibascon/proftpd     "/bin/sh -c /cmd.sh"     6 minutes ago       Exited (137) 3
seconds ago                 pruebas_servidor_1


root@ubuntu:/home/dani/# docker-compose rm
Going to remove pruebas_servidor_apache_1, pruebas_servidor_1
Are you sure? [yN] y
Removing pruebas_servidor_apache_1 ... done
Removing pruebas_servidor_1        ... done


root@ubuntu:/home/dani/# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED           STATUS
PORTS               NAMES
```

2.12. Herramientas de Hostcker

En este apartado vamos a definir los paquetes que utilizado para mi aplicación, los iré mencionado.

- python.
- apache2
- php5 y php7.0
- mysql.
- debian jessie, debian stretch.
- ubuntu 16.06.
- bottle.
- proftpd.
- phpmyadmin.
- net2ftp.
- docker.
- docker-compose.

Estas son las herramientas son las que he empleado para mi script.

2.13. Funcionamiento de HOSTCKER

Comenzaremos explicando la creación de las imágenes que he creado, que es lo que hacen con sus respectivos cmd.sh y como se lanzan.

Imagen danibascon/proftpd

Esta imagen nos creará un contenedor con el servicio proftpd activo, el cual nos creará los usuarios necesarios para acceder y añadir ficheros por ftp.

Dockerfile:

```
FROM proftpd

#Instalamos las depence
RUN apt install nano python -y
```

```
#Añadimos los ficheros necesarios
ADD proftpd.conf /etc/proftpd/proftpd.conf
ADD cmd.sh /cmd.sh
ADD usuario.py /usuario.py

#Añadimos el permiso de ejecución
RUN chmod +x /cmd.sh

#Añadimos el volumen al directorio
VOLUME /var/www/html

#Exponemos el contenedor a los puertos para que escuche
EXPOSE 21
EXPOSE 22

#Ejecutamos el script
ENTRYPOINT /cmd.sh
```

Aquí vamos a cargar una imagen que cree anteriormente, el motivo es que para la instalación del paquete proftpd, tenía que elegir unas opciones durante su proceso de instalación por que no puedo utilizarlo en el build, por lo que me cree una imagen con un debian jessie, realice un update, que es lo primero que debemos de hacer, ya que si no, no nos deja instalar ningún paquete.

Continuamos explicando el Dockerfile, con la imagen ya cargado de proftpd, instalo 2 paquetes, python para lanzar luego un pequeño script que he creado y nano, el cual no es necesario, pero lo manejo mejor que el editor de texto vim.

Posteriormente añado 2 script, uno en python y otro en bash, los cuales explicaré posteriormente, además añado un fichero de configuración para el paquete proftpd, abajo añado las líneas más importantes, que me nos sirven:

```
DefaultRoot     /var/www/html/%u
 <Anonymous /var/www/html>
```

Estas líneas que hemos modificado el original, quieren decir, que el servidor ftp apunta al directorio “/var/www/html/” y que cada usuario accederá con sus debidas credenciales a su respectivo directorio “/var/www/html/%u”, el “%u” nos permite hacerlo con todos los usuarios, si no, tendríamos que añadirlos uno a uno.

Continuamos con el Dockerfile, el siguiente comando empleado es para el cambio de permiso del fichero cmd.sh, para que se puede ejecutar.

Luego añadimos los 2 puertos para el servicio ftp, 21 y 22, los cuales estarán abierto cuando arranquemos el contenedor, escuchando todo lo que vaya dirigidos a ellos.

Posteriormente vamos a indicar que el directorio “/var/www/html estará enlazado a un volumen que añadiremos, con esto todo lo que se crea en este directorio, aunque el contenedor se elimine, la información quedará en el volumen que le hemos añadido.

Por ultimo ejecutamos el entrypoint primero y el cmd, que nos más que el inicio del demonio, para que el contenedor este levantado en todo momento con el servicio proftpd activo.

Ahora pasamos a explicar el entrypoint cmd.sh:

```
#!/bin/bash
set -e

USER=${USER:-""}
PASS=${PASS:-""}

if [[ $USER != "" ]] ;then
        python /usuario.py $USER $PASS
fi
exec proftpd --nodaemon
```

Este script como vemos, le estan llegando 2 variables USER y PASS, en el caso de que USER sea distinto de vacio, ejecutará nuestro script en python el cual añadiremos las 2 variables. En caso de vacío, no pasará nada.

Ahora pasamos al script de python, usuario.py, que nos crea los usuarios:

```
from sys import argv
import crypt
import commands

for i in range(len(argv[1].split(";"))):
        contra = crypt.crypt(argv[2].split(";")[i],"salt")
        commands.getoutput("if [ ! -d /var/www/html/" + argv[1].split(";")[i] + " ] ;then mkdir
/var/www/html/" + argv[1].split(";")[i] + " ;fi")
        commands.getoutput("useradd " + argv[1].split(";")[i] + " -p " + contra + "  -d
/var/www/html/" + argv[1].split(";")[i] + " -s /bin/bash")
        commands.getoutput("chown " + argv[1].split(";")[i] + ":" + argv[1].split(";")[i] + "
/var/www/html/"+ argv[1].split(";")[i])
```

Comenzamos importando argv para que nos permita inyectarle variables desde afuera, las variables son USER y PASS, que mencionamos anteriormente, la librería crypt para encryptar las contraseñas y la librería commands, para ejecutar comandos propios de bash.

Como vemos, hay un for y un split, el porque de esto, es porque las variables vienen de la siguiente forma:

- USER= user1;user2;user3
- PASS = pass1;pass2;pass3

Una vez explicado esto, ya se entiende mejor el script, el script en si hace lo siguiente:

- Nos separá las variables y las recorre tantas veces como usuarios haya, separado por el ";", en este caso, 3 veces, por lo que nos creará 3 usuarios.
- Primero nos crea una contraseña encriptada.
- Luego nos comprueba si ese usuario tiene carpeta, si no existe un usuario con esa carpeta, nos las crea la carpeta.
- Posteriormente nos crea el usuario, el cual tendra una contraseña y un directorio en "/var/www/html".
- Por último, cambios los permisos a la carpeta por precaución.

La última línea arrancará el servicio proftpd "exec proftpd --nodaemon" y listo.

Imagen danibascon/apache2:1

Esta imagen nos dará acceso a servidor web de casa usuario, todo lo que vayamos subiendo por ftp, se nos mostrará aquí.

Dockerfile:

```
FROM debian:jessie

# Install dependencias.
RUN apt update
RUN apt install  apache2 libapache2-mod-php5 nano python php5 -y

#Añadimos el volumen al directorio
VOLUME /var/wwww/html

#Añadimos los ficheros y creamos las carpetas en las rutas indicadas
RUN mkdir /var/www/html/net2ftp
ADD net2ftp /var/www/html/net2ftp
```

```
ADD www.conf /etc/apache2/sites-available/

#Cambiamos los permisos al directirio
RUN chmod 0777 /var/www/html/net2ftp/files_to_upload/temp

#Activamos el sitio que hemos creado y desactivamos el que viene por defecto
RUN a2ensite www.conf && a2dissite 000-default.conf

CMD apachectl -D FOREGROUND
```

Esta imagen parte de un debian jessie, el cual vamos a instalar ese conjunto de paquete, para poder utilizar el servicio web de nft, net2ftp, con el que podremos interactuar con nuestro servidor ftp.

A continuación añadimos el fichero de configuración para que podamos al directorio net2ftp, directorio que crearemos antes de añadir los ficheros y cambiamos los permisos.

Por último arrancamos el dominio de apache directamente desde el Dockerfile, como estamos viendo con “CMD apachectl -D FOREGROUND”.

Imagen danibascon/mysql-ubuntu:1

Está imagen nos creará un contenedor en el cual nos creará los usuarios en la base de datos, como vemos la imagen parte de sameersbn, explicaremos al final el motivo.

Dockerfile:

```
FROM sameersbn/ubuntu:16.04.20180124

ENV MYSQL_USER=mysql \
    MYSQL_DATA_DIR=/var/lib/mysql \
    MYSQL_RUN_DIR=/run/mysqld \
    MYSQL_LOG_DIR=/var/log/mysql

RUN apt-get update \
 && DEBIAN_FRONTEND=noninteractive apt-get install -y mysql-server \
 && rm -rf ${MYSQL_DATA_DIR} \
 && rm -rf /var/lib/apt/lists/*

COPY entrypoint.sh /sbin/entrypoint.sh
```

```
RUN chmod 755 /sbin/entrypoint.sh

EXPOSE 3306/tcp
VOLUME ["${MYSQL_DATA_DIR}", "${MYSQL_RUN_DIR}"]
ENTRYPOINT ["/sbin/entrypoint.sh"]
CMD ["/usr/bin/mysqld_safe"]
```

Esta imagen parte de una ido ubuntu, en el cual definimos una series de variables e instalamos mysql-server.

Abrimos el puerto 3306 para que escuche, y los clientes se puedan conectar al servidor y añadimos la ruta de “/var/lib/mysql/” al volumen el cual conectaremos más tarde, esto se define en la variable.

Posteriormente añadimos el fichero entrypoint.sh, el permiso de ejecución y lo ejecutamos:

entrypoint.sh:

```
set -e

#Cambio de permiso y depuración de información
chown -R mysql:mysql /var/lib/mysql
mysql_install_db --user mysql > /dev/null

#Definición de las variables
MYSQL_DATABASE=${MYSQL_DATABASE:-""}
MYSQL_USER=${MYSQL_USER:-""}
MYSQL_PASSWORD=${MYSQL_PASSWORD:-""}

#Definición de fichero temporal
tfile=`mktemp`
cat << EOF > $tfile
USE mysql;
FLUSH PRIVILEGES;
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION;
UPDATE user SET password=PASSWORD('root') WHERE user='root';
EOF

#Creación de base de datos y usuario con permisos
echo "CREATE DATABASE IF NOT EXISTS \`$MYSQL_DATABASE\`;" >> $tfile
echo "GRANT ALL ON \`$MYSQL_DATABASE\`.* to '$MYSQL_USER'@'%' IDENTIFIED
BY '$MYSQL_PASSWORD';" >> $tfile

#inyectamos el fichero para ejecutarlo con el servicio
/usr/sbin/mysqld --bootstrap --verbose=0 < $tfile
```

```
rm -f $tfile

#arrancamos el servicio
exec /usr/sbin/mysqld
```

En este script creamos la base datos y le asignaremos los permisos al usuario que crearemos, por último ejecutaremos el servicio mysql con el cual arrancamos el servicio con todo creado, en caso contrario, lo inicial sin más, sin crear nada, solo cargar lo que haya en el volumen en el cual le hemos asignado.

Imagen danibascon/phpmyadmin7-ubuntu:2

Esta imagen nos ayudará a acceder a la base de datos de cada usuario que se haya registrado en la base de datos.

Dockerfile:

```
FROM danibascon/phpmyadmin7-ubuntu:1
#FROM ubuntu:16.04

#paquetes instalados:
#RUN apt install nano apache2 php7.0 libapache2-mod-php7.0  php7.0-mysql phpmyadmin unzip wget -y

ADD www.conf /etc/apache2/sites-available/www.conf
RUN ln -s /usr/share/phpmyadmin /var/www/html/
RUN a2ensite www.conf && a2dissite 000-default.conf

CMD apachectl -D FOREGROUND
```

Esta imagen parte de una imagen que he creado, he arrancado una imagen ubuntu:16.04, posteriormente he actualizado he instalado los paquetes que estan comentado, realizamos un commit y listo, el commit es la imagen principal “danibascon/phpmyadmin7-ubuntu:1”.

He tenido que realizar esta acción debido a que phpmyadmin no se instalaba solo, necesitaba realizar una serie de acciones.

Añadimos ahora un fichero de configuración especificando la ruta para el phpmyadmin, el fichero es “www.conf”.

www.conf:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
       ServerName  dit.hostcker-phpmyadmin.org
        Alias / /var/www/html/phpmyadmin/
        <Directory /var/www/html/phpmyadmin/>
                Options Indexes
#                AllowOverride None
#                Require all granted
        </Directory>

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

Este fichero de configuración nos redirecciona directamente al directorio de phpmyadmin.

config-db.php:

```
<?php

$dbuser='';
$dbpass='';
$basepath='';
$dbname='';
$dbserver='mysql';
$dbport='3306';
$dbtype='mysql';
```

Por último añadimos el fichero de configuración para el phpmyadmin, con esto nuestro contenedor apuntará al servidor mysql.

Aplicación proyecto.py

Vamos a explicar la aplicación escrita en python, utilizando el framework Bottle, vamos a ir explicando la aplicación por parte.

Librerías

Comenzamos importando las librerías que necesitamos utilizar, principalmente la de Bottle, la cual necesitamos para nuestra aplicación. De esta librería utilizaremos principalmente los route, template, get y post.

Además de importar argv, el cual nos facilitara arrancar la aplicación con variables, y la librería commands, que nos ayudará para ejecutar comandos de la terminal desde nuestra aplicación:

```
from sys import argv
import bottle
from bottle import Bottle,route,run,request,template,static_file,redirect,get,post, default_app,
response
import requests
import commands
```

Función docker_start_usuario

Vamos ahora a definir las funciones que he creado para iniciar los servicios necesarios. Vamos a subdividirlo por parte.

El primero que no encontramos es la función “docker_start_usuario”, esta función se inicia con 3 variables, “user” es el usuario con el que se inicia el contenedor, “puerto”, es el puerto que se le asigna para que nuestro host, habrá el puerto para ese contenedor específico y “var” es una variable la cual nos pasa para cargar o no el CMS.

Luego simplemente arrancamos el contenedor con la imagen correspondiente pasándole las variables correspondiente, además creamos un link con otro contenedor y le añadimos el volumen proftdp a la ruta de “/var/www/html/”.

Este servicio lo que hará es arrancar un contenedor por casa usuario que ya este creado, siendo el puerto el indicado en la base de datos y la variable como nula, y si es llamado cuando se creá un usuario, la variable “var” entra en función sí queremos o no añadir el CMS:

```
def docker_start_usuario(user,puerto, var):
  commands.getoutput("docker run -d -p " + str(puerto) + ":80 --name " + user +" --link
mysql:mysql -v proftpd:/var/www/html -e SERVER_NAME='" + user + "' -e VAR='" + var + "' -e
DOCUMENTROOT='" + user + "' danibascon/apache2-usuario:7")

  return
```

Función docker_stop_servidor

Continuamos ahora con la función “docker_stop_servidor”, esta función nos para todos los servicios que hemos arrancado hasta el momento para iniciarlos con nuevas variables, el motivo de esto, es que no he encontrado otra solución a la hora de enviarles ciertos comandos al contenedor, por lo que he tenido que recurrir a esta metodología tan ruda:

```
def docker_stop_servidor():
  commands.getoutput("docker stop servidor servidor_apache mysql phpmyadmin; docker rm servidor servidor_apache mysql phpmyadmin")

  return
```

Función docker_start_mysql

Ahora pasamos a la función “docker_start_mysql”, esta función nos crea un usuario y una base de datos para el usuario en concreto, además nos arrancará otro contenedor, el de phpmyadmin al puerto 82 de nuestro host, con un link al servidor de mysql.

En servidor mysql se inicia con un conjunto de variables, el usuario root, su contraseña, y bind_address, que es como se define como “MYSQL_ROOT_HOST”, lo dejamos en “0.0.0.0” para que nuestro phpmyadmin nos deje conectarnos.

Ahora pasamos al if, en función del valor que posee la variable “usuario”, arrancará el servidor msyql de una forma u otra, pero es simple, si el usuario es igual a vació, nos los inicia normal, carga todo lo que haya en ese momento en la base de datos, mientras que si la variable es distinta de vacía, se creará el usuario con una contraseña y una base de datos:

```
def docker_start_mysql(usuario,contra):
  if usuario =="":
    commands.getoutput("docker run --name mysql -e DB_REMOTE_ROOT_NAME==root -e DB_REMOTE_ROOT_PASS=root -e MYSQL_ROOT_HOST=0.0.0.0 -d -v mysql:/var/lib/mysql danibascon/mysql-ubuntu:1")

  else:
    commands.getoutput("docker run --name mysql -e DB_REMOTE_ROOT_NAME==root -e DB_REMOTE_ROOT_PASS=root -e DB_USER=" + usuario + " -e DB_NAME=" + usuario + " -e DB_PASS=" + contra + " -e MYSQL_ROOT_HOST=0.0.0.0 -d -v mysql:/var/lib/mysql danibascon/mysql-ubuntu:1")

  commands.getoutput("docker run -d -p 82:80 --name phpmyadmin --link mysql:mysql danibascon/phpmyadmin7-ubuntu:2")
  return
```

Función docker_start_servidor

Esta función nos arranca el servidor ftp y el net2ftp, que es un cliente ftp que interactúa con el servidor ftp.

Comenzamos definiendo 4 variables, por defecto las variables usuario y contra son vacíos, esto es debido a que sí la variable “num” es igual a cero, que nos arranque los 2 contenedores sin variables, es decir, que sí la variable “num” que nos representa cuantos usuarios hay en nuestra base de datos es cero, el servidor ftp, no tendrá registrado ningún usuario.

También cabe destacar si la variable “num” es cero, la variable “variables” cera vació, ya que prácticamente es parecida, salvo que recoge información de los usuarios registrados en ese momento, el motivo es, que sí arrancamos la aplicación de nuevo o se registre un usuario, no haya problemas.

En el caso de existir algún usuario, es decir, “num” es distinto de cero, nos arrancar el servicio “docker_start_usuario”, que hemos definido antes, y nos lo arrancará el servicio tantas veces como usuario haya, así tendremos un contenedor para cada usuario:

```
def docker_start_servidor():
  usuario =""
  contra=""
  num=int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from usuarios;'").split("\n")[2])
  variables = commands.getoutput("mysql -u dani -pdani proyecto -e 'select usuario,contra,puerto from usuarios'").split("\n")[2:]


  if num!=0:
    var=""
    for i in range(len(variables)):
      docker_start_usuario(variables[i].split("\t")[0],variables[i].split("\t")[2],var)
      usuario = usuario + variables[i].split("\t")[0] + ";"
      contra = contra + variables[i].split("\t")[1] + ";"

    commands.getoutput("docker run -d --name servidor -e USER='" + usuario[:-1] + "' -e PASS='" + contra[:-1] + "' -v proftpd:/var/www/html danibascon/proftpd")

  else:
    commands.getoutput("docker run -d --name servidor -e USER='" + usuario + "' -e PASS='" + contra + "' -v proftpd:/var/www/html danibascon/proftpd")



  commands.getoutput("docker run -d -p 21:21 -p 22:22 -p 81:80 --name servidor_apache --link servidor:servidor danibascon/apache2:1")
  return

def docker_stop_usuario(user):
```

```
  commands.getoutput("docker stop " + user + ";docker rm " + user)

  return
```

Arranque de las funciones

Las siguientes dos líneas nos arrancará los contenedores, como vemos el de mysql lo hará definiendo la variable usuario y contra como vacías, porque solo queremos iniciar el servicio:

```
docker_start_mysql(usuario ="",contra="")
docker_start_servidor()
```

Route “/”

Este route es el inicial, nos carga el primer template, que nos redirecciona a la pantalla principal:

```
@route('/', method = "get")
def inicio():
        return template('inicio.tpl',variable='')
```

Route “/login”

Este route es que nos realizará el login y en función de las credenciales que aportemos, nos redireccionará a una ventana u otra:

```
@route('/login', method = "post")
def login():
  user = request.forms.get('user')
  passwd = request.forms.get('passwd')
  num = int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from usuarios where usuario = "+'"'+user+'";'+"'").split("\n")[2])

  if num == 0:
    var = 'El usuario no existe'
  else:
    if passwd != commands.getoutput("mysql -u dani -pdani proyecto -e 'select contra from usuarios where usuario = "+'"'+user+'";'+"'").split("\n")[2]:
      var = 'La clave no es correcta'
```

```
    else:
      return template('correcto.tpl', variable = user)

  return template('inicio.tpl', variable = var)
```

Route “/register”

Este route funciona para dar de alta a un usuario:

```
@route('/register', method = "get")
def register():
  return template('register.tpl', variable = '')
```

Route “/registro”

Este route nos creará el usuario, con su respectiva contraseña, si surge algún error o simplemente el usuario el cual que queremos registrar ya esta registrado, nos los dirá.

En caso de poder registrar el nuevo usuario, nos enviará un correo con la información y con las características de su Host.

Por último paramos los servicios y los volvemos a arrancar, como todo esta en volumen aislados de los contenedores, la información es persistente en todo momento:

```
@route('/registro',method = "post")
def registro():
  user = request.forms.get('user')
  passwd = request.forms.get('passwd')
  nombre = request.forms.get('nombre')
  apellido = request.forms.get('apellido')
  email = request.forms.get('email')
  wordpress = request.forms.get('wordpress')

  if int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from
usuarios;'").split("\n")[2]) == 0:
    puerto = 83
  else:
    puerto = int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select puerto from usuarios
order by puerto desc;'").split("\n")[2]) + 1

  if int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from usuarios
where usuario = "+'""'+user+'";'+"'").split("\n")[2]) != 0:
    var = 'Ese usuario ya esta registrado'
    return template('register.tpl', variable = var)
```

```
  else:
    insert = "insert into usuarios values
('"+user+"','"+passwd+"','"+nombre+"','"+apellido+"','"+email+"','"+str(puerto)+"');"
    commands.getoutput("mysql -u dani -pdani proyecto -e "+""+insert+"")

    if int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from
usuarios where usuario = "+""+user+"';'+"").split("\n")[2]) == 0:
      var = "Ha ocurrido un problema a la hora de registar el usuario '"+user+""
      return template('register.tpl', variable = var)

    else:
      docker_stop_servidor()
      docker_start_mysql(user,passwd)
      docker_start_servidor()
      docker_start_usuario(user,puerto,wordpress)
      return template('registrado.tpl', variable = user)
```

Route “/baja”

Este route nos redireccionará al template para dar de baja a un usuario específico:

```
@route('/baja', method = 'get')
def baja():
  return template('baja.tpl', variable = '')
```

Route “/darbaja”

Este route nos dará de baja a un usuario tanto en la base de datos como en el servidor ftp, para que no haya información de este usuario.

Antes que nada comprueba si existe o no ese usuario en la base de datos:

```
@route('/darbaja', method = 'post')
def darbaja():
  user = request.forms.get('user')
  passwd = request.forms.get('passwd')
  email = request.forms.get('email')
  if int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from usuarios
where usuario = "+""+user+"';'+"").split("\n")[2]) == 1:
    delete = "delete from usuarios where usuario = '"+user+""
    commands.getoutput("mysql -u dani -pdani proyecto -e "+""+delete+"")
    var="El usuario '"+user+"' ha sido eliminado satisfactoriamente"
    docker_stop_usuario(user)
    return template('bajacuenta.tpl', variable=var)
```

```
    if int(commands.getoutput("mysql -u dani -pdani proyecto -e 'select count(usuario) from
usuarios where usuario = "+'"'+user+'";'+'"').split("\n")[2]) == 1:
      var = "Ha ocurrido un problema a la hora de dar baja al usuario "'+user+'"'
  else:
    var = 'Ese usuario no existe'
  return template('baja.tpl', variable=var)
```

Route “/static/”

Este route nos cargar la carpeta static, donde están todos ficheros estáticos:

```
@route('/static/<filepath:path>')
def server_static(filepath):
  return static_file(filepath, root='static')
```

Por último esta línea lo que nos permite arrancar la aplicación en un determinado puerto, el cual se lo pasaremos cuando ejecutemos el script:

```
run(host='0.0.0.0',port=argv[1])
```

Para ejecuta el script lo haremos de la siguiente forma:

- Debemos ejecutarlo como root, porque hay comandos que deben ejecutarse como superusuario.
- 8080 es el puerto al cual debemos acceder a través del navegador.

```
$ sudo python proyecto.py 8080
```

3. Pruebas y resultados.

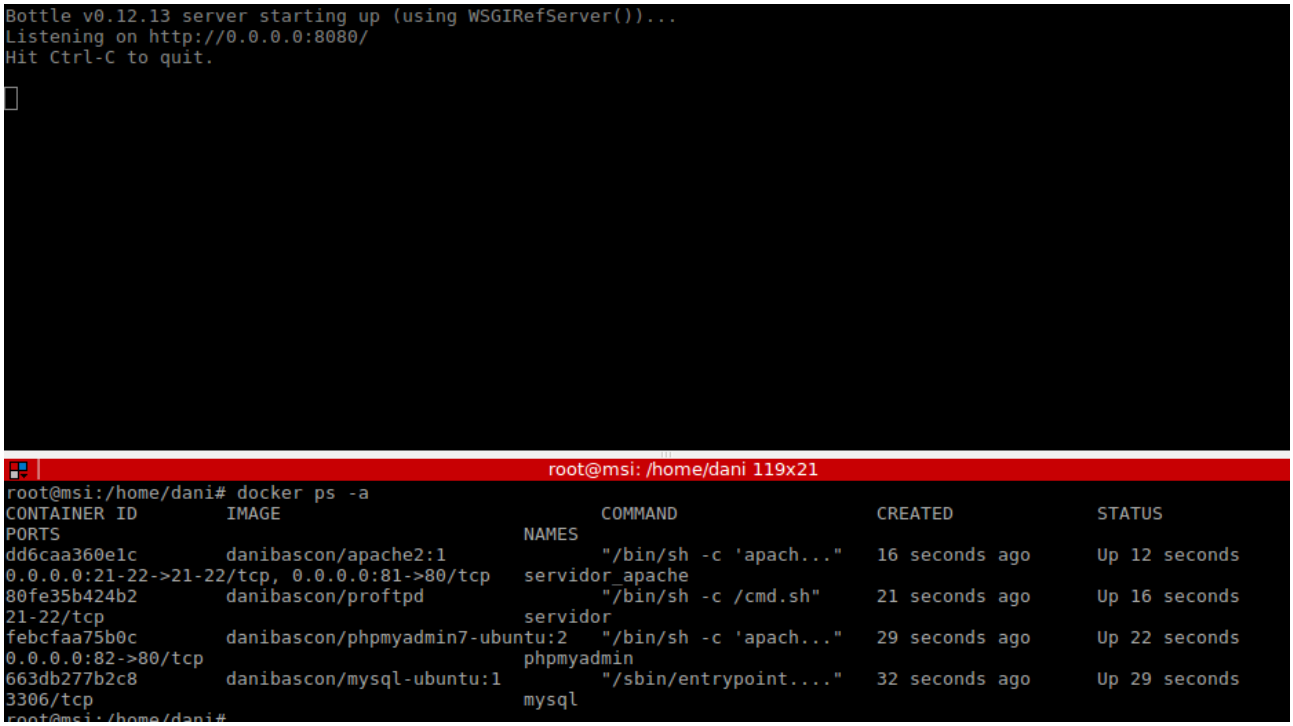
Conforme a la idea de proyecto inicial, he conseguido varias de mis objetivos, una aplicación la cual lancé contenedores para cliente, los cuales se registren en la aplicación. Que tengan un servidor ftp, donde los usuarios, a través de una web pueda subir ficheros o un cms para instalarlo.

También añadir una base de datos con el que el cliente podrá interactuar, mediante phpmyadmin, todo esto y además con un espacio para el cliente.

Todo esto es lo que hace mi aplicación, los objetivos que no he conseguido son los de desplegarlo con kubernetes y añadir un DNS bien operativo, pues el que tengo funciona, pero no como me gustaría.

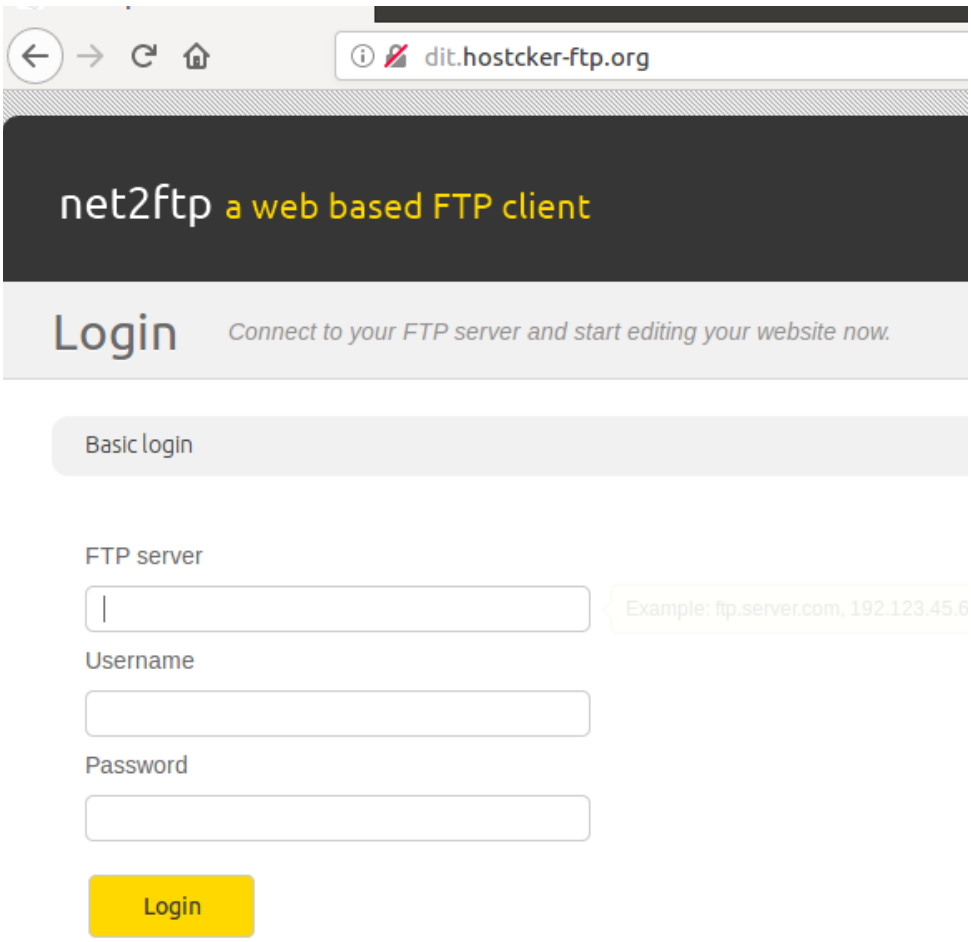
Comenzamos las pruebas.

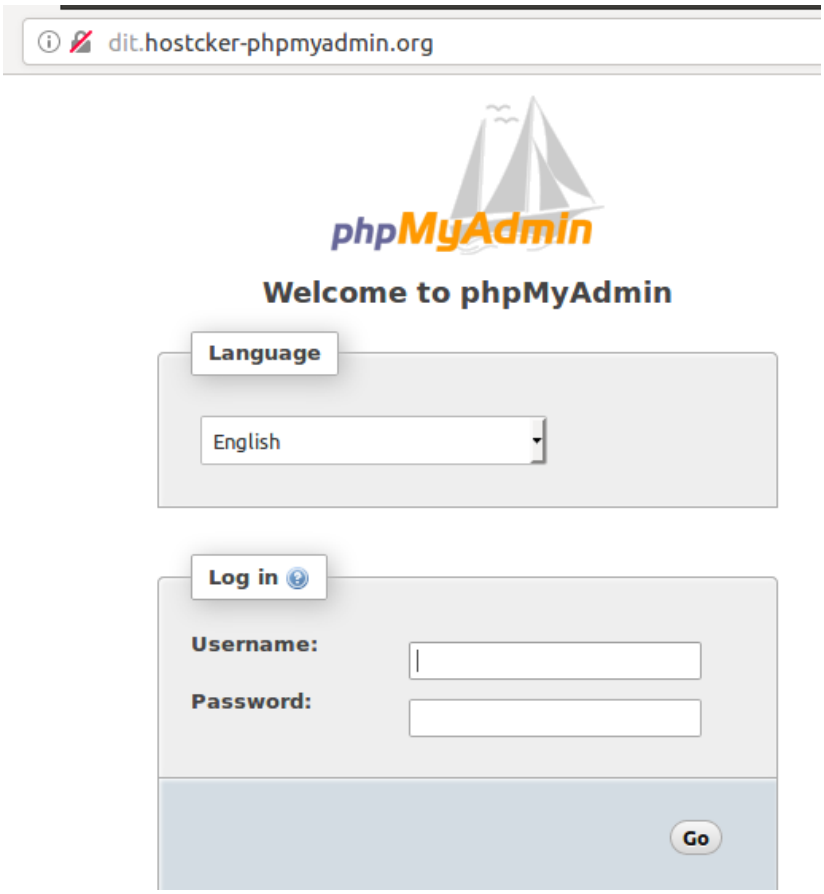
Primero arrancamos la aplicación en el puerto 8080:



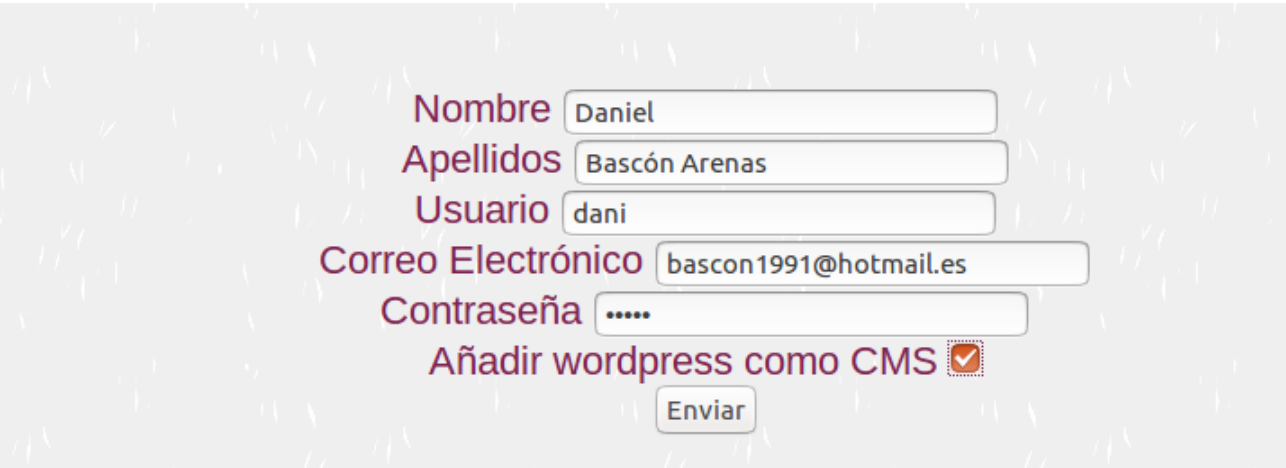
Como vemos se han iniciado bien los contenedores, vamos a probar que efectivamente podemos acceder a las 3 páginas:


Hostcker
localhost:8080
Bienvenidos a Hostcker
BIENVENIDOS A HOSTCKER
Usuario
Contraseña
Enviar

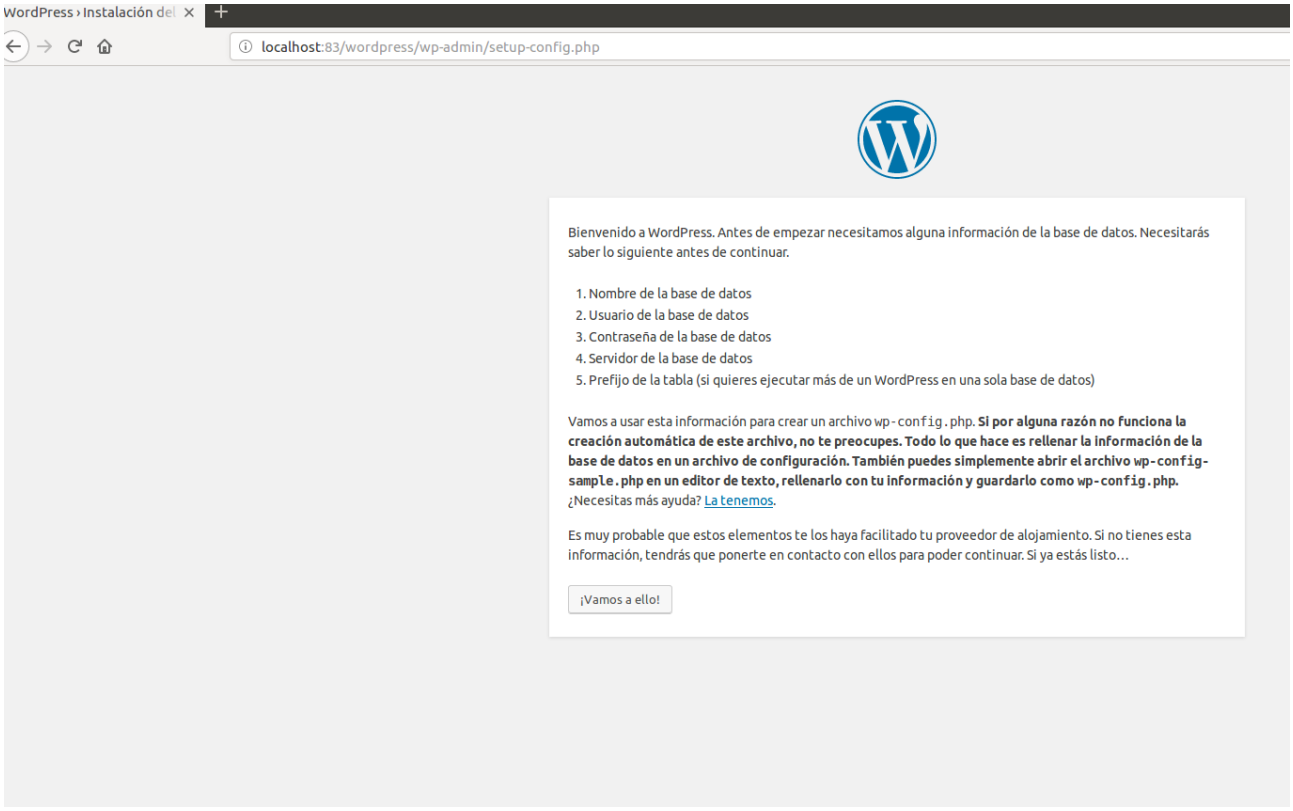

dit.hostcker-ftp.org
net2ftp a web based FTP client
Login
Connect to your FTP server and start editing your website now.
Basic login
FTP server
Example: ftp.server.com, 192.123.45.6
Username
Password
Login

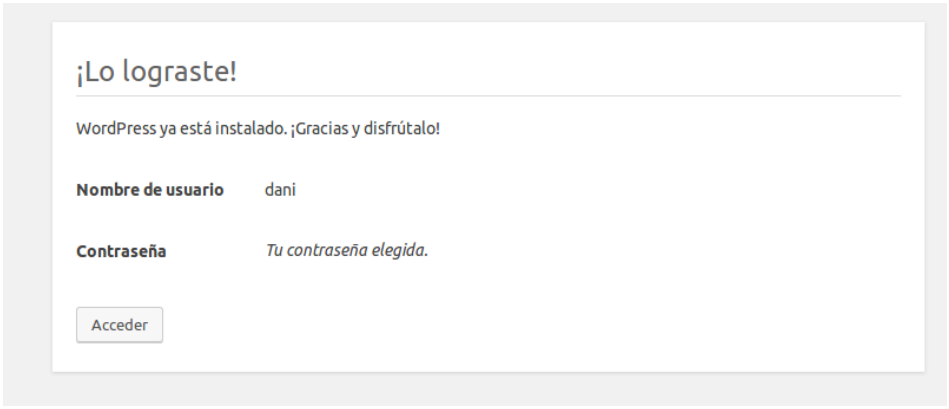


Como vemos tenemos las 3 páginas funcionando, ahora vamos a registrarnos:

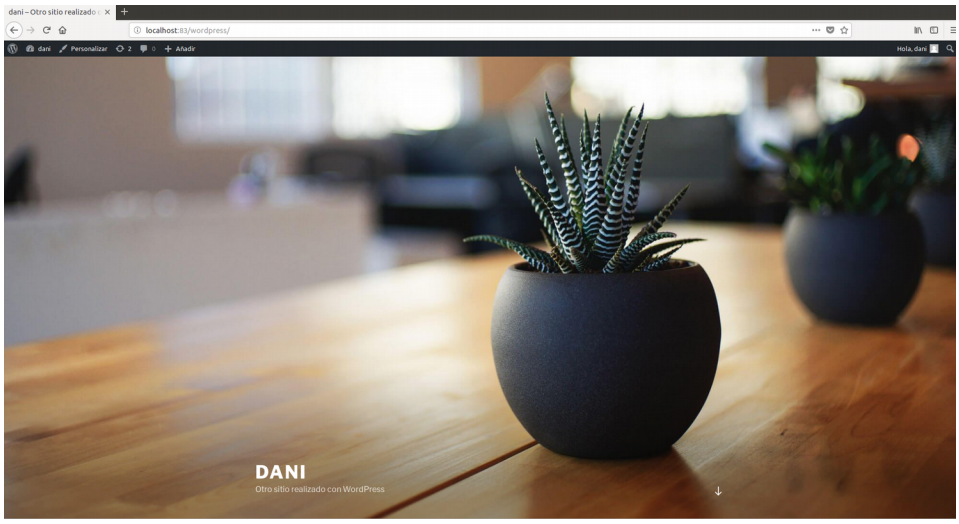


Accedemos a la url, a la dirección de wordpress para instalarla:



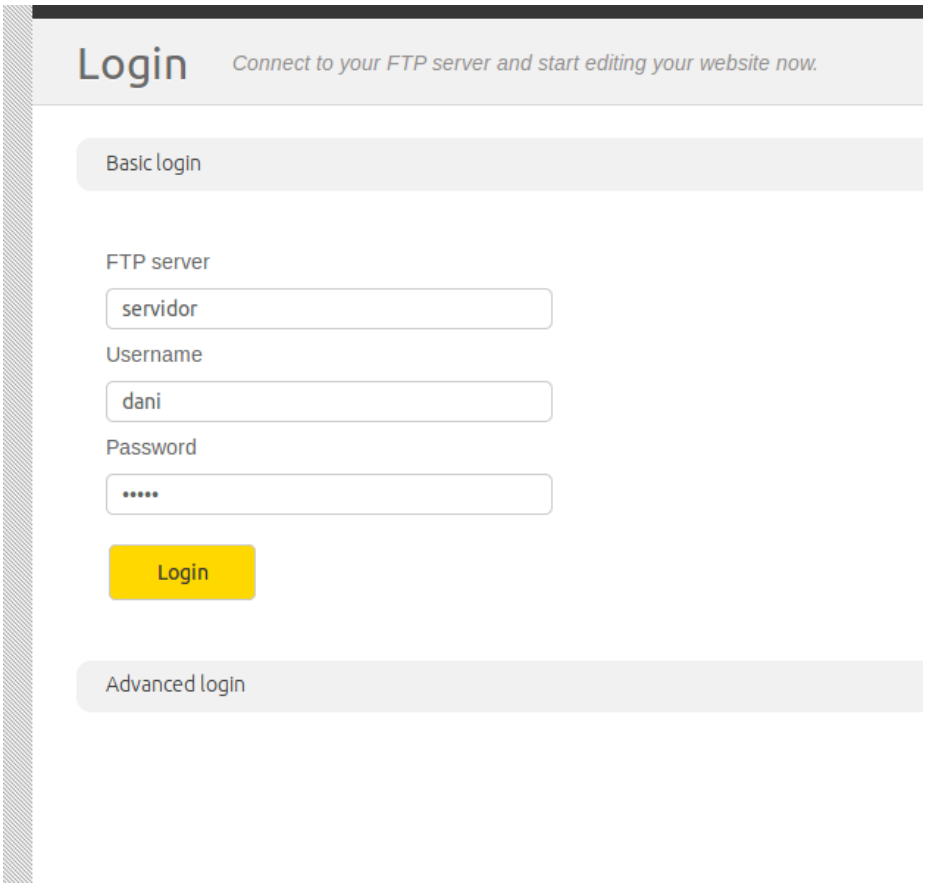


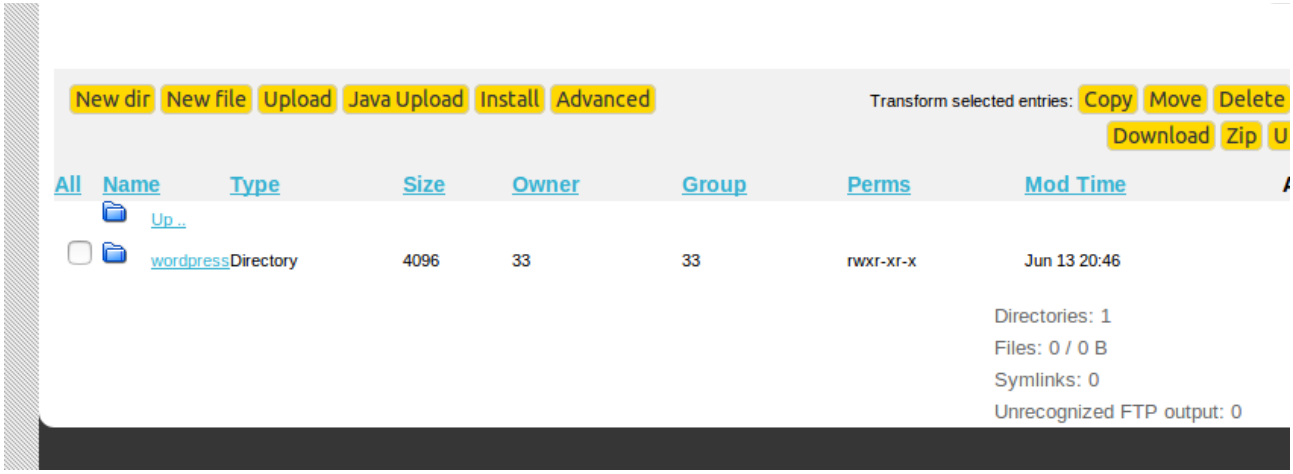
Por último accedemos:



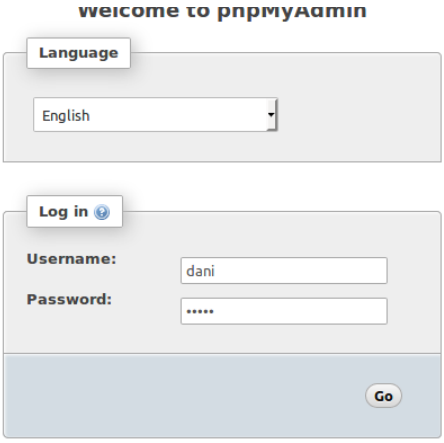
```
root@msi:/home/dani# docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED
STATUS              PORTS                                  NAMES
e5a9da95bf29        danibascon/apache2:1           "/bin/sh -c 'apach..."   9 minutes ago       Up 8
minutes         0.0.0.0:21-22->21-22/tcp, 0.0.0.0:81->80/tcp   servidor_apache
5ebf567057af        danibascon/proftpd             "/bin/sh -c /cmd.sh"     9 minutes ago       Up 9
minutes         21-22/tcp                              servidor
89851e5fb391        danibascon/apache2-usuario:7   "/cmd.sh"                9 minutes ago       Up 9
minutes         0.0.0.0:83->80/tcp                     dani
9fca36a054ce        danibascon/phpmyadmin7-ubuntu:2   "/bin/sh -c 'apach..."   9 minutes ago
Up 9 minutes        0.0.0.0:82->80/tcp                     phpmyadmin
8130fd90649b        danibascon/mysql-ubuntu:1      "/sbin/entrypoint...."   9 minutes ago       Up 9
minutes         3306/tcp                               mysql
```

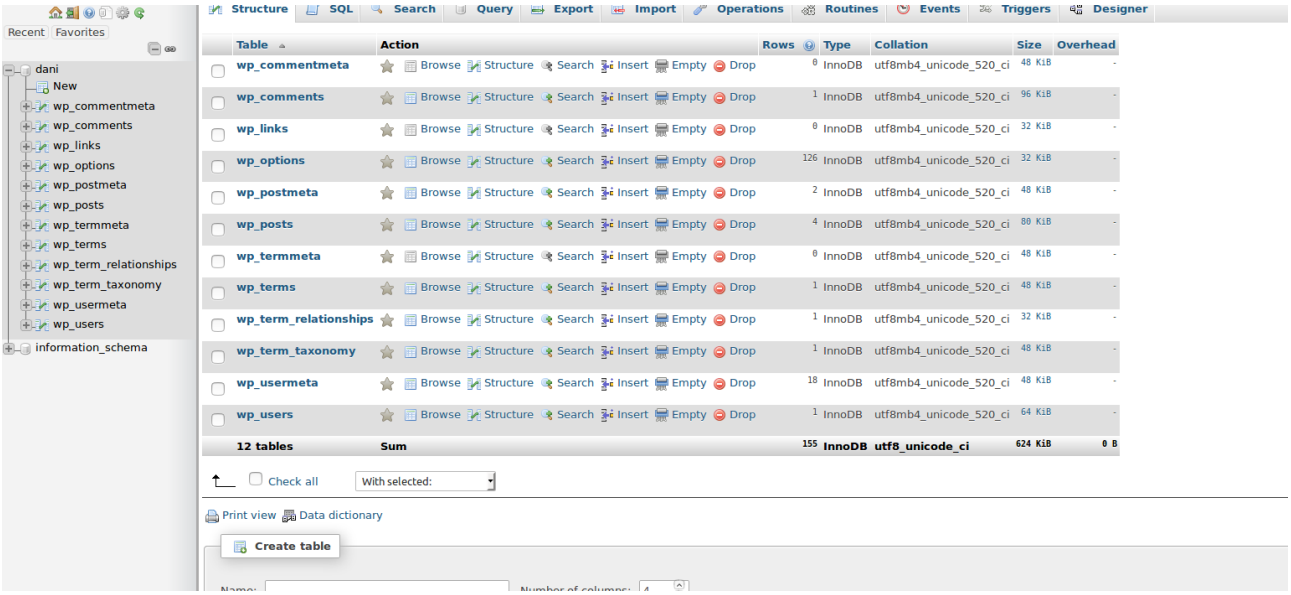
Como vemos, todos los contenedores están activos y en perfecto funcionamiento.





Como vemos, el cliente ftp funciona.





4. Conclusiones.

Conclusiones que he podido sacar a la hora de realizar mi proyecto.

Principalmente destacar que elegir un proyecto con Docker me ayudado bastante a decidirme en mi futuro profesional porque es algo a lo que me gustaría dedicarme, además desplegarlo con Cloud, bien sea AWS u otra alternativa. Es una de mis principales metas.

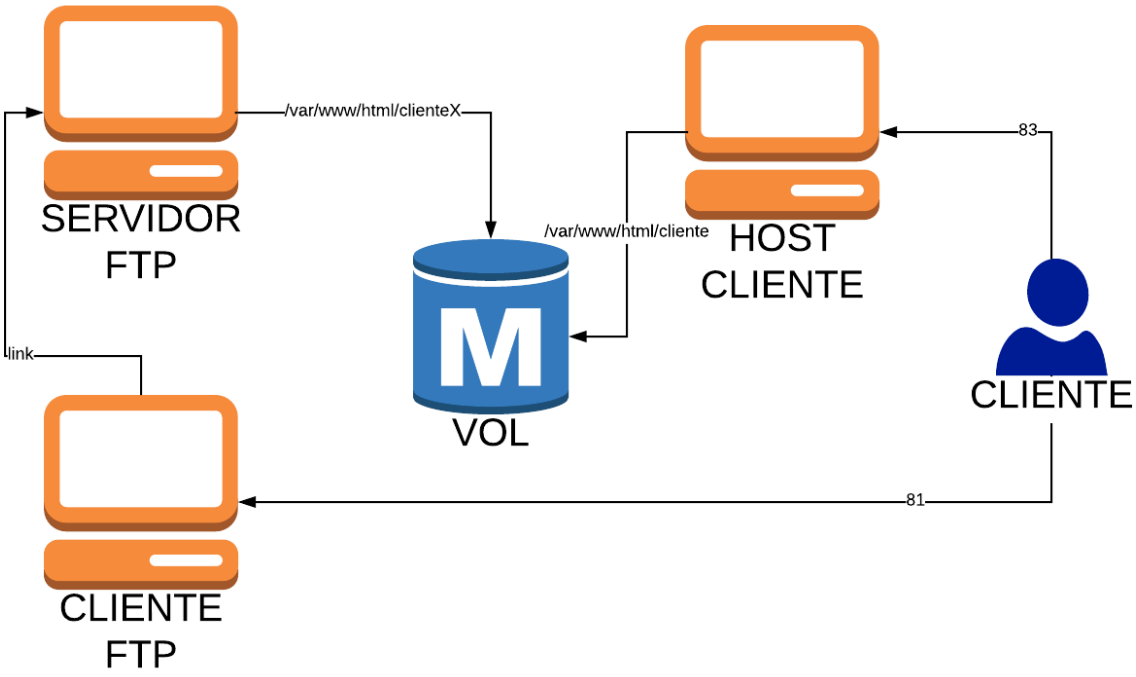
Por otra parte, durante la creación de imágenes de docker que ido utilizando a lo largo del proyecto, he tenido muchos problemas, pensaba que un proyecto con docker, el cual solo tenía que desplegar 4 o 5 contenedores conectados unos con otros no me iba a dar muchos problemas a la hora de hacer el proyecto, pero me equivocaba, vamos a ir detallando los principales problemas con los que me he ido topando a lo largo de las distintas fases que ha tenido mi proyecto.

Primera fase. Creación de proftpd, cliente ftp y usuario.

Son las primeras imágenes docker que cree, algo simples, pues una imagen tenía instalado el paquete proftpd, que es el servidor FTP y otro net2ftp, un clienteFPT con apache2 que se conecta al servidor, y un cliente que no sabía bien a donde conectarlo.

Visto así parece fácil, pero me costó lanzarlo y como lanzarlo, porque claro, se debía crear todo en un contenedor, crear un link, puertos los cuales exponet… Muchas cosas las cuales obvie por falta de conocimiento, pero gracias a las ayuda de Jose Domingo, puede resolverlo, pero tenía las herramientas ya casi listas, pero me faltaba la idea.

Al final conseguirlo hacerlo de la siguiente forma, tenemos 3 contenedores, proftpd, cliente proftpd y cliente, adicionalmente un volumen:



Como podemos ver esta es la estructura que tiene el usuario cuando quiere subir algo por ftp y ver lo que ha subido. Como ya hemos explicado.

Segunda fase. Versiones de php incompatibles con CMS.

Esta fase es la que solucione más rápido pero la que me surgió al final y me volvió un poco lo el proyecto.

La estructura de mis imágenes parten todas de debian jessie, la parte que concierne a msyql es la siguiente, tenia instalado php5 y una versión de mysql compatible con esa versión, el problema fue, que tarde que el mysql-server me cargará las variables que le pasaba a la hora de crear una base de datos, que la final conseguí, el problema fue que la versión de php no cargaba los CMS o pienso yo que ese fue el error, por lo que tuve que tirar toda la estructura de mysql y recrearla primero en debian stretch y posterior en ubuntu 16.04 que es la actual y funciona perfectamente.

El problema que hay con strecth fue que no conseguía pasarle las variables por lo que tuve que optar en utilizar una imagen ya creada de un usuario y retocarla a mis necesidades para que funcionase.

Después de todo esto, conseguí que funcionase todo y perfectamente.

Aunque parece errores simples, tener que programarlo todo y que funciona todo bien y una cosa con la otra, no resulta fácil pero lo conseguí.

Tercera fase. Problemas con DNS.

Esta parte ha sido casi la última, pero también me ha ocasionado problemas para la resolución de nombre, he probado con nginx, como servidor proxy, pero nada, al ser contenedor que escuchan por un puerto y además en nuestra máquina también se abren puerto para los contenedores, me ha ocasionado problema, por lo que tuve que que desistir.

Como segunda opción, utilicé el de apache, pero paso lo mismo, a diferencia es que me si accedía a otro destino que lo fuera el principal, no lo encontraba, y en el principal me redirigía al que le daba la gana, también desistí.

Mi tercena opción fue haproxy, pero tampoco tuve resultado.

Como realicé un pequeño script, el cual recoge las ip de los contenedores y los añade a mi hosts, no es una solución muy buena, pero me ha funcionado. Me gustaría saber una solución a esto, porque añadir DNS, estoy en las mismas, no sabría como aplicarlo a los contenedores.

Como conclusión final, pienso que es un buen proyecto, con sus respectivas carencias en ciertos aspecto, pero como proyecto de integración es una buena idea de lanzar un hosting con docker.

He aprendido bastante de contenedores con este proyecto, como crear una imagen, como exponerlo a otros, como crear link, como tener la información persistente...

5. Trabajos futuros.

Como trabajos futuros para mi proyecto, como a ir comentando los que creo cuales son más importantes y porque:

- **Copias de seguridad**. Esto estaba claro, mantener volúmenes persistente es una cosa, pero se pueden perder, por lo que realizar copias de seguridad de los volúmenes sería la primera opción como mejora.
- **Mejorar aplicación.** Mejorarla principalmente es la segundad prioridad, utilizar otro lenguaje como php, que trabaje siempre en la misma url, y no accediendo a otras. Además mejorar los script de como arrancar los contenedores
- **Kubernetes**. Desplegar los contenedores con kubernetes, los servicios en concreto, replicado en varios nodos y a los clientes, igual.
- **DNS**. Funcionamiento con DNS, para que se resuelva los nombre.
- **Versiones**. Utilizar la misma imagen principal para todo los contenedores. Principalmente ubuntu, por ejemplo.
- **Funciones**. Añadir más funciones al hosting.
- **Docker-compose**. Desplegar los contenedores con docker-compose

- Migración. Desplegar en nube los contenedores.

6. Referencias y bibliografía.

Enlaces que he utilizado como ayuda:

- https://www.josedomingo.org/pledin/tag/docker/
- https://www.digitalocean.com/community/tutorials/docker-explicado-como-crear-contenedores-de-docker-corriendo-en-memcached-es
- http://www.nacionrosique.es/2016/06/componentes-de-dockers.html
- https://picodotdev.github.io/blog-bitix/2014/11/inicio-basico-de-docker/
- https://www.campusmvp.es/recursos/post/como-empezar-a-desarrollar-utilizando-docker.aspx
- https://www.ondho.com/que-es-docker-para-que-sirve/
- https://hub.docker.com/
- https://hub.docker.com/r/mysql/mysql-server/
- https://github.com/mysql/mysql-docker
- https://hub.docker.com/_/mysql/
- https://github.com/docker-library/mysql/tree/fc3e856313423dc2d6a8d74cfd6b678582090fc7
- https://hub.docker.com/r/bitnami/mysql/
- https://github.com/bitnami/bitnami-docker-mysql
- https://coreos.com/quay-enterprise/docs/latest/mysql-container.html
- https://www.redhat.com/es/topics/containers/what-is-docker
- http://babel.es/es/blog/blog/febrero-2017/que-es-docker
- http://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html
- https://www.docker.com/enterprise-edition
- https://blog.codeship.com/the-shortlist-of-docker-hosting/
- https://hub.docker.com/_/httpd/
- https://stackoverflow.com/questions/27768194/how-to-use-docker-container-as-apache-server
- https://blog.irontec.com/desarrollando-con-docker/
- https://writing.pupius.co.uk/apache-and-php-on-docker-44faef716150

Algunos me ha ayudado especialmente en la parte del mysql y para apache, sobre todo para mysql, otros como documentación de docker, definiciones, etc.