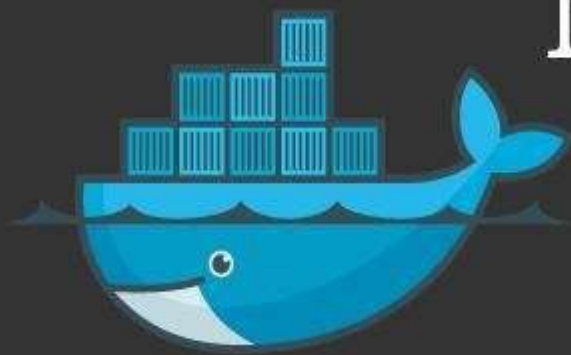


**Build, Ship and Run**



**Meet  
docker**

**Guía para Iniciarte**

# Tabla de contenido

Introducción	0
Acerca del Autor	1
Fundamentos de Docker	2
Arquitectura de Docker	2.1
Instalando Docker	3
Primeros Pasos con Docker	4
Creando Imágenes	5
DockerFiles	5.1
Mejores Prácticas DockerFiles	5.2
Comando Docker Run	6
Otros Comandos	7
Docker Compose	8
Archivo docker-compose en detalle	8.1
Cierre	9

# Acerca del Libro

Muchos han escuchado hablar de Docker pero no muchos sabe exactamente de que se trata, por que es tan popular y en que nos puede beneficiar según nuestro ambiente.

- Qué es Docker
- Entendiendo la Arquitectura
- Como se compara Docker con las Máquinas Virtuales
- Instalando Docker
- Usando Docker

Cuando termine de leer esta guía será capaz de entender lo que es docker y como utilizarlo.

## Experiencia

Mas de 10 años de Experiencia en el Area de TI, en la Administración de sistemas Linux y algunos años en Seguridad de Información. En Ocasiones da consultorías en el area de Linux/Windows hardening, Seguridad y realiza test de Penetración. Actualmente es el Encargado de Seguridad de información y Linux SysAdmin en una organización.

## Experiencia Informática

**Sistemas Operativos:** UNIX, GNU/Linux, Windows, Windows Server, Oracle Solaris, FreeBSD

**Desarrollo:** Bash Scripting, Python

**IT Security/Ethical Hacking:** Penetration Testing, Metasploit Framework, Ethical Hacking, Linux/Windows Hardening, Wireless Security, Network Security

**DevOps:** Chef, Jenkins, Docker, Puppet

**Cloud:** OpenStack, CloudU Certification

## Proyectos

Constantemente hace publicaciones sobre Seguridad, Ethical Hacking, Linux entre otros, en su blog [https://www.itskills.es](#). La Idea es compartir el conocimiento con todos los profesionales del área de la Tecnología de la Información especialmente en Seguridad informática. Cuenta además con otros proyectos:

### IT-Talks

Aquí creamos Hangouts periódicamente sobre temas del área de TI, que incluyen, Seguridad, Desarrollo, Hacking, Administración de Sistemas, entre otros.

## Security Feed

Aquí Recopilamos los titulares de los Blogs más conocidos de Seguridad y Ethical Hacking.

# Fundamentos de Docker

## Que es Docker?



Docker es un proyecto Open Source basado en contenedores de Linux, es básicamente un motor de contenedores que usa características del Kernel de Linux como espacios de nombres y controles de grupos para crear contenedores encima del Sistema operativo y automatizar el despliegue de aplicaciones en estos contenedores. Nos permite además un flujo de trabajo bastante eficiente al momento de mover nuestras aplicaciones desde un ambiente de desarrollo, a pruebas y finalmente a producción.

## Hablamos de Espacios de nombres y Control de Grupos, pero que son estos?

### Espacios de Nombres (namespaces)

Un espacio de nombre envuelve un recurso de Sistema global en una abstracción que le hace parecer al proceso dentro del espacio de nombre, que tiene su propia instancia aislada del recurso. Los cambios al recurso global son visibles para los proceso dentro del espacio de nombre pero invisible para otros procesos. Una de las implementaciones mas utilizada son los contenedores.

**Linux provee los espacios de nombre:**

**IPC (Interprocess Communication Mechanism):** Mecanismo de comunicación Interproceso, aisle ciertos recursos IPC, específicamente System V IPC y cola de mensajes POSIX.

**NETWORK:** Provee aislamiento de los recursos del Sistema asociados con las comunicaciones de red, Dispositivos de red, Stack de protocolos IPv4 e IPv6, tablas de enrutamiento, firewalls, directorio /proc/net, sockets de comunicación entre otros.

**PID:** Aisla el espacio de nombre del ID de proceso, que significa que procesos en otros espacios de nombres PID, pueden tener el mismo PID. Esto le permite a los contenedores proveer funcionalidades de suspensión y recuperación de procesos en el contenedor y la migración del contenedor a otro host manteniendo el mismo PID.

**User:** Aisla atributos e identificadores relacionados con la seguridad, particularmente los ID's de usuarios y grupos, el directorio raíz, permisos, etc. El usuario y el grupo de un proceso puede ser distinto dentro y fuera del espacio de nombre de usuario, Quiere decir que un proceso puede tener un ID no privilegiado fuera del espacio de nombre y un ID de 0 dentro del espacio de nombre.

**UTS:** Provee aislamiento de dos identificadores de Sistema. El hostname y el nombre del dominio NIS.

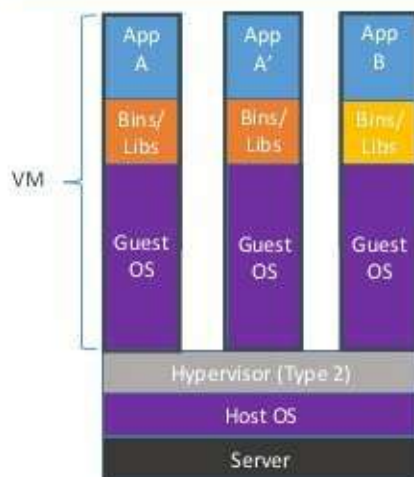
### **Control de Grupos**

Los recursos usados por un contenedor son controlados por el control de grupos. Se puede configurar cuanto CPU y memoria usa un contenedor haciendo uso de ellos.

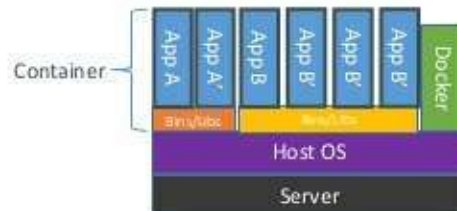
## **Docker VS Máquinas Virtuales**

La diferencia mas notable es que las Máquinas Virtuales son sistemas operativos completos por lo que tendríamos que contar con muchos recursos. Docker en cambio aprovecha los recursos del sistema operativo sobre el cual se ejecuta, como el Kernel y librerías. Es por esta razón que terminamos con imágenes que ocupan muy poco espacio.

# Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries





# Arquitectura de Docker

**Docker** usa una arquitectura **cliente-servidor**. El cliente de Docker habla con el Daemon de Docker que hace el trabajo de crear, correr y distribuir los contenedores. Tanto el cliente como el Daemon pueden ejecutarse en el mismo Sistema, o puede conectar un cliente remoto a un daemon de docker.

El **cliente de Docker** es la principal interfaz de usuario para docker, acepta los comandos del usuario y se comunica con el daemon de docker.

Para entender como funciona docker internamente debemos conocer tres componentes.

## Imágenes de Docker (Docker Images)

Las imágenes de Docker son plantillas de solo lectura, es decir, una imagen puede contener el Sistema operativo de CentOS o Ubuntu con apache instalados, pero esto solo nos permitirá crear los contenedores basados en esta configuración. Si hacemos cambios en el contenedor ya lanzado, al detenerlo esto no se verá reflejado en la imagen. Mas adelante entenderemos esta parte.

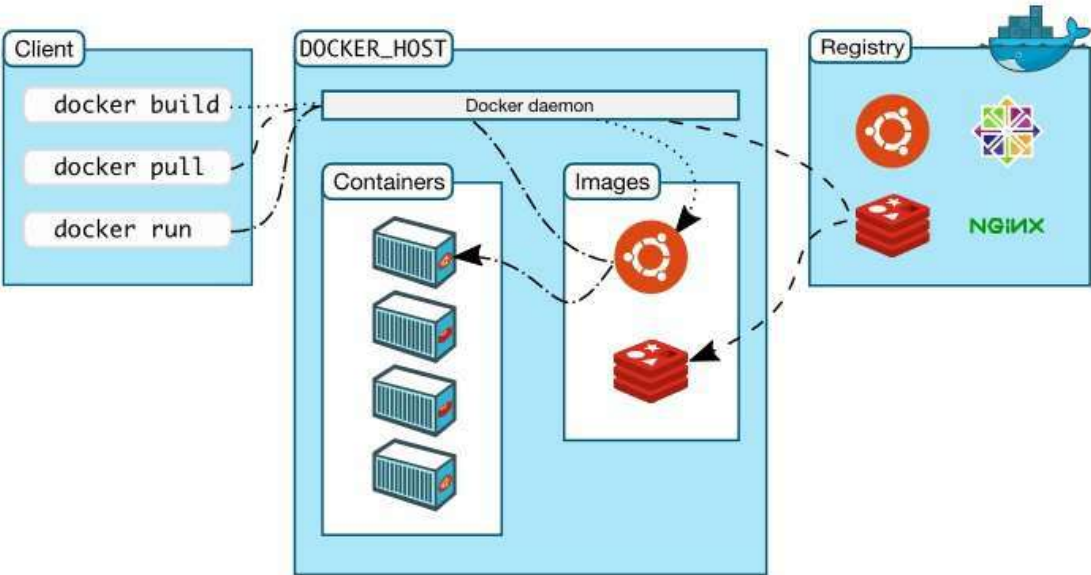
## Registros de Docker (Docker Registries)

Los registros de Docker guardan las imágenes, estos son repos públicos o privados donde podemos subir o descargar imágenes. El registro público lo provee el Hub de Docker que sirve una colección de imágenes para nuestro uso. Los registros de dockers básicamente son el componente de Distribución de Docker.

## Contenedores de Docker (Docker Containers)

El contenedor de docker aloja todo lo necesario para ejecutar una aplicación. Cada contenedor es creado de una imagen de docker. Cada contenedor es una plataforma aislada.

## Veamos gráficamente la arquitectura



# Instalando Docker

En esta guía veremos la instalación de Docker en CentOS, Ubuntu y Debian.

## CentOS 6

### Instalamos el Repo EPEL

```
$ rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

## CentOS 7

```
$ yum update
$ vi /etc/yum.repos.d/docker.repo

[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/$releasever/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
```

### Procedemos con la instalación de Docker

## CentOS 6

```
$ yum install docker-io
```

## CentOS 7

```
$ yum install docker-engine
```


## Ubuntu 14.04/15.04

```
$ apt-get update

$ apt-get install apt-transport-https ca-certificates
```

## Agregamos la nueva llave gpg

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F
```



## Agregamos los repos de Docker

```
$ nano /etc/apt/sources.list.d/docker.list
```

**\*\* Ubuntu 14.04**

```
``deb https://apt.dockerproject.org/repo ubuntu-trusty main
```

**\*\* Ubuntu 15.04**

```
``deb https://apt.dockerproject.org/repo ubuntu-wily main
```

**\*\*Procedemos con la Instalación**

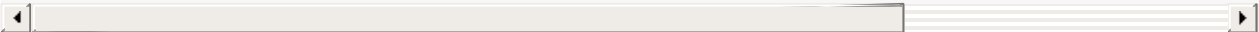
```
$ apt-get purge lxc-docker  
$ apt-get install docker-engine
```

## Debian 7/8

```
$ apt-get purge lxc-docker*  
$ apt-get purge docker.io*
```

## Agregamos la nueva llave gpg

```
$ apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A91
```



## Agregamos los repos de Docker

```
$ nano /etc/apt/sources.list.d/docker.list
```

**Debian 7**

```
deb https://apt.dockerproject.org/repo debian-wheezy main
```

## Debian 8

```
deb https://apt.dockerproject.org/repo debian-jessie main
```

```
$ apt-get update  
$ apt-get install docker-engine
```

Una vez instalamos vamos asegurarnos que el servicio de Docker suba cada vez que booteemos nuestros servers.

## Centos 6

```
$ chkconfig docker on
```

## Centos 7

```
$ systemctl enable docker.service
```

## Ubuntu 14.04/15.04

```
$ chkconfig docker on
```

Iniciamos los servicios para asegurarnos que todo anda bien.

## CentOS 6

```
$ service docker start
```

## Centos 7

```
$ systemctl start docker.service
```

## Ubuntu 14.04/15.04

```
$ service docker start
```

## Debian 7/8

```
$ service docker start
```

Ya que tenemos el ambiente listo. Vamos a ejecutar el commando Docker a ver que nos trae.

```
$ docker
```



```

root@~# docker
Usage: docker [OPTIONS] COMMAND [arg...]

A self-sufficient runtime for linux containers.

Options:
  --add-registry=[]          Registry to query before a public one
  --api-cors-header=         Set CORS headers in the remote API
  -b, --bridge=              Attach containers to a network bridge
  --bip=                     Specify network bridge IP
  --block-registry=[]        Don't contact given registry
  --confirm-def-push=true    Confirm a push to default registry
  -D, --debug=false          Enable debug mode
  -d, --daemon=false         Enable daemon mode
  --default-gateway=         Container default gateway IPv4 address
  --default-gateway-v6=     Container default gateway IPv6 address
  --default-ulimit=[]        Set default ulimits for containers
  --dns=[]                   DNS server to use
  --dns-search=[]            DNS search domains to use
  -e, --exec-driver=native   Exec driver to use
  --exec-opt=[]              Set exec driver options
  --exec-root=/var/run/docker Root of the Docker execdriver
  --fixed-cidr=              IPv4 subnet for fixed IPs
  --fixed-cidr-v6=           IPv6 subnet for fixed IPs
  -G, --group-docker          Group for the unix socket
  -g, --graph=/var/lib/docker Root of the Docker runtime
  -H, --host=[]              Daemon socket(s) to connect to
  -h, --help=false           Print usage
  --icc=true                  Enable inter-container communication
  --insecure-registry=[]     Enable insecure registry communication
  --ip=0.0.0.0                Default IP when binding container ports
  --ip-forward=true          Enable net.ipv4.ip_forward
  --ip-masq=true              Enable IP masquerading
  --iptables=true             Enable addition of iptables rules
  --ipv6=false                Enable IPv6 networking
  -l, --log-level=info        Set the logging level
  --label=[]                  Set key=value labels to the daemon
  --log-driver=json-file      Default driver for container logs
  --log-opt=map[]             Set log driver options
  --mtu=0                     Set the containers network MTU
  -p, --pidfile=/var/run/docker.pid Path to use for daemon PID file
  --registry-mirror=[]        Preferred Docker registry mirror
  -s, --storage-driver=       Storage driver to use
  --selinux-enabled=false     Enable selinux support
  --storage-opt=[]            Set storage driver options
  --tls=false                 Use TLS; implied by --tlsverify
  --tlscacert=~/.docker/ca.pem Trust certs signed only by this CA
  --tlscert=~/.docker/cert.pem Path to TLS certificate file
  --tlskey=~/.docker/key.pem   Path to TLS key file
  --tlsverify=false           Use TLS and verify the remote
  --userland-proxy=true       Use userland proxy for loopback traffic
  -v, --version=false         Print version information and quit

Commands:
  attach  Attach to a running container
  build   Build an image from a Dockerfile
  commit  Create a new image from a container's changes
  cp      Copy files/folders from a container's filesystem to the host path
  create  Create a new container

```

```

root@~:~
--label=[]                Set key=value labels to the daemon
--log-driver=json-file    Default driver for container logs
--log-opt=map[]           Set log driver options
--mtu=0                   Set the containers network MTU
-p, --pidfile=/var/run/docker.pid Path to use for daemon PID file
--registry-mirror=[]      Preferred Docker registry mirror
-s, --storage-driver=     Storage driver to use
--selinux-enabled=false   Enable selinux support
--storage-opt=[]          Set storage driver options
--tls=false               Use TLS; implied by --tlsverify
--tlscacert=~/docker/ca.pem Trust certs signed only by this CA
--tlscert=~/docker/cert.pem Path to TLS certificate file
--tlskey=~/docker/key.pem  Path to TLS key file
--tlsverify=false         Use TLS and verify the remote
--userland-proxy=true     Use userland proxy for loopback traffic
-v, --version=false       Print version information and quit

Commands:
  attach      Attach to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders from a container's filesystem to the host path
  create      Create a new container
  diff        Inspect changes on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Stream the contents of a container as a tar archive
  history     Show the history of an image
  images      List images
  import      Create a new filesystem image from the contents of a tarball
  info        Display system-wide information
  inspect     Return low-level information on a container or image
  kill        Kill a running container
  load        Load an image from a tar archive
  login       Register or log in to a Docker registry server
  logout      Log out from a Docker registry server
  logs        Fetch the logs of a container
  pause       Pause all processes within a container
  port        Lookup the public-facing port that is NAT-ed to PRIVATE_PORT
  ps          List containers
  pull        Pull an image or a repository from a Docker registry server
  push        Push an image or a repository to a Docker registry server
  rename      Rename an existing container
  restart     Restart a running container
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save an image to a tar archive
  search      Search for an image on the Docker Hub
  start       Start a stopped container
  stats       Display a stream of a containers' resource usage statistics
  stop        Stop a running container
  tag         Tag an image into a repository
  top         Lookup the running processes of a container
  unpause     Unpause a paused container
  version     Show the Docker version information
  wait        Block until a container stops, then print its exit code

Run 'docker COMMAND --help' for more information on a command.
[root@ ~]#

```

Nos trae información sobre las opciones que le podemos pasar a docker. Recuerdan que el post anterior hablamos de los registros de Docker o Docker Registries que es un repo con diversas imágenes para descargar, para probar que todo anda bien, vamos a realizar una búsqueda de alguna imagen y vamos a descargarla.

```
$ docker search Ubuntu
```

```

root@~# docker search ubuntu
INDEX      NAME                DESCRIPTION                STARS    OFFICIAL    AUTOMATED
docker.io  docker.io/ubuntu    Ubuntu is a Debian-based Linux operating s...  2539     [OK]
docker.io  docker.io/ubuntu-upstart  Upstart is an event-based replacement for ...  40       [OK]
docker.io  docker.io/torware/speedius-ubuntu  Always updated official Ubuntu docker imag...  25
docker.io  docker.io/sequenceiq/hadoop-ubuntu  An easy way to try Hadoop on Ubuntu          23       [OK]
docker.io  docker.io/tleyden5iwx/ubuntu-cuda   Ubuntu 14.04 with CUDA drivers pre-installed  18       [OK]
docker.io  docker.io/ubuntu-debootstrap  debootstrap --variant=minbase --components...  17       [OK]
docker.io  docker.io/rastashoop/ubuntu-sshd    Dockerized SSH service, built on top of of...  14       [OK]
docker.io  docker.io/quinhm/vagrant-ubuntu     Ubuntu with Oracle JDK. Check tags for ver...  11       [OK]
docker.io  docker.io/m321muk5/ubuntu-oracle-jdk  Ubuntu with Oracle JDK. Check tags for ver...  4       [OK]
docker.io  docker.io/bamersohn/ubuntu          This is a docker images different LTS vers...  3       [OK]
docker.io  docker.io/nuegben/ubuntu            Simple always updated Ubuntu docker images...  3       [OK]
docker.io  docker.io/armhf/ubuntu-debootstrap  ARMHF port of ubuntu-debootstrap            2       [OK]
docker.io  docker.io/armhf/ubuntu              [ARM] Ubuntu Docker images for the ARMv7(a...  2       [OK]
docker.io  docker.io/masexclou/ubuntu          Docker base image built on Ubuntu with Sup...  2       [OK]
docker.io  docker.io/basslibrary/ubuntu        ThoughtWorks Ubuntu Docker image            1       [OK]
docker.io  docker.io/densuke/ubuntu-jp-remix    Ubuntu Linux® 日本語 remix 風味 © 丁        1       [OK]
docker.io  docker.io/iisuper/bnac-ubuntu        This is just a small and clean base Ubuntu...  1       [OK]
docker.io  docker.io/seetheprogress/ubuntu      Ubuntu image provided by seetheprogress us...  1       [OK]
docker.io  docker.io/densuke/ubuntu-jp-remix:trusty  Ubuntu LTS                                  0       [OK]
docker.io  docker.io/esyoat/ubuntu              Ubuntu with the needful                      0       [OK]
docker.io  docker.io/rallias/ubuntu              https://github.com/tvaughan/docker-ubuntu    0       [OK]
docker.io  docker.io/vicamo/ubuntu-phablist-giexi  Dockerfile for developing Ubuntu JieXi PDK.  0       [OK]
docker.io  docker.io/roni/ubuntu                0       [OK]

```

Aquí vemos una lista de imágenes disponible. Vamos a descargar la imagen base de Ubuntu

```
$ docker pull docker.io/Ubuntu
```

O

```
$ docker pull Ubuntu:latest
```

Nota: deben pasar el nombre de la imagen como les aparece en la búsqueda

Si ejecutamos docker images debemos ver la imagen descargada.

```
$ docker images
```

```

root@~# docker images
REPOSITORY    TAG                IMAGE ID            CREATED             VIRTUAL SIZE
centos6       sersvex1          6477763d63fe       7 hours ago        475.4 MB
centos6       baseweb2          78749a1c4c07       8 hours ago        475.4 MB
centos6       baseweb           6342a99581bb       8 hours ago        475.4 MB
jstetch/ubuntu-apache  v1.0             b773b0560571       18 hours ago       223.5 MB
docker.io/wordpress  latest           6c99bdc27dd4       4 days ago         512.1 MB
docker.io/ubuntu     latest           a5a467fddcb8       5 days ago         187.9 MB
docker.io/centos     centos6          3bbb5f0aca958       2 weeks ago        190.6 MB

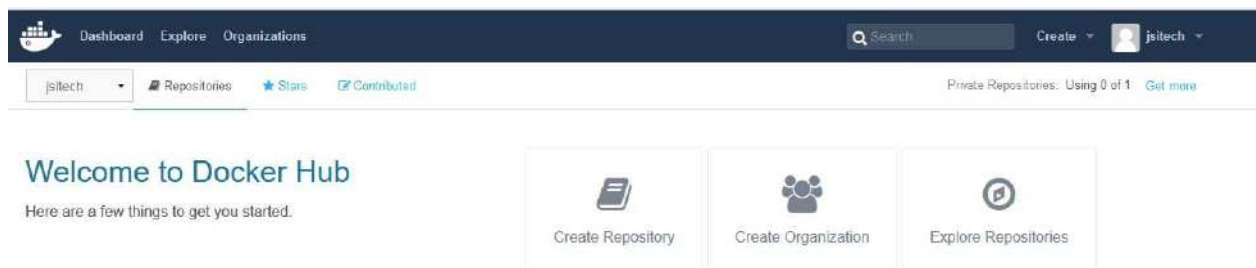
```

Si podemos realizar todas estas tareas nuestra instalación esta correcta.



# Primeros Pasos con Docker

En Capítulos anteriores vimos lo que es Docker y como instalarlo en Linux CentOS y Ubuntu. Pero antes de entrar de lleno con Docker, recuerdan que les hablaba de Docker registries, quiero hablarles brevemente de **Docker Hub**.



**Docker Hub** es un servicio de registro basado en la nube para crear y entregar contenedores de servicio o aplicaciones. Provee un recurso centralizado para el descubrimiento de contenedores, distribución, control de cambios, colaboración de equipos y automatización de flujos de trabajo. Veanlo como como un **Github** de contenedores.

## Características de Docker Hub

- **Repositorios de Imágenes:** Encuentra, administra, sube y descarga imágenes de la comunidad y oficiales
- **Imágenes automáticas:** Crea nuevas imágenes cuando haces un cambio en la Fuente de Github o BitBucket
- **Webhooks:** creaciones automáticas de imágenes al hacer un push exitoso a un repositorio.
- **Integración con GitHub y BitBucket**

Docker Hub le provee a usted y su organización un lugar donde alojar y entregar las imágenes.

Se puede configurar los repositorios de Docker Hub de dos maneras:

**Repositorios**, que nos permiten subir y actualizar las imágenes cuando deseemos desde el docker daemon y las **imágenes automáticas** que nos permiten configurar una cuenta de Github o BitBucket que desencadenan la reconstrucción de una imagen cuando se realizar algún cambio en el repositorio.

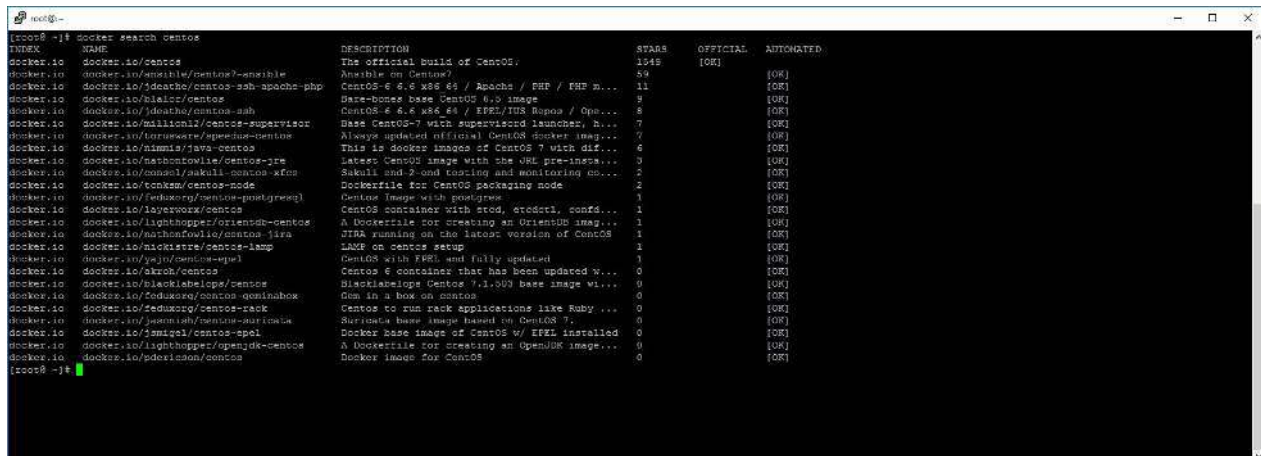
Se pueden crear repositorios públicos que pueden ser accedidos por otros usuarios de HUB, o se pueden crear repositorios privados con accesos limitados.

Pueden crear su cuenta en <https://hub.docker.com>, luego veremos como podemos aprovechar todo esto.

El daemon de Docker hace uso de Hub para obtener y subir sus imágenes. Ya que tenemos una base, vamos a jugar un poco con Docker.

## Busquemos una imagen base de Centos

```
$ docker search centos
```



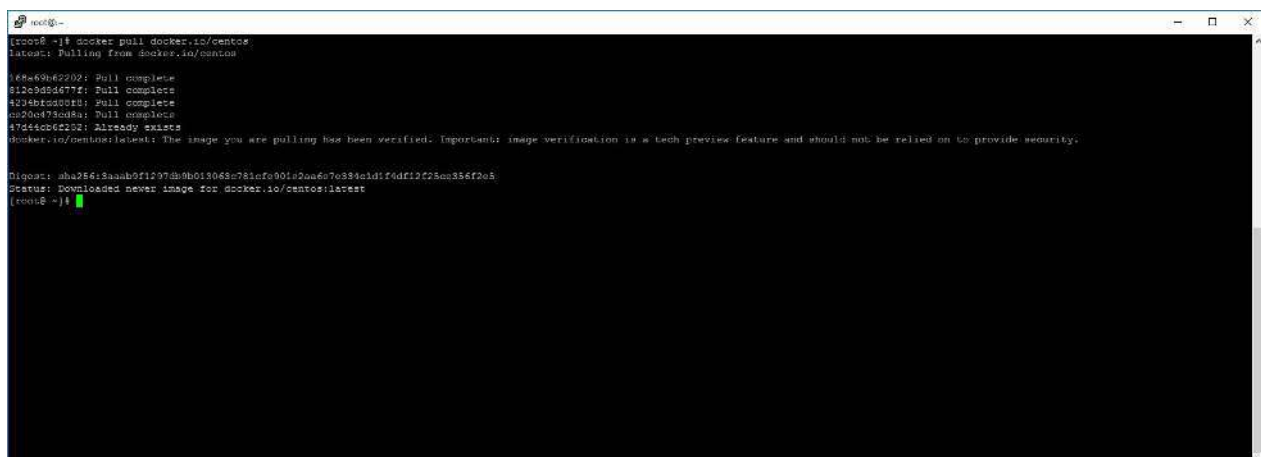
```

[roos@ ~]$ docker search centos
INDEX      NAME                                     DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
docker.io/docker.io/centos              The Official Build of CentOS;               1346      [OK]
docker.io/centos/centos7-ansible        Available on CentOS7;                       89
docker.io/centos/centos7-ssh-apache-php CentOS-6 6.6 x86_64 / Apache / PHP / PHP M... 11
docker.io/centos/centos7-ssh-apache-php Bare-bones base CentOS 6.9 image             9
docker.io/centos/centos7-ssh-apache-php CentOS 6 6.6 x86_64 / PHP/PHP Mysql / Qna... 8
docker.io/centos/centos7-ssh-apache-php Base CentOS7 with supervisor launcher, h... 7
docker.io/centos/centos7-ssh-apache-php Always updated official CentOS docker imag... 7
docker.io/centos/centos7-ssh-apache-php This is docker images of CentOS 7 with dif... 6
docker.io/centos/centos7-ssh-apache-php Latest CentOS image with the OEL pre-insta... 3
docker.io/centos/centos7-ssh-apache-php Sakuli and 2-ond testing and monitoring co... 2
docker.io/centos/centos7-ssh-apache-php Dockerfile for CentOS packaging mode         2
docker.io/centos/centos7-ssh-apache-php CentOS image with postgres                  1
docker.io/centos/centos7-ssh-apache-php CentOS container with stord, etcdctl, confd... 1
docker.io/centos/centos7-ssh-apache-php A Dockerfile for creating an OpenStack imag... 1
docker.io/centos/centos7-ssh-apache-php JIRA running on the latest version of CentOS 1
docker.io/centos/centos7-ssh-apache-php LAMP on centos setup                       1
docker.io/centos/centos7-ssh-apache-php CentOS with FRR and fully updated          1
docker.io/centos/centos7-ssh-apache-php CentOS 6 container that has been updated w... 0
docker.io/centos/centos7-ssh-apache-php AlpineLinux CentOS 7.1.309 base image w... 0
docker.io/centos/centos7-ssh-apache-php Gem in a box on centos                    0
docker.io/centos/centos7-ssh-apache-php CentOS to run rack applications like Ruby ... 0
docker.io/centos/centos7-ssh-apache-php Suricata base image based on CentOS 7.      0
docker.io/centos/centos7-ssh-apache-php Docker base image of CentOS w/ IRIX install... 0
docker.io/centos/centos7-ssh-apache-php A Dockerfile for creating an OpenJDK image... 0
docker.io/centos/centos7-ssh-apache-php Docker image for CentOS                    0
[roos@ ~]$

```

Vamos a descargar la imagen oficial base de centos. Asi que nos fijamos en NAME y lo pasamos al docker pull

```
$ docker pull docker.io/centos
```



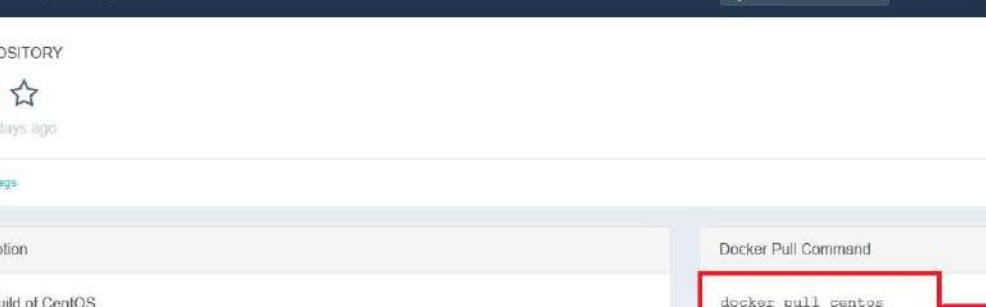
```

[roos@ ~]$ docker pull docker.io/centos
latest: Pulling from docker.io/centos
46a6942202: Pull complete
412e98d4777f: Pull complete
4734b1d0018: Pull complete
4a20e473e08a: Pull complete
47d4c06f20a: Already exists
docker.io/centos:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.
Digest: sha256:8a2ab0f1297db0b013068b781af900102a6f07e384c4d1f6df12f28a356f2e5
Status: Downloaded newer image for docker.io/centos:latest
[roos@ ~]$

```

Otra forma que podemos usar para buscar una imagen es mediante el Hub.

Si hacemos la búsqueda de centos tendremos algo así.



OFFICIAL REPOSITORY

centos ☆

Last pushed: 8 days ago

Repo info Tags

Short Description

The official build of CentOS.

Full Description

## Supported tags and respective Dockerfile links

- latest, centos7, 7 (docker/Dockerfile)
- centos6, 6 (docker/Dockerfile)
- centos5, 5 (docker/Dockerfile)
- centos7.1.1503, 7.1.1503 (docker/Dockerfile)
- centos7.0.1406, 7.0.1406 (docker/Dockerfile)
- centos6.7, 6.7 (docker/Dockerfile)
- centos6.6, 6.6 (docker/Dockerfile)

Tags

Docker Pull Command

```
docker pull centos
```

COMANDO

Aquí tenemos el comando y los tags para obtener una version específica. Si ven tenemos como comando `docker pull centos`, y si deseamos por ejemplo descargar la version 7 de centos o la última version, los comandos se verían así.

```
$ docker pull centos:centos7
```

```
$ docker pull centos:latest
```

Ya descargada la imagen, confirmamos que este ahí con

```
$ docker images
```

docker.io/centos latest ce20c473cd8a 2 weeks ago 172.3 MB

```
root@ ~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
docker.io/ubuntu:spashe    v1.0               27f30b360671       20 hours ago       222.9 MB
docker.io/wordpress:latest    latest             4c990ac27d64       4 days ago         312.1 MB
docker.io/ubuntu:latest      latest             a5a167cdddb8       6 days ago         187.9 MB
docker.io/centos:centos6     centos6            3bbb1faca353       2 weeks ago        180.6 MB
docker.io/centos:latest      latest             be20c473b00a       2 weeks ago        172.3 MB
root@ ~#
```

Ya vemos que tenemos la imagen ahí. Ahora como podemos interactuar con esa imagen o lanzar un contenedor con esa imagen? A eso vamos, pero antes recuerden esto, cuando lanzamos un contenedor, y hacemos cualquier tarea esto no se ve reflejado al momento de lanzar otro contenedor con la misma imagen

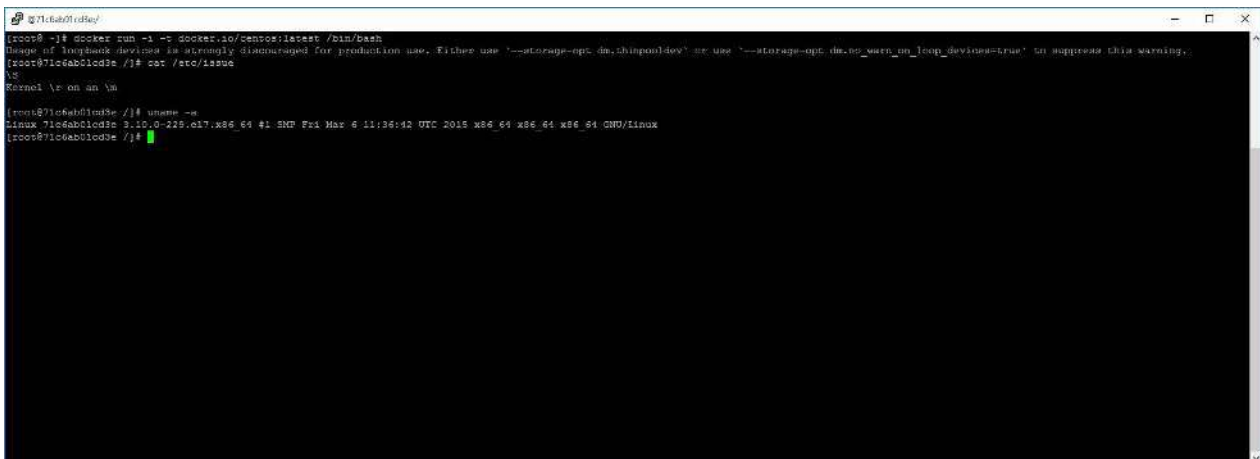
Vamos a lanzar un contenedor basado en esa imagen e interactuar con la línea de comandos de ese contenedor

```
$ docker run -i -t docker.io/centos:latest /bin/bash
```

-i : Interactivo

-t: tty

[root@71c6ab01cd3e /]#



```
[root@71c6ab01cd3e /]# docker run -i -t docker.io/centos:latest /bin/bash
Warning: Permanently added the GPG key for the repository, but the key is not certified by a trusted keyring.
Usage of insecure devices is strongly discouraged for production use. Either use '--storage-opt dm.discarddev' or use '--storage-opt dm.no_warn_on_loop_devices=true' to suppress this warning.
[root@71c6ab01cd3e /]# cat /etc/issue
Linux
Kernel \r on an \n
[root@71c6ab01cd3e /]# uname -a
Linux 71c6ab01cd3e 3.10.0-229.el7.x86_64 #1 SMP Fri Mar 6 11:36:42 UTC 2015 x86_64 x86_64 GNU/Linux
[root@71c6ab01cd3e /]#
```

Aquí ya estamos interactuando con el contenedor, esa numeración que ven luego de root es el id del contenedor, podemos actualizar el Sistema, instalar paquetes, etc. Este contenedor también cuenta con una IP con la que podemos interactuar con el contenedor.

Veamos los contenedores que tenemos corriendo. Abran otra terminal en el server.

```
$ docker ps
```

```
root@~# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
71c6ab01cd3e   docker.io/cen   "/bin/bash"             3 minutes ago   Up 3 minutes   reverent_bardeen
```

Aquí tenemos el ID de contenedor, la imagen que usamos para lanzarlo, el comando que corrimos y al final el Nombre. Con este nombre que docker le asignó aleatoriamente podemos interactuar con el contenedor. Este Nombre por igual lo podemos asignar nosotros.

Si queremos detener el contenedor

```
$ docker stop reverent_bardeen (que es el nombre asignado por docker)
```

Si corremos docker ps, veremos que ya no hay nada. Lo Bueno del caso es que ese contenedor permanece ahí, con los cambios que nosotros le realizamos.

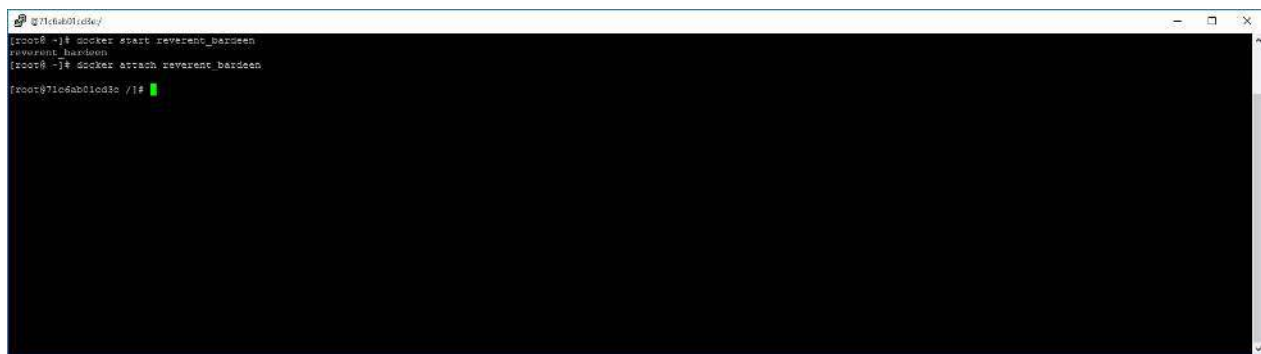
```
$ docker ps -a
```

```
root@~# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
71c6ab01cd3e   docker.io/cen   "/bin/bash"             25 minutes ago   Exited (137) 15 seconds ago   reverent_bardeen
4359d0c4139e   docker.io/ubuntu:latest   "/bin/bash"             13 hours ago     Dead
```

Podemos reiniciar nuevamente ese contenedor y conectarnos a el

```
$ docker start reverent_bardeen
```

```
$ docker attach reverent_bardeen
```



```
@nick01olev/  
[root@ ~]# docker start reverent_bardeen  
reverent_bardeen  
[root@ ~]# docker attach reverent_bardeen  
/root@71c6ab10d3e: /#
```

Aquí estamos nuevamente conectado con nuestro contenedor.

# Creando Imágenes

las imágenes son plantillas de solo lectura que podemos usar como base para lanzar contenedores. Esto quiere decir que lo que hagamos en el contenedor solo persiste en ese contenedor, esas modificaciones no lo hacemos en la imagen, es decir, que si queremos contar con una imagen personalizada debemos crearla para nuestros futuros contenedores. En este apartado vamos a ver como podemos crear una imagen desde un contenedor que ya hemos personalizado.

## Creando imágenes desde un contenedor

Esta puede ser la manera más sencilla de crear una imagen, ya que lanzamos un contenedor, descargamos paquetes por ejemplo y podemos crear una imagen a partir de ahí para utilizarla como base en futuros contenedores. Vamos a ver como lo podemos hacer.

Lo primero es que debemos lanzar un contenedor con una imagen, en este caso tenemos un contenedor basado en Kali Linux, y queremos crear una imagen que tenga NMAP ya instalado.

Confirmo las imágenes

```
$ docker images
```

```
[root@Lap-Sec ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
kalilinux/kali-linux-docker  latest             e49f6054690a       2 weeks ago        420.3 MB
[root@Lap-Sec ~]#
```

Lanzamos el contenedor con la imagen que tenemos de Kali.

```
$ docker run -i -t kalilinux/kali-linux-docker:latest /bin/bash
```

Ya dentro del contenedor instalamos NMAP

```
root@c539a98f1f80:/# apt-get update

root@c539a98f1f80:/# apt-get install nmap
```

**Ya que lo instalamos, como creo una imagen a partir de este contenedor para contar con una imagen con Nmap preinstalado?**

Salimos del contenedor

```
root@c539a98f1f80:/# exit
```

Vamos ahora a mostrar todos los contenedores que hemos lanzado aún no esten corriendo

```
$ docker ps -a
```

```
[root@Lap-Sec ~]# docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS                    PORTS          NAMES
c539a98f1f80   kalilinux/kali-linux-docker:latest  "/bin/bash"            8 minutes ago  Exited (255) 2 minutes ago  big_cori
```

Aquí vemos listado el último contenedor que lanzamos, solamente debemos tomar el **CONTAINER ID** para hacer un commit a una nueva imagen

```
$ docker commit c539a98f1f80 jsitech/kali-nmap:latest
```

Docker creará la imagen, confirmemos que es así, listando las imágenes que tenemos y debemos tener la que acabamos de crear.

```
$ docker images
```

```
[root@Lap-Sec ~]# docker images
REPOSITORY          TAG             IMAGE ID          CREATED           VIRTUAL SIZE
jsitech/kali-nmap   latest         fdde4bcbb3a7     57 seconds ago   661.1 MB
kalilinux/kali-linux-docker  latest        e49f6054690a     2 weeks ago      420.3 MB
```

Efectivamente, ahí tenemos la imagen que creamos a partir del contenedor donde instalamos nmap

Esto es una forma sencilla de crear una imagen, pero es posible que necesitemos un poco más de personalización en nuestra imagen, que tome ciertos parámetros y que esta tal vez ejecute algo cuando se usen para lanzar contenedores además de que nos faciliten la manera de compartir estas imágenes en un grupo de desarrollo por ejemplo. Este nivel de personalización podemos lograrlo con los Dockerfiles.



# DockerFiles

Un **DockerFile** es un documento de texto que contiene todos los comandos que queramos ejecutar en la línea de comandos para armar una imagen. Esta imagen se creará mediante el comando **docker build** que irá siguiendo las instrucciones.

Antes de hablar de los Dockerfiles vamos a hablar un poco del comando **docker build** que es el que ejecutaremos una vez tenemos las instrucciones a seguir en un archivo.

El comando **docker build** arma una imagen siguiendo las instrucciones de un DockerFile que se puede encontrar en el directorio actual o un repositorio. La creación de la imagen es ejecutada por el daemon de Docker. Es importante tener en cuenta que docker build le manda todo el contexto del directorio actual al daemon, por lo que es buena práctica colocar el DockerFile en un directorio limpio y agregar los archivos necesarios en ese directorio en caso de ser necesario.

El **Docker Daemon** corre las instrucciones en un Dockerfile línea por línea y va lanzando los resultados en pantalla. Un punto importante es que cada instrucción es ejecutada en nuevas imágenes, hasta que muestra el ID de la imagen resultante una vez finalizada las instrucciones, el daemon irá haciendo una limpieza automáticamente de las imágenes intermedias.

Nota: Dicho esto de que el Docker daemon va creando imágenes intermedias durante la creación de la imagen, si por ejemplo en un comando ejecutamos `cd /scripts/` y en otra línea le mandamos a ejecutar un script no va a funcionar, ya que ha lanzado otra imagen intermedia. Teniendo esto en cuenta, la manera correcta de hacerlo sería `cd /scripts/ ; ./install.sh`

**Ahora, viene otra pregunta, como nos ayuda todo esto de las Imágenes intermedias o Cache?** Si por alguna razón la creación de la imagen falla, ya sea por un comando mal digitado en el archivo, o lo que sea, cuando corregimos el Dockerfile, este no iniciará todo el proceso nuevamente, sino, que hará uso de las imágenes intermedias, y continuará la creación en el punto donde falló.

Ya que entendemos que es el **Dockerfile**, vamos a ver ahora el formato y las opciones que podemos pasarle.

Lo primero es que un DockerFile Inicia con una instrucción:

## FROM

FROM indica la imagen base que va a utilizar para seguir futuras instrucciones. Buscará si la imagen se encuentra localmente, en caso de que no, la descargará de internet.

### Sintaxis

```
FROM <imagen>
FROM <imagen>:<tag>
```

### Ejemplos

```
FROM centos:latest
```

El **tag** es opcional, en caso de que no la especifiquemos, el daemon de docker asumirá **latest** por defecto.

Vamos a seguir con las otras instrucciones

## MAINTAINER

Esta instrucción nos permite configurar datos del autor que genera la imagen.

### Sintaxis

```
MAINTAINER <nombre> <Correo>
```

### Ejemplo

```
MAINTAINER Jason Soto "jason_soto@jsitech.com"
```

## RUN

RUN tiene 2 formatos:

```
RUN <comando>, esta es la forma shell, /bin/sh -c
RUN ["ejecutable", "parámetro1", "parámetro2"], este es el modo ejecución
```

Esta instrucción ejecuta cualquier comando en una capa nueva encima de una imagen y hace un commit de los resultados. Esa nueva imagen intermedia es usada para el siguiente paso en el Dockerfile.

El modo ejecución nos permite correr comandos en imágenes bases que no cuenten con `/bin/sh` , nos permite además hacer uso de otra shell si así lo deseamos, ej: `RUN ["/bin/bash", "-c", "echo prueba"]` .

## ENV

ENV tiene 2 formas:

```
ENV <key><valor> , variable única a un valor
ENV <key><valor> ... , Múltiples variables a un valor
```

Esta instrucción configura las variables de ambiente, estos valores estarán en los ambientes de todos los comandos que sigan en el DockerFile. Pueden por igual ser sustituidos en una linea.

Estos valores persistirán al momento de lanzar un contenedor de la imagen creada. Pueden ser sustituida pasando la opción `-env` en docker run. Ej: `docker run -env <key>=<valor>`

## ADD

ADD tiene 2 formas:

```
ADD <fuente> ..<destino>
ADD ["fuente", ... "<destino>"]
```

Esta instrucción copia los archivos, directorios de una ubicación especificada en y los agrega al sistema de archivos del contenedor en la ruta especificada en .

Ejemplo:

```
ADD ./prueba.sh /var/tmp/prueba.sh
```

## EXPOSE

Esta instrucción le especifica a docker que el contenedor escucha en los puertos especificados en su ejecución. EXPOSE no hace que los puertos puedan ser accedidos desde el host, para esto debemos mapear los puertos usando la opción `-p` en docker run

Ejemplo:

```
EXPOSE 80 443
```

```
docker run centos:centos7 -p 8080:80
```

## CMD

CMD tiene tres formatos:

```
CMD ["ejecutable", "parámetro1", "parámetro2"] , este es el formato de ejecución  
CMD ["parámetro1", "parámetro2"] , parámetro por defecto para punto de entrada  
CMD comando parámetro1 parámetro2, modo shell
```

Solo puede existir una instrucción CMD en un DockerFile, si colocamos más de uno, solo el último tendrá efecto. El objetivo de esta instrucción es proveer valores por defecto a un contenedor. Estos valores pueden incluir un ejecutable u omitir un ejecutable que en dado caso se debe especificar un punto de entrada o ENTRYPOINT en las instrucciones.

Ejemplos:

Modo Shell

```
CMD echo "Esto es una prueba"
```

Si queremos ejecutar un comando sin un shell, debemos expresar el comando en formato JSON y dar la ruta del ejecutable. Es el formato recomendado.

```
CMD ["/usr/bin/service", "httpd", "start"]
```

Si lo que queremos es que el mismo ejecutable corra todo el tiempo, lo que necesitamos es un punto de entrada **ENTRYPOINT** en combinación con CMD. En caso de pasarle un comando mediante docker run, este correrá en vez del especificado por CMD.

## ENTRYPOINT

ENTRYPOINT tiene 2 formas:

```
ENTRYPOINT ["ejecutable", "parámetro1", "parámetro2"], forma de ejecución  
ENTRYPOINT comando parámetro1 parámetro2, forma shell
```

Cualquier argumento que pasemos en la línea de comandos mediante `docker run` serán anexados después de todos los elementos especificados mediante la instrucción `ENTRYPOINT`, y anulará cualquier elemento especificado con `CMD`. Esto permite pasar cualquier argumento al punto de entrada. Ejemplo:

```
docker run centos:centos7 -c
```

Esto le pasará el argumento `-c` al punto de entrada o `ENTRYPOINT` que especificamos en el `DockerFile`. Podemos anular las instrucciones del punto de entrada pasando la opción `entrypoint` al comando `docker run`.

Veamos un ejemplo de uso `ENTRYPOINT` en un `DockerFile`

```
FROM centos:centos7

ENTRYPOINT ["http", "-v "]

CMD ["-p", "80"]
```

En este ejemplo corremos `httpd`, en modo verbose, en el `ENTRYPOINT`, y los argumentos que entendamos puedan cambiar con `CMD`, puerto 80. Si quisiera correr el contenedor con `httpd` corriendo en el puerto 8080, solo tendría que ejecutar `docker run centos:centos7 -p 8080 .`

Es posible también hacer uso de scripts para ejecutar ciertas cosas, pero lo mantendré simple.

## VOLUME

Esta instrucción crea un punto de montaje con un nombre especificado y lo marca con un volumen montado externamente desde el host y otro contenedor. El valor pueden ser pasado en formato JSON o argumento plano.

```
VOLUME ["/var/tmp"]
VOLUME /var/tmp
```

El comando `docker run` inicializará el nuevo contenedor con cualquier data que exista en la ubicación dentro de la imagen base.

## USER

Esta instrucción configura el nombre de usuario a usar cuando se lanza un contenedor y para la ejecución de cualquier instrucción RUN, CMD o ENTRYPOINT. WORKDIR

```
WORKDIR ruta/de/trabajo
```

Esta instrucción configura el directorio de trabajo para cualquier instrucción **RUN, CMD, ENTRYPOINT, COPY o ADD** en un DockerFile. Puede ser usada varias veces dentro de un DockerFile. Si se da una ruta relativa, esta será la ruta relativa de la instrucción WORKDIR anterior.

Esta instrucción tiene la capacidad de resolver variables de ambiente previamente configuradas mediante la instrucción ENV. Ejemplo:

```
ENV rutadir /ruta

WORKDIR $rutadir
```

Aquí están las opciones que más usaremos y que veremos en muchos DockerFiles. Existen otras opciones, pero para mantenerlo un poco simple, ya hablaremos mas adelante de ellos. Aquí les copio un DockerFile que cree para crear una imagen de un servidor maligno usando como imagen base kali linux. Pueden verlo también en [GitHub](#).

```
#Docker Container With Maligno, Metasploit Payload Server

#Use Kali Linux Official Docker image

FROM kalilinux/kali-linux-docker

MAINTAINER Jason Soto "jason_soto@jsitech.com"

ENV DEBIAN_FRONTEND noninteractive

EXPOSE 443 22

#Updates Repo and installs Maligno Dependencies

RUN apt-get update; apt-get -y -force-yes install openssl ; apt-get -y install python-ipc

#Installs OpenSSH

RUN apt-get -y install openssh-server

#Installs Metasploit Framework

RUN apt-get -y install metasploit-framework

#Downloads And install Maligno Server

RUN wget -no-check-certificate http://www.encrypted.no/tools/maligno-2.4.tar.gz

RUN tar xzvf maligno-2.4.tar.gz; cd maligno-2.4/; ./install.sh

#Adds config XML with correct metasploit Path

ADD ./server_config.xml /maligno-2.4/server_config.xml
```

Ya cuando se tiene todo lo queremos definido solo es correr el comando docker build desde la ruta donde se encuentra el Dockerfile.

```
$ docker build -t jsitech/kali-maligno .
```

Nota: El archivo debe estar nombrado Dockerfile

# Mejores Prácticas DockerFiles

Vamos a ver algunas mejores prácticas y métodos recomendados por Docker Inc. y la comunidad para crear Dockerfiles fáciles de usar y efectivos.

## Usar un Archivo .dockerignore

Como hablamos es recomendable colocar cada DockerFile en un directorio limpio, y según la imagen que vayamos a crear pues agregamos los archivos que son necesarios. Es posible que tengamos algún archivo en el directorio que cumpla una función pero no queremos que sea agregado a la imagen, es por esto que debemos hacer uso de un archivo `.dockerignore` para que docker build excluya esos archivos durante la creación de la imagen.

Ejemplo de un .dockerignore

```
*/prueba*

*/*/prueba

prueba?
```

## No instale Paquetes Innecesarios

Para reducir la complejidad, dependencias, tiempo de creación y tamaño de la imagen resultante, se debe evitar instalar paquetes extras o innecesarios solo para tenerlos ahí y que no van a cumplir ninguna función en la imagen. Si algún paquete es necesario durante la creación de la imagen, lo mejor es desinstalarlo durante el proceso. Minimizar el número de capas

Debemos encontrar el balance entre la legibilidad del Dockerfile y minimizar el número de capa que utiliza.

## Correr un solo proceso por contenedor

En la mayoría de los casos, se debe correr solo un proceso en un contenedor, claro está esto dependerá del uso que le vamos a dar al contenedor. Desacoplar los componentes de una aplicación entre múltiples contenedores, hará mas sencillo escalar horizontalmente y reutilizar los contenedores.



## Organice argumentos de Múltiples Líneas

Cada vez que sea posible y para hacer más facil futuros cambios, organice argumentos que contengan múltiples líneas, esto evitará la duplicación de paquetes y hará que el archivo sea más fácil de leer. Ej:

```
RUN apt-get update && apt-get install -y \  
  
git \  
  
wget \  
  
apache2 \  
  
php5
```

## Cache de la Creación de Imágenes

Durante el proceso de la creación de imágenes, Docker seguirá las instrucciones del DockerFile en el orden especificado. En cada instrucción Docker busca una imagen existente en el caché que pueda reutilizar, en vez de generar imágenes duplicadas. En caso de que no quieran que Docker haga uso de imágenes en el caché simplemente es pasarle `--no-cache=true` al comando `docker build`. Sin embargo es importante que se entienda cuando docker encontrará o no una imagen que pueda reutilizar, en caso de que dejemos que use el caché.

- Comenzando con la imagen base que ya está en caché, la siguiente instrucción es comparada con todas las imágenes derivadas de esa imagen base para ver si una de ellas se creó usando exactamente las mismas instrucciones, si no, la caché es invalidada.
- En muchos de los casos con solo comparar el DockerFile con una imagen derivada debe ser suficiente. Sin embargo, hay ciertas instrucciones que requieren de mas verificación
- Para las instrucciones ADD y COPY, el contenido de los archivos en la imagen son examinadas y un Checksum es calculado para cada archivo. Durante la búsqueda en el caché el Checksum es comparado contra los Checksums de las imágenes ya creadas. Si algo cambió, la caché es invalidada.
- Si durante una instrucción, por ejemplo un `RUN apt-get update`, los archivos son actualizados dentro del contenedor, estos no serán examinados para determinar si existe algo en la caché, en este caso solo se tomará en cuenta el Comando para encontrar una imagen existente.
- Una vez la caché es invalidada, todas las instrucciones siguientes en el DockerFile

generarán nuevas imágenes y no harán uso de la caché.

## Mejores Prácticas para las Instrucciones en los DockerFiles

En este apartado vamos a ver algunas recomendaciones para algunas instrucciones que tenemos a la mano al momento de escribir un Dockerfile.

### FROM

Cada vez que sea posible, haga uso de imágenes oficiales para basar sus imágenes.

### RUN

Para hacer mas legible y entendible su DockerFile, divida comandos extensos y complicados en múltiples líneas.

#### Apt-get

Una de los comandos más utilizado con `RUN` son la instalación de Paquetes o actualizaciones, por ejemplo, `apt-get update` , `apt-get upgrade` . Debemos evitar el uso de `apt-get upgrade` O `dist-upgrade` , ya que los paquetes esenciales no llegan a ser actualizados en un contenedor no privilegiado. Si se encuentra con una imagen que tiene un paquete desactualizado, sencillamente haga un `apt-get install -y` para que lo actualice automáticamente.

Siempre combine `RUN apt-get update` y `apt-get install` en la misma instrucción, ya que de hacerlo por separado nos toparíamos con un tema de Caching. Recuerden todo lo que hablamos del uso de la caché, si Docker encuentra una imagen que pueda reutilizar y no necesariamente es la que acaba de generar con `apt-get update` , terminará con paquetes desactualizados.

Ejemplo de un correcto uso:

```
RUN apt-get update && apt-get install -y \  
  
git \  
  
apache2 \  
  
php5 \  
  
CMD
```

### CMD

La instrucción CMD debe ser usado para correr aplicaciones dentro de sus contenedores seguido de cualquier argumento. Debe ser usado siempre en la forma `CMD ["ejecutable", "parámetro1", "parámetro2" ...]` , Ejemplo: si el contenedor es para servir una aplicación web tendríamos algo así , `CMD ["apache2" , "-D", "FOREGROUND" ]`

En otros casos, debe ser utilizado para dar un shell interactivo. Ejemplo: `CMD ["python"]` o `CMD ["/bin/bash" ]` , de esta manera terminaríamos con una shell listo para trabajar. Tenemos la opción de solo pasarlo los parámetros a CMD en conjunto con ENTRYPOINT para debemos entender bien como funciona todo esa combinación.

## ADD y COPY

Aunque ADD y COPY tiene funcionalidades similares, COPY es el preferido. COPY soporta el copiado básico de archivos locales al contenedor, mientras ADD tiene otras funcionalidades como la extracción local de archivos tar y el soporte a URL's remotos. Así que en estos casos que se requiera una auto extracción de un archivo hacia el contenedor, es mejor hacer uso de ADD.

## ENV

Para hacer mas sencillo la ejecución de aplicaciones dentro del contenedor, podemos hacer uso de ENV para actualizar las variables de las rutas por ejemplo, de la aplicación que el contenedor ejecuta. Ejemplo: `ENV PATH /opt/app/bin:$PATH` , esto permitirá que `CMD ["app"]` funcione.

## USER

Si un servicio puede correr sin privilegios, use USER para cambiarlo a un usuario no privilegiado. Lo primero es crear el usuario y el grupo en una instrucción del DockerFile con RUN.

## WORKDIR

Para claridad y legibilidad, siempre se debe hacer uso de rutas absolutas para el directorio de trabajo WORKDIR, y siempre debe ser usado y no hacer uso de `RUN cd /ruta/` ya que igual puede traer temas con el caching y problemas para hacerle el troubleshooting de lugar.

Bueno, creo que en este punto punto vamos bien encaminados con las creaciones de imágenes y mas mediante los DockerFiles.

# Comando Docker Run

A lo largo de esta guía hemos mencionado algunas veces a `docker run`, pero solo lo básico. Dado que es un comando que usaremos mucho, obviamente, quiero dedicarle un poco de tiempo ya que es mucho lo que podemos hacer con el, al momento que estamos lanzando un contenedor.

```
$ docker run [opciones] [imagen] [comandos] [argumentos]
```

Cuando ejecutamos `docker run` debemos especificar una imagen que usaremos de base al momento de lanzar el contenedor, otro punto es que las opciones pueden sustituir casi todos los valores predeterminados configurados en la ejecución, ver la entrada de los Dockerfile para que vean a que me refiero.

## Opciones

### Detached y Foreground

Cuando lanzamos el contenedor por defecto corre en modo foreground y atacha la consola en la entrada y salida estándar del proceso. Puede pretender incluso ser un tty que es lo que esperan muchos de los ejecutables. Todo esto es configurable

```
-a=[]          : Atacha a `STDIN`, `STDOUT` o `STDERR`  
  
-t=false      : Asigna un pseudo-tty  
  
-sig-proxy=true : Captura todas las señales y las envía al proceso (modo non-TTY)  
  
-i=false      : Mantiene STDIN abierto aun no esté atachado
```

Si no se especifica nada con `-a`, Docker atachara todos los streams.

#### Ejemplo:

```
$ docker run -a stdout -i -t centos:centos7
```

Si pasamos la opción `-d`, el contenedor se lanzará en modo Detached. En este modo cuando el proceso raíz haya finalizado el contenedor se detiene. Debemos tener pendiente que si le pasamos un comando al contenedor en la ejecución, una vez el comando se

ejecute, se detendrá. Es por esto que siempre debemos configurar muy bien lo que deseamos en los Dockerfiles, específicamente hablando los `EntryPoint`s y `CMD`.

**Ejemplo:**

```
$ docker run -i -t -d centos:centos /bin/bash
```

## Identificando los Contenedores

Cuando lanzamos un contenedor el daemon de Docker lo identifica con 3 valores:

```
UUID Largo  
UUID Corto  
Nombre
```

Docker por defecto le asigna un nombre aleatorio al contenedor. Este nombre lo podemos utilizar para interactuar de diversas maneras, algunos ejemplos serían, obtener algunos detalles del contenedor, detenerlo, reiniciarlo y mas. Este nombre lo podemos obtener cuando ejecutamos un `docker ps`. Este nombre podemos configurarlo como deseemos, de esta manera podemos identificar el contenedor con sus funciones por ejemplo. Esto lo hacemos con la opción `--name`.

**Ejemplo:**

```
$ docker run -i -t -d --name="webserver" -p 80:80 centos:webimage /bin/bash
```

## Configuración de Red

Por defecto los contenedores tienen las conexiones de redes habilitadas y pueden hacer conexiones salientes sin restricciones. Podemos deshabilitar esto por completo pasando la opción `--net none`. En estos casos las conexiones de entrada y salida se hacen por archivos y los streams estándar.

```
-dns=[] : Configurar dns al contenedor

-net="bridge" : Conecta el contenedor a una red

    'bridge': crea un nuevo stack para el contenedor en el puente de docker

    'none': Sin conexión de red

    'container:<name|id>': Reutiliza el stack de conexiones de otro contenedor

    'host': Usa el stack de conexiones del host en el contenedor

    'NETWORK': conecta el contenedor a una red creada por el usuario mediante el comando

-add-host="" : Agrega una línea a /etc/hosts (host:IP) -mac-address="" : Configura la MAC
```

**Nota:** Cuando usamos la opción host le da al contenedor acceso por completo a los servicios del sistema local por lo que se considera Inseguro.

### Algunos Ejemplos:

#### Configurando los DNS en el contenedor

```
$ docker run -i -t -dns="8.8.8.8" centos:centos7 /bin/bash
```

#### Agregando una entrada en /etc/hosts del contenedor

```
$ docker run -i -t -dns="8.8.8.8" -add-host webserver:10.0.0.5 centos:centos7 /bin/bash
```

#### Crear una red y conectar el contenedor a ella

```
$ docker network create -d overlay mi-red

$ docker run -net=mi-red -i -t -d centos:centos7
```

## PID

```
-pid="" : Configura el espacio de nombre PID para el contenedor

'host' : Utiliza el espacio de nombre del host dentro del contenedor
```

### Ejemplos:

```
$ docker run --pid=host centos7 strace -p 123
```

Esto le permite al contenedor ver todos los procesos en el host. Es útil cuando se desea correr procesos de depuración o lo que necesiten en el host pero desean hacerlo desde el contenedor. Políticas de Reinicio

Usando `--restart` podemos especificar una política de reinicio de como un contenedor debe o no debe reiniciarse. Docker Soporta las Políticas:

```
no : No reiniciarse cuando se detiene un contenedor, este es el predeterminado  
on failure : Reiniciar el contenedor cuando se detiene por un status de salida non-zero.  
Always: Siempre reinicia el contenedor independiente del status de salida. Inicialá el co  
Unless-stopped: Reiniciar siempre el contenedor independiente del status de salida, pero
```

Ejemplo:

```
$ docker run --restart=always centos:centos7  
  
$ docker run --restart=on-failure:20 centos:centos7
```

## Limitando los Recursos usados por los contenedores

<b>-m, --memory=""</b>	<b>Límite Memoria (formato: [], donde unidad= b, k, m or g)</b>
<b>--memory-swap=""</b>	Total límite de memoria (memory + swap, formato: [], donde unidad = b, k, m or g)
<b>--memory-reservation=""</b>	Límite flexible de memoria (formato: [], donde unidad= b, k, m or g)
<b>--kernel-memory=""</b>	Límite memoria Kernel (formato: [], donde unidad= b, k, m or g)
<b>-c, --cpu-shares=0</b>	CPU (peso relativo)
<b>--cpu-period=0</b>	Limitar Período CPU CFS (Completely Fair Scheduler)
<b>--cpuset-cpus=""</b>	CPU's en donde permitir ejecución (0-3, 0,1)
<b>--cpuset-mems=""</b>	Nodos de memoria en donde permitir ejecución (0-3, 0,1)
<b>--cpu-quota=0</b>	Limitar cuota CPU CFS (Completely Fair Scheduler)
<b>--blkio-weight=0</b>	Bloquear Peso IO (Peso relativo) aceptar valor de peso entre 10 y 1000.
<b>--oom-kill-disable=false</b>	Desahabilitar OOM Killer para el contenedor
<b>--memory-swappiness=""</b>	Configurar el comportamiento d Swappiness del contenedor

## Límites de uso de Memoria

<b>memory=inf, memory-swap=inf (default)</b>	<b>No hay límites de memoria para el contenedor</b>
memory=L<inf, memory-swap=inf	(Especificar memoria y configurar memory-swap como -1) El contenedor no puede usar más de la memoria especificada por L, pero puede usar toda la memoria swap que necesite
memory=L<inf, memory-swap=2*L	(Especificar memoria sin memory-swap) El contenedor no puede usar mas de la memoria especificada por L, y usar el doble como swap
memory=L<inf, memory-swap=S<inf, L<=S	(Especificar memoria y memory-swap) El contenedor no puede usar más de la memoria especificada por L, y la memoria Swap especificada por S

### Ejemplos:

```
$ docker run -i -t -d -m 500m centos:centos7 /bin/bash
```

```
$ docker run -i -t -d -m 500m --memory-swap 2G centos:centos7 /bin/bash
```



## Otros Ejemplos:

```
$ docker run -i -t -d --cpuset-cpus="1" centos:centos7 /bin/bash

$ docker run -i -t -d --cpuset-mems="0-2" centos:centos7 /bin/bash

$ docker run -i -t -d --memory-swappiness=0 centos:centos7 /bin/bash
```

**--entrypoint=""** : Sobreescribe el entrypoint especificado en el Dockerfile

## Ejemplo:

```
$ docker run -i -t -d --entrypoint /bin/bash centos:centos7
```

**--expose=[]** : Expone un puerto o un rango en el contenedor

## Ejemplos:

```
$ docker run -i -t -d --expose=80 -p 80:80 centos:centos7 /bin/bash
```

Si la imagen creada ya tiene por defecto los puertos expuesto solo debemos mapear los puertos

```
$ docker run -i -t -d -p 80:80 centos:centos7 /bin/bash

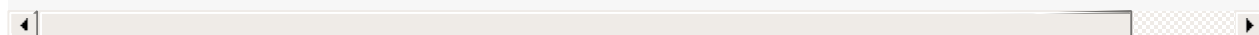
$ docker run -i -t -d --expose=80 -P centos:centos7 /bin/bash
```

**Nota:** **-P** mapea todos los puertos expuestos en el contenedor a puertos aleatorios en el Host.

**-v=[]** : Enlaza un Directorio en el host al contenedor o crea un volumen

## Ejemplos:

```
$ docker run -i -t -d -v /home/jsitech/sitio/:/var/www/html/ -p 80:80 centos:centos7 /bin
```



**-u=""** : Configura el usuario por defecto dentro del contenedor

## Ejemplos:

```
$ docker run -i -t -d --expose=80 -p 80:80 -u jsitech centos:centos7 /bin/bash
```

Estas serían las opciones que mas estaríamos utilizando con `docker run` . Acá les dejo todas las opciones de docker run, que pueden visualizarlas ejecutando un `docker run --help` .

```

[jlsitech@Lap-Sec Proyectos]$ docker run --help
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

-a, --attach=[]                Attach to STDIN, STDOUT or STDERR
--add-host=[]                  Add a custom host-to-IP mapping (host:ip)
--blkio-weight=0               Block IO (relative weight), between 10 and 1000
--cpu-shares=0                 CPU shares (relative weight)
--cap-add=[]                   Add Linux capabilities
--cap-drop=[]                  Drop Linux capabilities
--cgroup-parent=[]             Optional parent cgroup for the container
--cidfile=[]                   Write the container ID to the file
--cpu-period=0                 Limit CPU CFS (Completely Fair Scheduler) period
--cpu-quota=0                  Limit CPU CFS (Completely Fair Scheduler) quota
--cpu-set-cpus=[]              CPUs in which to allow execution (0-3, 0,1)
--cpu-set-mems=[]              MEMs in which to allow execution (0-3, 0,1)
-d, --detach=false             Run container in background and print container ID
--device=[]                    Add a host device to the container
--disable-content-trust=true    Skip image verification
--dns=[]                       Set custom DNS servers
--dns-opt=[]                   Set DNS options
--dns-search=[]                Set custom DNS search domains
-e, --env=[]                   Set environment variables
--entrypoint=[]                Overwrite the default ENTRYPOINT of the image
--env-file=[]                  Read in a file of environment variables
--expose=[]                    Expose a port or a range of ports
--group-add=[]                 Add additional groups to join
-h, --hostname=[]              Container host name
--help=false                   Print usage
-i, --interactive=false        Keep STDIN open even if not attached
--ipc=[]                       IPC namespace to use
--kernel-memory=[]             Kernel memory limit
-l, --label=[]                 Set meta data on a container
--label-file=[]                Read in a line delimited file of labels
--link=[]                      Add link to another container
--log-driver=[]                Logging driver for container
--log-opt=[]                   Log driver options
--lxc-conf=[]                  Add custom lxc options
-m, --memory=[]                Memory limit
--mac-address=[]               Container MAC address (e.g. 92:d0:c6:0a:29:33)
--memory-reservation=[]        Memory soft limit
--memory-swap=[]               Total memory (memory + swap), '-1' to disable swap
--memory-swappiness=-1         Tuning container memory swappiness (0 to 100)
--name=[]                      Assign a name to the container
--net=default                  Set the Network for the container
--oom-kill-disable=false        Disable OOM Killer
-p, --publish-all=false       Publish all exposed ports to random ports
-p, --publish=[]               Publish a container's port(s) to the host
--pid=[]                       PID namespace to use
--privileged=false             Give extended privileges to this container
--read-only=false              Mount the container's root filesystem as read only
--restart=no                    Restart policy to apply when a container exits
--rm=false                     Automatically remove the container when it exits
--security-opt=[]              Security Options
--sig-proxy=true                Proxy received signals to the process
--stop-signal=SIGTERM          Signal to stop a container, SIGTERM by default
-t, --tty=false                Allocate a pseudo-TTY
-u, --user=[]                   Username or UID (format: <name|uid[:<group|gid>])
--ulimit=[]                    Ulimit options
--uts=[]                       UTS namespace to use
-v, --volume=[]                Bind mount a volume
--volume-driver=[]             Optional volume driver for the container
--volumes-from=[]              Mount volumes from the specified container(s)
-w, --workdir=[]               Working directory inside the container

```

## Otros Comandos

En esta entrada lo que veremos son los otros comandos que podemos usar para interactuar con los contenedores. Lo veremos a modo de **Cheat Sheet**.

Vamos a ver algunos de ellos.

Si corremos docker en la linea de comandos nos devolverá una lista de las opciones

```
$docker
```

```

root@~# docker
Usage: docker [OPTIONS] COMMAND [arg...]

A self-sufficient runtime for linux containers.

Options:
  --add-registry=[]          Registry to query before a public one
  --api-cors-header=         Set CORS headers in the remote API
  -b, --bridge=              Attach containers to a network bridge
  --bip=                     Specify network bridge IP
  --block-registry=[]        Don't contact given registry
  --confirm-def-push=true    Confirm a push to default registry
  -D, --debug=false          Enable debug mode
  -d, --daemon=false         Enable daemon mode
  --default-gateway=         Container default gateway IPv4 address
  --default-gateway-v6=     Container default gateway IPv6 address
  --default-ulimit=[]        Set default ulimits for containers
  --dns=[]                   DNS server to use
  --dns-search=[]            DNS search domains to use
  -e, --exec-driver=native   Exec driver to use
  --exec-opt=[]              Set exec driver options
  --exec-root=/var/run/docker Root of the Docker execdriver
  --fixed-cidr=              IPv4 subnet for fixed IPs
  --fixed-cidr-v6=           IPv6 subnet for fixed IPs
  -G, --group=docker         Group for the unix socket
  -g, --graph=/var/lib/docker Root of the Docker runtime
  -H, --host=[]              Daemon socket(s) to connect to
  -h, --help=false           Print usage
  --icc=true                 Enable inter-container communication
  --insecure-registry=[]     Enable insecure registry communication
  --ip=0.0.0.0               Default IP when binding container ports
  --ip-forward=true          Enable net.ipv4.ip forward
  --ip-masq=true             Enable IP masquerading
  --iptables=true            Enable addition of iptables rules
  --ipv6=false               Enable IPv6 networking
  -l, --log-level=info       Set the logging level
  --label=[]                 Set key=value labels to the daemon
  --log-driver=json-file     Default driver for container logs
  --log-opt=map[]            Set log driver options
  --mtu=0                    Set the containers network MTU
  -p, --pidfile=/var/run/docker.pid Path to use for daemon PID file
  --registry-mirror=[]       Preferred Docker registry mirror
  -s, --storage-driver=       Storage driver to use
  --selinux-enabled=false     Enable selinux support
  --storage-opt=[]            Set storage driver options
  --tls=false                 Use TLS; implied by --tlsverify
  --tlscacert=~/.docker/ca.pem Trust certs signed only by this CA
  --tlscert=~/.docker/cert.pem Path to TLS certificate file
  --tlskey=~/.docker/key.pem   Path to TLS key file
  --tlsverify=false           Use TLS and verify the remote
  --userland-proxy=true       Use userland proxy for loopback traffic
  -v, --version=false         Print version information and quit

Commands:
  attach  Attach to a running container
  build   Build an image from a Dockerfile
  commit  Create a new image from a container's changes
  cp      Copy files/folders from a container's filesystem to the host path
  create  Create a new container

```

Vamos a ver ahora que hacen cada uno

## docker ps

Muestra los contenedores que están corriendo, si le pasamos -a nos mostrará también los contenedores que hemos detenido

**Ejemplo:**

```

$ docker ps

$ docker ps -a

```

```
[root@ ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
c171f37f3224   jsitech/kali-nmap  "/bin/bash"            14 seconds ago Up 12 seconds          naughty_colden
[root@ ~]#
```

```
[root@ ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
f1c6b0c1cd3e   docker.io/cirros:latest  "/bin/bash"            25 minutes ago Exited (137) 15 seconds ago          reverent_bardeen
8184b0c4199e   docker.io/ubuntu:latest  "/bin/bash"            13 hours ago   Dead
[root@ ~]#
```

## docker attach

Se conecta a un contenedor que esta corriendo, podemos hacerlo por el nombre o por el ID de contenedor

**Ejemplo:**

```
$ docker attach c171f37f3224
$ docker attach naughty_colden
```

## docker start

Reinicia un contenedor detenido. Lo podemos hacer por el Nombre o el ID

## docker stop

Detiene un contenedor

**Ejemplo:**

```
$ docker start naughty_colden
```

```
[root@ ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
c171f37f3224   jsitech/kali-nmap  "/bin/bash"            7 minutes ago Up 7 minutes          naughty_colden
0a27073301ea   jsitech/kali-malicious:latest  "/bin/bash"            2 weeks ago   Exited (0) 15 minutes ago          stupid_robot
863d4d816b8    kali/kali-linux-docker  "/bin/bash"            2 weeks ago   Exited (0) 15 minutes ago          nccost_k17ch
6e4390c5935a   kali/kali-linux-docker  "/bin/bash"            2 weeks ago   Exited (0) 2 weeks ago          naughty_morse
[root@ ~]# docker start naughty_morse
naughty_morse
[root@ ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
c171f37f3224   jsitech/kali-nmap  "/bin/bash"            7 minutes ago Up 7 minutes          naughty_colden
6e4390c5935a   kali/kali-linux-docker  "/bin/bash"            2 weeks ago   Up 4 seconds          naughty_morse
[root@ ~]#
```

## docker diff

Lista los cambios hechos en el sistema de archivos de un contenedor. Hay 3 eventos que muestra

**A** – Agregado

**D** – Eliminado

**C** – Cambio

**Ejemplo:**

```
$ docker diff naughty_morse
```

```
[root@ ~]# docker diff naughty_colden
[root@ ~]# docker diff naughty_morse
C /root
A /root/.bash_history
C /run
A /run/secrets
C /var
C /var/lib
C /var/lib/apt
C /var/lib/apt/lists
C /var/lib/apt/lists/partial
A /var/lib/apt/lists/partial/http.kali.org_kali_dists_sana_InRelease.reverify
A /var/lib/apt/lists/partial/security.kali.org_kali-security_dists_sana_updates_InRelease.reverify
D /var/lib/apt/lists/security.kali.org_kali-security_dists_sana_updates_InRelease
D /var/lib/apt/lists/http.kali.org_kali_dists_sana_InRelease
C /var/cache
C /var/cache/apt
A /var/cache/apt/pkgcache.bin
A /var/cache/apt/srcpkgcache.bin
[root@ ~]#
```

## docker events

Muestra eventos en tiempo real del estado de los contenedores.

**Ejemplo:**

```
$ docker events
```

```
[root@ ~]# docker events
2015-11-28T13:10:46.600000000-05:00 6e49d0c5935ea48f2250dedd6daedea953c54d97b17d5cace171b97a63ad2d49: (from kalilinux/kali-linux-docker) die
2015-11-28T13:10:46.600000000-05:00 6e49d0c5935ea48f2250dedd6daedea953c54d97b17d5cace171b97a63ad2d49: (from kalilinux/kali-linux-docker) stop
2015-11-28T13:10:53.600000000-05:00 6e49d0c5935ea48f2250dedd6daedea953c54d97b17d5cace171b97a63ad2d49: (from kalilinux/kali-linux-docker) start
```

## docker exec

Ejecuta un comando en un contenedor activo

**Ejemplo:**

```
$ docker exec -d kali/kali-nmap touch /tmp/file
```

## docker inspect

Muestra informaciones de bajo nivel del contenedor o la imagen

### Ejemplo

```
$ docker inspect naughty_morse
```

```
$ docker inspect naughty_morse | grep IPAddress
```

```
[root@ ~]# docker inspect naughty_morse
[
  {
    "Id": "6e49d0c5935ea48f2250dedd5daedea053c54d97b17d5caca171b97a63ad2d49",
    "Created": "2015-11-13T10:02:23.01703404Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 3065,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2015-11-28T18:10:53.598459523Z",
      "FinishedAt": "2015-11-28T18:10:46.835583694Z"
    },
    "Image": "8c9a4099d037398e1541bdaf0cf42f36ba19e9091de7797764b6fa8ad59f0960",
    "NetworkSettings": {
      "Bridge": "",
      "EndpointID": "9d177318b95de7d837c98aa47b05f81ba39969289703bb3308417dfbff2ac574",
      "Gateway": "172.17.42.1",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "HairpinMode": false,
      "IPAddress": "172.17.0.3",
      "IPPrefixLen": 16,
      "IPv6Gateway": "",
      "LinkLocalIPv6Address": "",
      "LinkLocalIPv6PrefixLen": 0,
      "MacAddress": "02:42:ac:11:00:03",
      "NetworkID": "9bb367d38b798c44e2c5ced489b53a223a656329b2e209c04382c0341845e0f9",
      "PortMapping": null,

```

```
[root@ ~]# docker inspect naughty_morse | grep IPAddress
  "IPAddress": "172.17.0.3",
  "SecondaryIPAddresses": null,
[root@ ~]#
```

## docker export

Exporta el contenido del sistema de archivo de un contenedor a un archivo tar

### Ejemplo:

```
$ docker export naughty_morse > kalicont.tar
```

```
$ docker export -o kalicont.tar naughty_morse
```

## docker search

Busca una imagen en el registro de docker

## Ejemplo:

```
$ docker search ubuntu
```

```
[root@ ~]# docker search ubuntu
INDEX      NAME                DESCRIPTION                STARS    OFFICIAL    AUTOMATED
docker.io  docker.io/ubuntu    Ubuntu is a Debian-based Linux operating s...  2782     [OK]
docker.io  docker.io/ubuntu-upstart Upstart is an event-based replacement for ...  48       [OK]
docker.io  docker.io/dorowu/ubuntu-desktop-lxde-vnc    Ubuntu with openssh-server and NoVNC on po...  28
docker.io  docker.io/sequenceiq/hadoop-ubuntu          An easy way to try Hadoop on Ubuntu          25       [OK]
docker.io  docker.io/torusware/speedus-ubuntu           Always updated official Ubuntu docker imag...  25       [OK]
docker.io  docker.io/ubuntu-debootstrap                  debootstrap --variant=minbase --components...  28       [OK]
docker.io  docker.io/tleyden5iwx/ubuntu-cuda            Ubuntu 14.04 with CUDA drivers pre-installed  18       [OK]
docker.io  docker.io/rastashoop/ubuntu-sshd             Dockerized SSH service, built on top of of...  15       [OK]
docker.io  docker.io/n3ziniuka5/ubuntu-oracle-jdk       Ubuntu with Oracle JDK. Check tags for ver...  5       [OK]
docker.io  docker.io/loft/armhf-ubuntu                  [ABR] Ubuntu Docker images for the ARMv7(a...  4       [OK]
docker.io  docker.io/nuagoboc/ubuntu                   Simple always updated Ubuntu docker images...  4       [OK]
docker.io  docker.io/nimmis/ubuntu                     This is a docker images different LTS vers...  3       [OK]
docker.io  docker.io/maxexcloo/ubuntu                  Docker base image built on Ubuntu with Sup...  2       [OK]
docker.io  docker.io/densuke/ubuntu-jp-remix            Ubuntu Linuxの日本語remix風味です            1       1          [OK]
docker.io  docker.io/seetheprogress/ubuntu              Ubuntu image provided by seetheprogress us...  1       [OK]
docker.io  docker.io/sylvainlasnier/ubuntu              Ubuntu 15.04 root docker images with commo...  1       [OK]
docker.io  docker.io/densuke/ubuntu-supervisor          densuke/ubuntu-jp-remix:trusty 上で supe...  0       [OK]
docker.io  docker.io/esycat/ubuntu                     Ubuntu LTS                                     0       [OK]
docker.io  docker.io/konstruktoid/ubuntu                Ubuntu base image                             0       [OK]
docker.io  docker.io/partlab/ubuntu                     Simple Ubuntu docker images.                  0       [OK]
docker.io  docker.io/rallias/ubuntu                     Ubuntu with the needful                        0       [OK]
docker.io  docker.io/teamrock/ubuntu                    TeamRock's Ubuntu image configured with AW...  0       [OK]
docker.io  docker.io/tvaughan/ubuntu                    https://github.com/tvaughan/docker-ubuntu    0       [OK]
docker.io  docker.io/vicamo/ubuntu-phablet-jiexi        Dockerfile for developing Ubuntu JieXi PDK.    0       [OK]
docker.io  docker.io/zoni/ubuntu                        0       [OK]
```

## docker pull

Descarga una imagen del registro de Docker

## Ejemplo:

```
$ docker pull docker.io/ubuntu
```

## docker history

Muestra el historial de una imagen

## Ejemplo:

```
$ docker history jsitech/kali-nmap
```

```
[root@ ~]# docker history jsitech/kali-nmap
IMAGE          CREATED          CREATED BY          SIZE          COMMENT
a4059a9396ad   2 weeks ago     /bin/bash          44.39 MB
8c9a4099d037   4 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]
35e347722e9f   4 weeks ago     /bin/sh -c apt-get -y update && apt-get -y di 119.7 MB
c97a4e8cedc9   4 weeks ago     /bin/sh -c #(nop)  ENV DEBIAN_FRONTEND=noninte 0 B
599d44c189d4   4 weeks ago     /bin/sh -c echo "deb http://http.kali.org/ka 278 B
799ca175e184   4 weeks ago     /bin/sh -c #(nop)  MAINTAINER steev@kali.org 0 B
93d580d5b6d4   3 months ago    /bin/sh -c #(nop)  MAINTAINER steev@kali.org 0 B
b4be4f2256bf   3 months ago    /bin/sh -c #(nop)  MAINTAINER steev@kali.org 0 B
e1bf89993bf    3 months ago    /bin/sh -c #(nop)  MAINTAINER steev@kali.org 300.6 MB
Imported from -
```

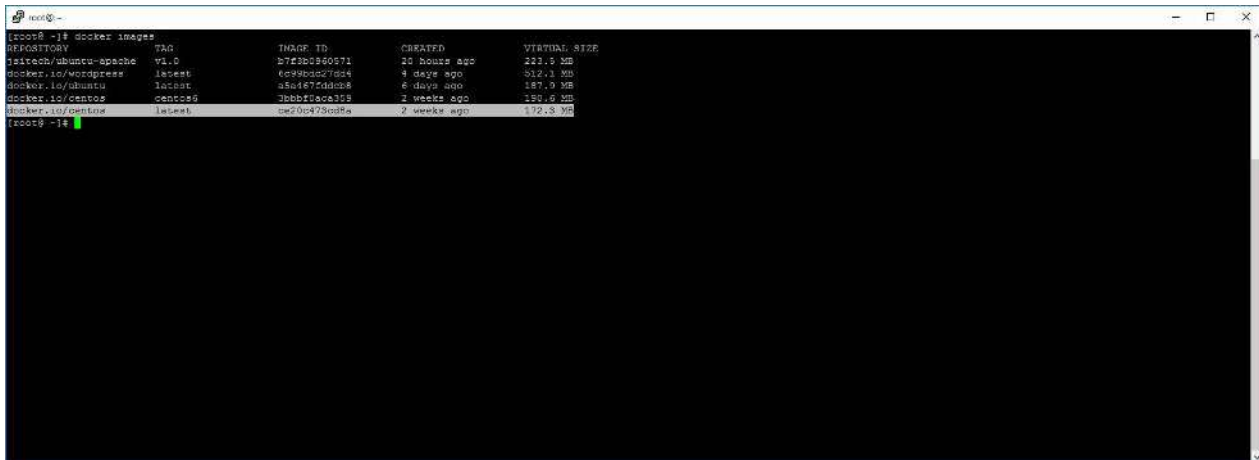
## docker images

Muestra las imágenes que tenemos disponible localmente



## Ejemplo:

```
$ docker images
```



## docker kill

Detiene un contenedor enviando un SIGKILL.

## Ejemplo:

```
$ docker kill naughty_morse
```

## docker load

Carga una imagen desde un archivo tar

## Ejemplo:

```
$ docker load -input kalicont.tar
```

```
$ docker load < kalicont.tar
```

## docker rmi

Elimina una o mas imagen. Si la imagen a borrar tiene un contenedor dependiente, se deben eliminar esos contenedores con docker rm para poder borrar la imagen

## Ejemplo:

```
$ docker rmi jsitech/kali-nmap
```

## docker rm

Elimina uno o mas contenedores.

### Ejemplo:

```
$ docker rm 6e49d0c5935e
```

El **id** lo podemos obtener con un `docker ps -a`

## docker login

Se registra o loguea en un servidor de registro de Docker, si no se especifica el servidor “<https://index.docker.io/v1/>” es el seleccionado por defecto

### Ejemplo:

```
$ docker login 10.0.0.10:8080
```

```
$ docker login -e jason_soto@jsitech.com -p 123 -u jsitech 10.0.0.10:8080
```

## docker logout

Se desconecta del servidor de registro de docker.

## Docker stats

Muestra el uso de recursos de los contenedores

### Ejemplo:

```
$ docker stats naughty_colden
```

```
CONTAINER          CPU %       MEM USAGE/LIMIT  MEM %       NET I/O
naughty_colden     0.00%       3.822 MB/1.042 GB 0.37%       2.772 kB/828 B
```

## docker build

Crea una imagen a partir de un DockerFile

### Ejemplo:

```
$ docker build jsitech/kali-nmap .  
  
$ docker build jsitech/kali-nmap -f /ruta/Dockerfile
```

## docker commit

Crea una imagen a partir de un contenedor

**Ejemplo:**

```
$ docker commit c171f37f3224 jsitech/kali-nmap
```

## docker pause

Congela todos los procesos en un contenedor

**Ejemplo:**

```
$ docker pause naughty_colden
```

## docker rename

Renombra un contenedor con el nombre deseado

```
$ docker rename naughty_colden jsitech_kali
```

```
^C[root@ ~]# docker ps  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES  
c171f37f3224   jsitech/kali-nmap  "/bin/bash"             49 minutes ago Up 49 minutes          naughty_colden  
6e49d8c5935e   kalilinux/kali-linux-docker  "/bin/bash"             2 weeks ago   Up 34 minutes          naughty_morse  
[root@ ~]# docker rename naughty_colden jsitech_kali  
[root@ ~]# docker ps  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES  
c171f37f3224   jsitech/kali-nmap  "/bin/bash"             51 minutes ago Up 51 minutes          jsitech_kali  
6e49d8c5935e   kalilinux/kali-linux-docker  "/bin/bash"             2 weeks ago   Up 36 minutes          naughty_morse  
[root@ ~]#
```

## Daemon de Docker

<code>--api-cors-header=""</code>	Configura los Headers CORS a un API remoto
<code>-b, --bridge=""</code>	Conecta los contenedores a un puente de red
<code>--bip=""</code>	Especificar la IP del puente de red
<code>-D, --debug=false</code>	Habilitar modo de depuración
<code>--default-gateway=""</code>	Gateway por defecto IPv4
<code>--default-gateway-v6=""</code>	Gateway por defecto IPv6
<code>--dns=[]</code>	Servidor DNS a utilizar
<code>--dns-search=[]</code>	Dominios de búsqueda DNS a utilizar
<code>--default-ulimit=[]</code>	Configurar Ulimit a los contenedores
<code>-e, --exec-driver="native"</code>	Driver Exec a utilizar
<code>--exec-opt=[]</code>	Configurar opciones Driver Exec
<code>--exec-root="/var/run/docker"</code>	Ruta del execdriver de docker
<code>--fixed-cidr=""</code>	Subred Ipv4 para direcciones fijas
<code>--fixed-cidr-v6=""</code>	Subred Ipv6 para direcciones fijas
<code>-G, --group="docker"</code>	Grupo para el socket de Unix
<code>-g, --graph="/var/lib/docker"</code>	Ruta del runtime de Docker
<code>-H, --host=[]</code>	Sockets de Conexión del Daemon de Docker
<code>-h, --help=false</code>	Mostrar ayuda
<code>--icc=true</code>	Habilitar comunicación entre contenedores
<code>--insecure-registry=[]</code>	Habilitar comunicación insegura al registro
<code>--ip=0.0.0.0</code>	IP por defecto al mapear los puertos
<code>--ip-forward=true</code>	Habilitar net.ipv4.ip_forward
<code>--ip-masq=true</code>	Habilitar IP masquerading
<code>--iptables=true</code>	Habilitar Reglas de Iptables
<code>--ipv6=false</code>	Habilitar comunicaciones IPv6
<code>-l, --log-level="info"</code>	Configurar nivel de logging
<code>--label=[]</code>	Configurar etiqueta al daemon
<code>--log-driver="json-file"</code>	Driver predeterminado para logs
<code>--log-opt=[]</code>	Opciones para el Driver de logs
<code>--mtu=0</code>	configurar MTU
<code>-p, --pidfile="/var/run/docker.pid"</code>	Ruta PID daemon
<code>--registry-mirror=[]</code>	Registro de Docker preferido
<code>-s, --storage-driver=""</code>	Driver de almacenamiento a utilizar
<code>--selinux-enabled=false</code>	Habilitar Soporte para Selinux
<code>--storage-opt=[]</code>	Opciones para driver de almacenamiento
<code>--tls=false</code>	Usar TLS; <code>--tlsverify</code>
<code>--tlscacert=~/.docker/ca.pem"</code>	Confiar solo en este CA
<code>--tlscert=~/.docker/cert.pem"</code>	Ruta Certificado
<code>--tlskey=~/.docker/key.pem"</code>	Ruta llaves
<code>--tlsverify=false</code>	Usar TLS y verificar el remoto
<code>--userland-proxy=true</code>	usar proxy para tráfico Loopback

## Crear un Puente de red y conectar los contenedores a el.

### Pasos ejecutados en Centos

```
$ ip link add br10 type bridge  
  
$ ip addr add 10.0.100.1/24 dev br10  
$ ip link set br10 up  
  
$ docker -d -b br10 &
```

Al ejecutar estos pasos, al momento de lanzar los contenedores tomarán una IP del rango que especificamos.

# Docker Compose

**Docker Compose** es una herramienta para definir y correr aplicaciones multicontenedores. Hacemos uso de un archivo `docker-compose` para configurar los servicios que necesita la aplicación. Luego haciendo uso de un simple comando, creamos los contenedores necesarios e iniciamos todos los servicios especificados en la configuración.

**Compose** es bueno para desarrollo, pruebas y flujos de trabajo de integración continua.

En el caso de los desarrolladores, tienen la habilidad de correr las aplicaciones en un ambiente aislado e interactuar con la aplicación. El **archivo de compose** provee una forma de documentar y configurar todas las dependencias de la aplicación y al final solo tiene que hacer uso de un comando para subir todo el ambiente.

Otro punto importante en los procesos de Despliegue e integración continua es el banco de pruebas. Este banco de pruebas requiere de un ambiente donde realizar las pruebas. Compose provee una manera de crear y destruir los ambientes de prueba aislados para ese banco de pruebas. Definiendo todo el ambiente en un archivo compose se puede crear y destruir los ambientes con pocos comandos.

## Instalando Docker Compose

Antes de proceder con la instalación de compose es bueno consultar la página de lanzamiento <https://github.com/docker/compose/releases> y ajustar el comando de instalación acorde con la última versión estable.

```
$ curl -L https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname  
$ chmod +x /usr/local/bin/docker-compose
```

### Comprobamos la instalación

```
$ docker-compose --version
```

## Usando Compose

Vamos a ver el uso de Compose en un caso simple, definiendo un ambiente para Wordpress.

**Descargamos el CMS de WordPress y lo descomprimos.**

```
$ curl https://wordpress.org/latest.tar.gz | tar -xvzf -
```

**El directorio resultante es wordpress/, esto lo podemos cambiar a nuestro gusto**

```
$ mv wordpress/ jsitech/
```

**Accedemos al directorio**

```
$ cd jsitech/
```

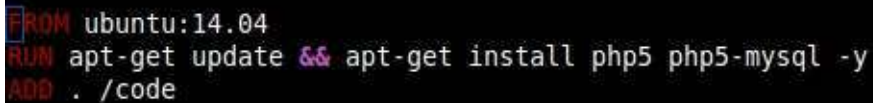
**El Próximo paso es crear un Dockerfile dentro del directorio, que definirá el contenedor que estará ejecutando la aplicación.**

```
$ vi Dockerfile
```

```
FROM ubuntu:14.04

RUN apt-get update && apt-get install php5 php5-mysql -y

ADD . /codigo
```

A screenshot of a terminal window with a black background and red and white text. The text shows the first three lines of the Dockerfile: FROM ubuntu:14.04, RUN apt-get update && apt-get install php5 php5-mysql -y, and ADD . /code.

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install php5 php5-mysql -y
ADD . /code
```

Este **Dockerfile** crea el Contenedor base con los requisitos para ejecutar el cms de WordPress y agrega los archivos correspondiente

El próximo paso es crear un archivo `docker-compose.yml` que iniciará los servicios y una instancia separada de MySQL.

```
$ nano docker-compose.yml
```

```
web:
  build: .
  command: php -S 0.0.0.0:8080 -t /jsitech
  ports:
    - "8080:8080"

  links:
    - db

  volumes:
    - ./jsitech

db:
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: jsitech
    MYSQL_DATABASE: jsitech
    MYSQL_USER: jsitech
    MYSQL_PASSWORD: jsitech
```



```
GNU nano 2.4.2      File: docker-compose.yml
web:
  build: .
  command: php -S 0.0.0.0:8080 -t /jsitech
  ports:
    - "8080:8080"
  links:
    - db
  volumes:
    - ./jsitech
db:
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: jsitech
    MYSQL_DATABASE: jsitech
    MYSQL_USER: jsitech
    MYSQL_PASSWORD: jsitech

[ Read 16 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

## Vamos a explicar brevemente los argumentos

**web** : Nombre que define el servicio

**build** : crea el contenedor, en este caso pasamos . , para que haga uso del Dockerfile que creamos recientemente.

**command** : El comando que le pasamos al contenedor al momento de su ejecución.



**ports:** Mapeo de Puertos

**links :** Definimos el enlace de los contenedores

**volumes :** Creamos el volumen que contiene nuestro código

**db:** Nombre que define nuestro contenedor con la base de datos

**image:** la imagen donde basará su contenedor

**environment:** aquí pasamos las variables, en este caso es un contenedor con MySQL y le pasamos la base de datos a crear y las credenciales.

Ya aquí tenemos el ambiente **multi-contenedor** listo, pero antes de lanzar nuestro proyecto de wordpress debemos configurar `wp-config.php` con las credenciales que le pasamos en las variables

```
$ mv wp-config-sample.php wp-config.php
```

```
$ vi wp-config.php
```

```
// ** MySQL settings - You can get this info from your web host ** //

/** The name of the database for WordPress */
define('DB_NAME', 'jsitech');

/** MySQL database username */
define('DB_USER', 'jsitech');

/** MySQL database password */
define('DB_PASSWORD', 'jsitech');

/** MySQL hostname */
define('DB_HOST', 'db:3306');
```

Ya con todo definido en los archivos correspondiente, solo es lanzar el proyecto multi-contenedor. Ejecutamos el comando

```
# docker-compose up
```

Esto se encargará de crear las imágenes necesarias y lanzar los contenedores correspondientes a la web y la base de datos.

```
root@JsiTech-pc:~/jsitech.com# docker-compose up
Pulling db (mysql:latest)...
latest: Pulling from library/mysql
7268d8f794c4: Pull complete
a3ed95caeb02: Pull complete
e5a99361f38c: Pull complete
50aeb59ed433: Pull complete
5bedb4177480: Pull complete
366f809ce7dd: Pull complete
b847b8b053fd: Pull complete
0bde342dd8a1: Pull complete
fae810009e99: Pull complete
7af5709d8aa1: Pull complete
Digest: sha256:7665507aea0785e89e51c193381ec33ec8662d02cd5c995b9f31e432fcaaa541
Status: Downloaded newer image for mysql:latest
Creating jsitechcom_db_1
Building web
Step 1 : FROM ubuntu:14.04
14.04: Pulling from library/ubuntu
a64038a0eeaa: Pull complete
2ec6e7edf8a8: Pull complete
0a5fb6c3c94b: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:a3799e46440cabaa5414d42972b1693980b30bd2772b969fe11d08d99a8b753c
Status: Downloaded newer image for ubuntu:14.04
--> 14b59d36bae0
Step 2 : RUN apt-get update && apt-get install php5 php5-mysql -y
--> Running in 87007d2deb45
Ign http://archive.ubuntu.com trusty InRelease
Get:1 http://archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:2 http://archive.ubuntu.com trusty-security InRelease [65.9 kB]
Hit http://archive.ubuntu.com trusty Release.gpg
```

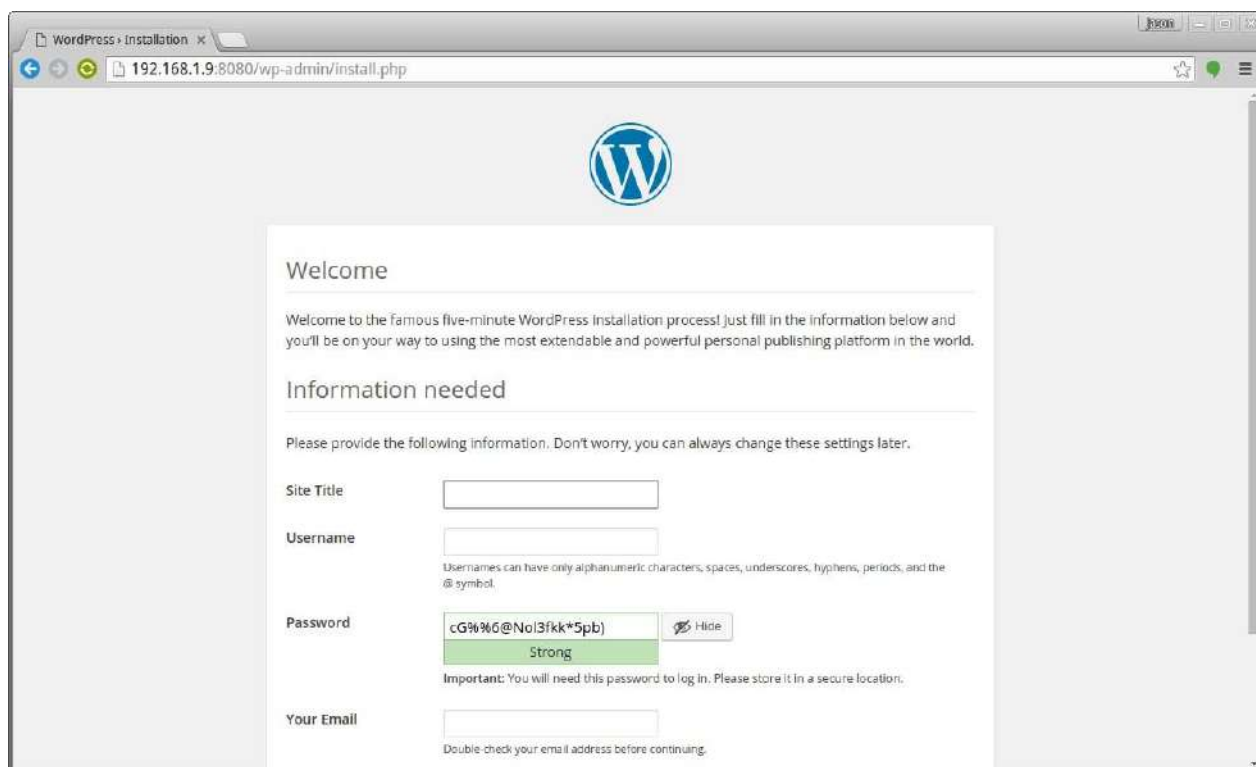
```

Setting up php5 (5.5.9+dfsg-1ubuntu4.14) ...
Processing triggers for libc-bin (2.19-0ubuntu6.7) ...
Processing triggers for sgml-base (1.26+nmu4ubuntu1) ...
---> 656a17ee6dfd
Removing intermediate container 87007d2deb45
Step 3 : ADD . /code
---> 1dd61fe180a6
Removing intermediate container 66f6c66a4789
Successfully built 1dd61fe180a6
Creating jsitechcom_web_1
Attaching to jsitechcom_db_1, jsitechcom_web_1
db_1 | Initializing database
db_1 | 2016-02-23T14:00:13.290828Z 0 [Warning] InnoDB: New log files created, L
SN=45790
db_1 | 2016-02-23T14:00:14.986022Z 0 [Warning] InnoDB: Creating foreign key con
straint system tables.
db_1 | 2016-02-23T14:00:15.363097Z 0 [Warning] No existing UUID has been found,
so we assume that this is the first time that this server has been started. Gen
erating a new UUID: c38da7eb-da35-11e5-882b-0242ac110002.
db_1 | 2016-02-23T14:00:15.420619Z 0 [Warning] Gtid table is not ready to be us
ed. Table 'mysql.gtid_executed' cannot be opened.
db_1 | 2016-02-23T14:00:15.421668Z 1 [Warning] root@localhost is created with a
n empty password ! Please consider switching off the --initialize-insecure optio
n.
db_1 | 2016-02-23T14:00:46.907557Z 1 [Warning] 'user' entry 'root@localhost' ig
nored in --skip-name-resolve mode.
db_1 | 2016-02-23T14:00:46.907621Z 1 [Warning] 'user' entry 'mysql.sys@localhos
t' ignored in --skip-name-resolve mode.
db_1 | 2016-02-23T14:00:46.907662Z 1 [Warning] 'db' entry 'sys mysql.sys@localh
ost' ignored in --skip-name-resolve mode.
db_1 | 2016-02-23T14:00:46.907696Z 1 [Warning] 'proxies_priv' entry '@ root@loc
alhost' ignored in --skip-name-resolve mode.

```

Vamos a ver si todo funciona bien, si recuerdan creamos un mapeo de puertos 8080:8080, lo que quiere decir que debemos poder acceder al contenedor con la IP del host de docker mediante ese puerto.

<http://192.168.1.9:8080>



Excelente todo funciona correctamente.

Si deseamos bajar el ambiente solo debemos correr:

```
$ docker-compose down
```

```
root@Jsitech-pc:~/jsitech.com# docker-compose down
Stopping jsitechcom_web_1 ... done
Stopping jsitechcom_db_1 ... done
Removing jsitechcom_web_1 ... done
Removing jsitechcom_db_1 ... done
```

En este punto ya entendemos como funciona compose y ya podemos ir pensando en otros casos de uso.

# Archivo docker-compose en detalle

El `archivo compose` es un archivo `YAML` donde definimos los servicios, redes y volúmenes. Usualmente lo colocamos en la ruta del directorio donde tendremos todo lo necesario para armar el ambiente.

Estas deficiones contienen toda la configuración que serán aplicadas a cada contenedor iniciado por ese servicio. Sería lo mismo que pasarle estos parámetros con el comando `docker-run`, de la misma manera las definiciones de las redes y volúmenes serían semejantes a los comandos `docker network create` y `docker volume create`.

**Vamos a ver ahora las opciones que podemos utilizar para las deficiones de servicio.**

## build

Aquí definimos las opciones de configuración que serán aplicadas al momento de crear el contenedor. Build puede ser especificado como un valor conteniendo una ruta con un contexto de creación o un objeto con la ruta especificada debajo de `context` y opcionalmente un `dockerfile` o argumento (`args`).

### Ejemplo:

```
build: ./directorio

build:
  context: ./directorio
  dockerfile: <ruta a dockerfile alternativo>

build: . (dockerfile en la misma ruta docker-compose.yml)
```

## context

Aquí definimos una ruta conteniendo un `Dockerfile` o una URL con un repositorio de git. Cuando el valor especificado es una ruta relativa, es interpretada como una ubicación relativa al archivo `docker-compose`. Este directorio será también el contexto de creación enviado al daemon de docker. Compose creará y le colocará un tag a la imagen con un nombre generado.

### Ejemplo:

```
build:
  context: ./directorio
```

## dockerfile

Archivo Dockerfile alternativo. Con esta definición compose utilizará un archivo alternativo para crear la imagen. La ruta debe estar especificada.

### Ejemplo:

```
build:
  context: .
  dockerfile: ruta/a/Dockerfile
```

## args

Con esto definimos **argumentos** de creación. Podemos hacer uso de diversos valores. Si hacemos uso de valores booleanos como, true, false, yes, no, necesitamos encerrarlos en comillas para asegurarnos que no sean convertidos a **TRUE** o **FALSE** por el analizador YML.

Argumentos con un solo valor son resueltos como valores de ambiente en la máquina donde estamos corriendo compose.

### Ejemplo:

```
build:
  args:
    version: 1
    user: jsitech

build:
  args:
    - version=1
    - user=jsitech
```

## image

Especificamos la imagen con el que crearemos el contenedor.

### Ejemplo:

```
image: jsitech/shodan
image: ubuntu:14.04
image: alpine
```

## command

Sustituye el comando por defecto del contenedor

### Ejemplo:

```
command: php -S 0.0.0.0:8080 -t /jsitech
command: [php, -S, 0.0.0.0:8080, -t, /jsitech]
```

## container\_name

Especificamos un nombre personalizado para el contenedor. Ya que los nombres de los contenedores de Docker deben ser únicos, si especificamos un nombre no podremos escalar el servicio en mas de un contenedor.

### Ejemplo:

```
container_name: web_jsitech
```

## depends\_on

Aquí definimos las dependencias entre los servicios.

### Ejemplo:

```
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

## dns / dns\_search

Aquí definimos servidores dns o dominios de búsqueda DNS personalizados

### Ejemplo:

```
dns: 8.8.8.8
dns:
  - 8.8.8.8
  - 8.8.4.4

dns_search: ejemplo.com
dns_search:
  - ejemplo1.com
  - ejemplo2.com
```

## entrypoint

Reemplaza el punto de entrada por defecto.

### Ejemplo:

```
entrypoint: ruby /ruta/app.rb
```

Puede ser también una lista

```
entrypoint:
  - ruby
  - /ruta/app.rb
```

## environment

Aquí agregamos variables de ambientes. Podemos hacer uso de diversos valores y debemos recordar que valores como **true**, **false**, **yes**, **no**, necesitan ser encerrados en comillas para que el analizador no los convierta a TRUE o FALSE.

### Ejemplo:



```
environment:
  MYSQL_DATABASE=jsitech
  MYSQL_USER=jsitech
  MYSQL_PASSWORD=jsitech

environment:
  MYSQL_DATABASE: jsitech
  MYSQL_USER: jsitech
  MYSQL_PASSWORD: jsitech
```

## expose

Expone los puertos sin publicarlos al Host. Solo estarán disponibles para los servicios linkeados. Solo se pueden especificar los puertos internos.

### Ejemplo:

```
expose:
  - "4500"
  - "6000"
```

## links

Linkeamos contenedores con otros servicios. Especificamos el nombre del servicio y el alias, o solo el nombre del servicio. Los contenedores linkeados serán alcanzados con el hostname identificado por el alias o por el nombre del servicio en caso de no haber especificado uno.

### Ejemplo:

```
web:
  links:
    - db
    - dbprueba:mysql
```

## external\_links

Linkeamos contenedores fuera de docker-compose.yml y del mismo compose.

### Ejemplo:

```
external_links:
  - alpine2
  - dbpueba:mysql
```

## logging

Configuración de logging para el servicio. Util cuando queremos centralizar los logs de los contenedores.

### Ejemplo:

```
logging:
  driver: syslog
  options:
    syslog-address: "tcp://10.0.0.45:123"
```

## network\_mode

Este modo usa los mismo valores con el cliente de docker cuando le pasamos la opción `--net`.

```
network_mode: "bridge"
network_mode: "host"
network_mode: "none"
network_mode: "service:[service name]"
network_mode: "container:[container name/id]"
```

## networks

Son las redes a las que se unirán los contenedores, referenciando las entradas debajo de las redes de nivel superior.

### Ejemplo:

```
networks:
  - Red Prueba
  - Red Producción
```

## alias

Los alias son hostnames alternativos que le podemos a los servicios en la red, es decir, que otros contenedores en la misma red pueden llamar al servicio por el nombre de este, o por un alias definidos.

**Ejemplo:**

```
networks:
  red-prueba:
    aliases:
      - alias 1
      - alias 2

  red-producción:
    aliases:
      - alias3
```

En este ejemplo un contenedor puede llamar el servicio u otro contenedor con el nombre del servicio o por alias 1 y alias 2 en la red-prueba o por el alias 3 en la red-producción.

## Ports

Aquí a diferencia de expose, publicamos los puertos al host y podemos especificar el mapeo (**HOST:CONTENEDOR**).

**Ejemplo:**

```
ports:
  - "8080"
  - "8080-8085"
  - "8080:80"
```

## volumes

Montamos rutas o volúmenes, especificamos una ruta en el host (HOST:CONTENEDOR), Igual podemos especificar modo de acceso (HOST:CONTENEDOR:rw)

**Ejemplo:**

```
volumes:
  - /opt/data:/var/lib/mysql
  - ./jsitech:/var/www/html
```

## volumes\_from

Montamos los volúmenes de otro servicio o contenedor

### Ejemplo:

```
volumes_from:  
  - db  
  - web_jsitech
```

Para mantenerlo en lo básico lo dejaré hasta aquí, hay otras versiones que en su momento verán. Para darles una idea opciones para especificar el uso de recursos, definición mas complejas de redes, etc. La idea es que vayan introduciendose con el tema y a partir de ahí van profundizando.

# Cierre

Aquí Finalizamos con esta guía de Docker esperando que haya sido de gran ayuda. Esta Guía puede servir de referencia para los que se están iniciando.

Con todo lo aprendido a lo largo de esta guía es suficiente para que cualquier usuario aprenda los fundamentos de Docker y puede comenzar a sacarle provecho. Tendrá los conocimientos básicos para seguir profundizando con este valioso proyecto.

Esta guía se seguirá actualizando.