

PROYECTO DOCKER: Ezequiel Arielli

Contenido:

Introducción a Docker	PAGE 1
Requisitos	PAGE 1
Características de Docker	PAGE 2
Ventajas y Desventajas	PAGE 2
Componentes	PAGE 3-4
Instalación de Docker	PAGE 5
Comandos de Docker	PAGE 6
Ejemplos de Redes en Docker	PAGE 7
Levantando Imágenes en Docker	PAGE 8
Buildeando Imagen desde Docker File	PAGE 9-10
Utilizando Docker Hub como Repo	PAGE 11-12
Instalando Docker Compose	PAGE 13
TKM en Docker-compose	PAGE 14
Test Aplicación TKM	PAGE 15-16



Introducción a Docker

Docker es un proyecto de código abierto que permite automatizar el despliegue de aplicaciones dentro de “contenedores”, abstrayéndose totalmente del Entorno virtual/Hardware esto significa que nosotros podemos ejecutar contenedores sin importar el tipo de Nube/Cloud o Ambiente local/Hibrido.

Docker corre en cualquier equipo Linux de 64bits que tenga una versión de Kernel superior al 3.8.

Requisitos:

- Kernel Linux Superior a 3.8
- Arquitectura de SO en 64 bits.

Características de Docker:

- Portabilidad: el contenedor Docker podemos desplegarlo en cualquier sistema, sin necesidad de volver a configurarlo o realizar las instalaciones necesarias para que la aplicación funcione, ya que todas las dependencias son empaquetadas con la aplicación en el contenedor.
- Ligereza: los contenedores Docker sólo contienen las modificaciones realizadas sin contar las del sistema operativo base.
- Autosuficiencia: un contenedor Docker no contiene todo un sistema operativo completo, sólo aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga.

VENTAJAS Y DESVENTAJAS

Usar contenedores Docker permite a desarrolladores y administradores de sistemas probar aplicaciones o servicios en un entorno seguro e igual al de producción, reduciendo los tiempos de pruebas y adaptaciones entre los entornos de prueba y producción.

Ventajas:

- Las instancias se inician en pocos segundos.
- Son fácilmente replicables.
- Es fácil de automatizar y de integrar en entornos de integración continua.
- Consumen menos recursos que las máquinas virtuales tradicionales.
- Mayor rendimiento que la virtualización tradicional ya que corre directamente sobre el Kernel de la máquina en la que se aloja, evitando al hypervisor (VMWare, Hyper-V , Xendesktop, KVM, QEMU).
- Ocupan mucho menos espacio que una máquina virtual.
- Permite aislar las dependencias de una aplicación de las instaladas en el host.
- Existe un gran repositorio de imágenes como Docker Hub donde se pueden encontrar muchísimas aplicaciones configuradas para ser ejecutadas.

Desventajas:

- Sólo puede usarse de forma nativa en entornos Unix con Kernel igual o superior a 3.8.
- Sólo soporta arquitecturas de SO 64 bits.
- Como es relativamente nuevo, puede haber errores de código entre versiones.

COMPONENTES:

Según la documentación oficial, Docker tiene dos principales componentes:

- Docker Plataforma open source de virtualización con contenedores.
- Docker Hub Plataforma de Software como servicio (SaaS, Software-as-a-Service) para compartir y administrar contenedores Docker.
- Docker Engine

Es el demonio que se ejecuta dentro del sistema operativo (Linux) y que expone una API para la gestión de imágenes, contenedores, volúmenes o redes. Sus funciones principales son:

- La creación de imágenes Docker.
- Publicación de imágenes en Docker Registry
- Descarga de imágenes desde Docker Registry.
- Ejecución de contenedores usando las imágenes. - Gestión de contenedores en ejecución (pararlo, arrancarlo, ver logs, ver estadísticas).

- Docker Client

Cualquier software o herramienta que hace uso de la API del demonio Docker, pero suele ser el comando docker, que es la herramienta de línea de comandos para gestionar Docker Engine. Éste cliente puede configurarse para hablar con un Docker local o remoto, lo que permite administrar nuestro entorno de desarrollo local como nuestros servidores de producción.

- Docker Images

Son plantillas de sólo lectura que contienen el sistema operativo base (más adelante entraremos en profundidad) dónde correrá nuestra aplicación, además de las dependencias y software adicional instalado, necesario para que la aplicación funcione correctamente. Las plantillas son usadas por Docker Engine para crear los contenedores Docker.

- Docker Registries

Son los registros de Docker donde se guardan las imágenes.

Pueden ser repositorios públicos o privados.

El registro público lo provee el Hub de Docker, que sirve tanto imágenes oficiales cómo las subidas por usuarios con sus propias aplicaciones y configuraciones.

Así tenemos disponibles para todos los usuarios imágenes oficiales de las principales aplicaciones (MySQL, MongoDB, Apache, Tomcat, etc.), así cómo no oficiales de infinidad de aplicaciones y configuraciones. DockerHub ha supuesto una gran manera de distribuir las aplicaciones. Es un proyecto open source que puede ser instalado en cualquier servidor

- Docker Containers

El contenedor de Docker, aloja todo lo necesario para ejecutar un servicio o aplicación. Cada contenedor es creado de una imagen base y es una plataforma aislada. Un contenedor es simplemente un proceso para el sistema operativo, que se aprovecha de él para ejecutar una aplicación. Dicha aplicación sólo tiene visibilidad sobre el sistema de ficheros virtual del contenedor

- Docker Compose

Es otro proyecto open source que permite definir aplicaciones multi-contenedor de una manera sencilla.

Es una alternativa más cómoda al uso del comando docker run, para trabajar con aplicaciones con varios componentes y es una buena herramienta para gestionar entornos de desarrollo y de pruebas o para procesos de integración continua.

- Docker Machine

Es un proyecto open source para automatizar la creación de máquinas virtuales con Docker instalado, en entornos Mac, Windows o Linux, pudiendo administrar así un gran número de máquinas Docker. Incluye drivers para Virtualbox, que es la opción aconsejada para instalaciones de Docker en local, en vez de instalar Docker directamente en el host.

Esto simplifica y facilita la creación o la eliminación de una instalación de Docker, facilita la actualización de la versión de Docker o trabajar con distintas instalaciones a la vez.

Usando el comando docker-machine podemos iniciar, inspeccionar, parar y reiniciar un host administrado, actualizar el Docker client y el Docker daemon, y configurar un cliente para que hable con el host anfitrión.

A través de la consola de administración podemos administrar y correr comandos Docker directamente desde el host. Éste comando docker-machine automáticamente crea hosts, instala Docker Engine en ellos y configura los clientes Docker

Instalación de Docker:

- Linux Ubuntu :

Apt-get install Docker

groupadd docker

usermod -aG docker Ubuntu

service start docker

- Linux Fedora/centos:

Yum install -y docker

Setenforce 0

Systemctl enable docker

Systemctl start docker

Instalando la última versión de Docker-CE/ Esta versión es la recomendada para utilizar Docker Swarm ya que nos agrega el comando de Deploy de Contenedores.

Link de Instalación en Centos/Fedora:

<https://docs.docker.com/engine/installation/linux/centos/>

- Windows / MAC:

En Windows y Mac Docker no corre de forma nativa, esto significa que para instalarlo debemos descargar la herramienta Docker tools la cual nos creara una máquina virtual de Linux en Virtual Box con la cual ejecutara Docker.

Link de Docker Toolbox:

<https://www.docker.com/products/docker-toolbox>

COMANDOS DOCKER:

docker info: Nos muestra la información de nuestro docker engine.

docker version: Muestra la version de docker instalada.

docker images: Muestra las imagenes descargadas en nuestro Registry Local.

docker pull imagen:tag : Descarga una imagen especifica de Docker Hub.

docker pull imagen : Descarga la última versión de la imagen latest.

docker run imagen: ejecuta una imagen y si no la tengo la baja.

docker ps: "Muestra contenedores en ejecución"

docker ps -a : Muestra contenedores ejecutados y finalizados.

docker run ubuntu ps -efa : "Ejecuta el ps dentro del contenedor /ps puede ser cualquier otro comando" ejemplo docker run Ubuntu ping 8.8.8.8

Contenedores interactivos

FLAGS -i Utiliza el estandar input

-t Para que el contenedor pueda recibir comandos.

docker run -it ubuntu bash: "ENTRO EN EL CONTENEDOR COMO BIN/BASH"

CONTROL P + Q "salir del proceso de bash sin finalizar el contenedor"

docker ps -a --no-trunc "ver id unico" referenciarlo

docker rm imagen: Eliminar imagen de docker en ejecución a veces debemos forzar con el flag -f

docker run --name name : Poner nombre a la imagen que se ejecuta.

docker rmi imagen: Elimina la imagen de docker registry.

docker start ID: "re-ejecuto un contenedor detenido "

docker logs ID: Muestra los logs de un contenedor especifico.

docker logs -f : "ID" Logs realtime/Similar a Tail -f

Docker-machine ls :"VER LA IP DOCKER-ENGINE EN WINDOWS"

docker run -P -d tomcat : "-D el contenedor corre como demonio jejej" -P "ejecuta en un puerto aleatorio"

docker stop y docker kill : "detienen contenedor en ejecución"

docker pause ID : "pausa el contenedor el ejecución realtime" "para los dump o aplicaciones defectuosas"

docker unpause ID : vuelve a estado normal hasta terminar

docker inspect : se utiliza para acceder a información útil del contenedor"

docker inspect fadd | grep MacAddress: Muestra la mac-address

docker rm -f ID "detiene el contenedor específico en ejecución"

docker ps -a -q "muestra solo el ID del contenedor detenido"

docker rm \$(docker ps -a -q) : Limpiar "Borra todos los contenedores detenidos | limpia el output"

docker commit ID imagen: "Realiza un commit de una imagen la cual fue modificada"

docker commit "ID" "NOMBRE DE LA NUEVA IMAGEN" - guarda los cambios como nueva imagen

PD: Mientras tenemos un contenedor corriendo y le instalamos cosas pero no lo cerramos

Tiramos Docker diff "ID" de otra ventana nos muestra los cambios q le hicimos.

docker network create: Creamos una red específica la cual puede ser aislada para una aplicación o puede ser una red de tipo bridge.

docker network connect: Conecta a un Contenedor a una red diferente.

docker network ls: Lista las redes creadas en nuestro Docker Engine.

docker network rm: Eliminar red específica

docker network disconnect : Desconectar contenedor de red específica.

docker network inspect

--net especificar una red cuando se crea un contenedor

--link se puede acceder al contenedor por nombre

--net=host "Comparte el mismo Stack de red que mi Servidor docker"

EJEMPLOS DE REDES EN DOCKER:

docker network create Ubuntu-LAN

docker run -it --name ubuntu-LAN --net=LAN ubuntu bash

docker network connect bridge ubuntu-LAN :

Conectar Máquina ubuntu-LAN a la red BRIDGE por DEFAULT | Permite que se vean las 2 redes.

LEVANTANDO IMAGENES EN DOCKER EJEMPLO:

```
docker run --name mariadb-itshell -p 3301:3306 -e MYSQL_DATABASE=itshelldb -e  
MYSQL_ROOT_PASSWORD=1 -e MYSQL_USER=earielli -e MYSQL_PASSWORD=1 -d mariadb
```

```
docker run --name wordpress-itshell --link mariadb-itshell:itshelldb -p 80:80 -d wordpress
```

LEVANTANDO UNA IMAGEN CON ALMACENAMIENTO PERSISTENTE:

```
docker run -d --name mariadb-itshell -v mariadb:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=1 -e  
MYSQL_DATABASE=wordpress -e MYSQL_USER=earielli -e MYSQL_PASSWORD=1 mariadb:latest
```

```
docker run --name itshell-mysql -v /tmp/scripts:/tmp/scripts/ -e MYSQL_ROOT_PASSWORD=1 -d  
mysql:5.6
```

INGRESAR A CONTAINER DE MYSQL

```
docker exec -it mariadb-itshell bash
```

LEVANTANDO WORDPRESS CONECTANDO LA DB PERSONALIZADA

```
docker run --name wordpress-itshell -v /data/wordpress:/var/www/html --link itshell-  
mysql:itsh3080906209 -e WORDPRESS_DB_HOST=172.17.0.2 -e WORDPRESS_DB_USER=root -e  
WORDPRESS_DB_PASSWORD=1 -e WORDPRESS_DB_NAME=itshelldb -p 80:80 -d wordpress
```

Troubleshooting Container

```
docker logs -f <container>
```

Buldeando Imagen desde Docker File:

Componentes:

- FROM: indica la imagen base a partir de la cual crearemos la imagen que construirá el Dockerfile.
- MAINTAINER: documenta el nombre del creador de la imagen.
- ENV HOME: establece el directorio HOME que usarán los comandos RUN.
- RUN: permite ejecutar una instrucción en el contenedor, por ejemplo, para instalar algún paquete mediante el gestor de paquetes (apt-get, yum, rpm, ...).
- ADD: permite añadir un archivo al contenedor, en muchas ocasiones se utiliza para proporcionar la configuración de los servicios (ssh, mysql, ...).
- VOLUME: establece puntos de montaje que al usar el contenedor se pueden proporcionar determinado acceso a un directorio y proporcionar persistencia (las imágenes de docker son de solo lectura y no almacenan datos entre diferentes ejecuciones).
- EXPOSE: indica los puertos TCP/IP por los que se pueden acceder a los servicios del contenedor, los típicos son 22 (SSH), 80 (HTTP) y en este caso el puerto por defecto de mysql 3306.
- CMD: establece el comando del proceso de inicio que se usará si no se indica uno al iniciar un contenedor con la imagen.

Ejemplo de Imagen en Dockerfile – dev-mundotkm.

```
zz@localhost:TKM/Projects/docker_tkm/dockerfiles/dev-mundotkm
File Edit View Search Terminal Help
FROM centos:latest
MAINTAINER Infraestructura VIDA
LABEL Vendor="CentOS"

# Packages install
RUN yum -y update && yum clean all
RUN yum -y install epel-release && yum clean all
RUN yum -y install memcached git php-pecl-memcache php-mbstring unzip httpd php php-mysql php-gd mod_ssl php-pecl-apcu && yum clean all
RUN yum -y install libwebp.x86_64 libwebp-tools.x86_64 pngquant optipng libjpeg-turbo.x86_64 libjpeg-turbo-utils.x86_64 && yum clean all

# Enviroment creation
RUN useradd mundotkm -G apache
RUN mkdir -p /var/www/sites.enable/env
RUN chown mundotkm.mundotkm /var/www/sites.enable/ -R

# Cert
ADD ./ssl.crt /etc/httpd/conf/ssl.crt
ADD ./ssl.key /etc/httpd/conf/ssl.key

# Entrypoint
ADD ./entrypoint.sh /entrypoint.sh
RUN chmod 777 /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]

EXPOSE 80

CMD ["apachectl"]

-- INSERT --
```

Buildeando imagen desde docker file con nombre especifico, esto ejecutara las instrucciones del Dockerfile y nos creara la imagen “imagen-devmundo-test” en nuestro docker registry.

```
zz@localhost:/TKM/Projects/docker_tkm/dockerfiles/dev-mundotkm
File Edit View Search Terminal Help
-
[root@localhost dev-mundotkm]# docker build -t imagen-devmundo-test .
Sending build context to Docker daemon 8.192 kB
Step 1/17 : FROM centos:latest
--> 98d35105a391
Step 2/17 : MAINTAINER Infraestructura VIDA
--> Using cache
--> e8637710b15b
Step 3/17 : LABEL Vendor "CentOS"
--> Using cache
--> 685da93c212e
Step 4/17 : RUN yum -y update && yum clean all
--> Using cache
--> dbeffbae3dce
Step 5/17 : RUN yum -y install epel-release && yum clean all
--> Using cache
--> 7008620431ad
Step 6/17 : RUN yum -y install memcached git php-pecl-memcache php-mbstring unzip httpd php php-mysql php-gd mod_ssl php-pecl-apcu && yum clean all
--> Using cache
--> acb23570ce53
Step 7/17 : RUN yum -y install libwebp.x86_64 libwebp-tools.x86_64 pngquant optipng libjpeg-turbo.x86_64 libjpeg-turbo-utils.x86_64 && yum clean all
--> Using cache
--> 10e94fea2966
Step 8/17 : RUN useradd mundotkm -G apache
--> Using cache
--> 67c8cd101bf3
Step 9/17 : RUN mkdir -p /var/www/sites.enable/env
--> Using cache
--> 38ad44673e4d
Step 10/17 : RUN chown mundotkm.mundotkm /var/www/sites.enable/ -R
--> Using cache
```

Imagen generada en nuestro docker registry.

[root@localhost dev-mundotkm]# docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
imagen-devmundo-test	latest	811d51d882c3	3 minutes ago	392 MB

Utilizando Docker Hub como Repo:

Docker hub es el repositorio de imágenes de docker donde podemos subir nuestras imágenes versionadas y podemos elegir que sean privadas o de tipo publicas .

Para poder empezar a usar este servicio debemos crear una cuenta en <https://hub.docker.com/>

Una vez que tenemos nuestra cuenta creada debemos crear un repo (La única limitación que tiene la cuenta gratuita es que solamente podemos crear un solo repo si queremos más deberíamos tener una cuenta Enterprise).

The image shows two screenshots of the Docker Hub website. The top screenshot is the 'Welcome to Docker Hub' page, which includes a search bar, navigation links (Dashboard, Explore, Organizations, Create), and three main action buttons: 'Create Repository', 'Create Organization', and 'Explore Repositories'. The bottom screenshot is the 'Create Repository' form, which is divided into two columns. The left column contains a list of instructions: 1. Choose a namespace (Required), 2. Add a repository name (Required), 3. Add a short description, 4. Add markdown to the full description field, and 5. Set it to be a private or public repository. The right column contains the form fields: a dropdown menu for the namespace (set to 'mundotkm'), a text input for the repository name (set to 'dev'), a text area for the short description (set to 'Repositorio mundo TKM - test'), and a dropdown menu for visibility (set to 'private').

ntentando conectar a sp.adbri.com...

1. Choose a namespace (Required)
2. Add a repository name (Required)
3. Add a short description
4. Add markdown to the full description field
5. Set it to be a private or public repository

mundotkm dev

Short Description (100 Characters)

Repositorio mundo TKM - test

Visibility

private

Lo primero que necesitamos hacer para poder subir una imagen que la misma tenga el nombre de la cuenta y el repo que creamos, por ejemplo en este caso sería mundotkm/dev:1.0

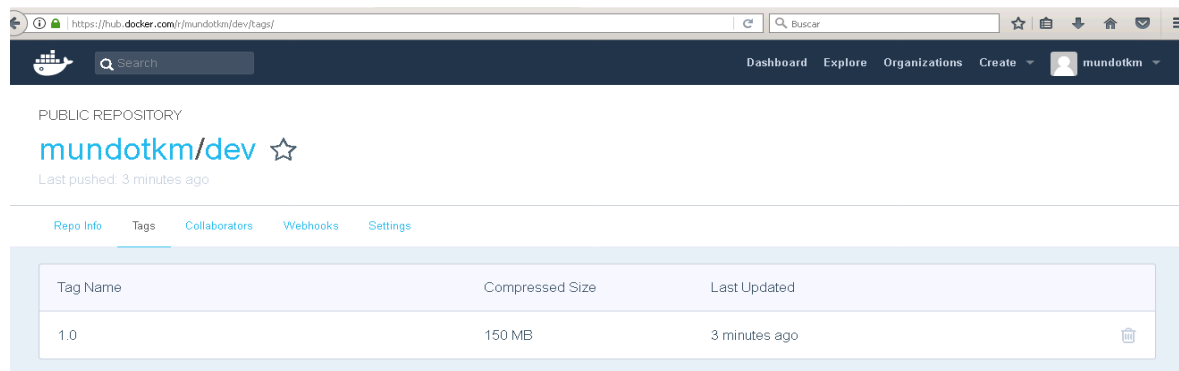
NOTA: "1.0 es el número de Version"

Ejemplo de Pull y push image docker hub:

El primer paso es logear nuestra cuenta de docker hub en el docker-engine para eso ejecutamos el comando "docker login" ingresamos nuestro usuario y ya estamos preparados para usar nuestro repo.

A continuación se puede ver como subir una imagen a docker hub , yo en este ejemplo ya tengo una imagen buildada con la sintaxis "Cuenta-RepoName-Version" la cual nos quedaría mundotkm/dev:1.0

```
---> 401fabfcca75
Step 16/17 : EXPOSE 80
---> Using cache
---> 67558a436789
Step 17/17 : CMD apachectl
---> Using cache
---> 811d51d882c3
Successfully built 811d51d882c3
[root@localhost dev-mundotkm]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
imagen-devmundo-test latest             811d51d882c3       22 minutes ago     392 MB
mundotkm/dev         1.0                811d51d882c3       22 minutes ago     392 MB
nightmarezel/zz1     101                eeff376f9f351      6 days ago         392 MB
nightmarezel/zz1     100.0              5b20cc0c2e7f       8 days ago         392 MB
dockerfiles_tkm      latest             852b7a7ac046       13 days ago        392 MB
<none>               <none>             9b42a8567c06       13 days ago        392 MB
dockerfiles_mongodb01 latest             c7852795b35b       13 days ago        286 MB
dockerfiles_memcache01 latest             239a8264e018       13 days ago        289 MB
centos               latest             98d35105a391       2 weeks ago        193 MB
nightmarezel/zz1     10.0              6b3f2a9ed9da       3 weeks ago        449 MB
[root@localhost dev-mundotkm]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username (nightmarezel): mundotkm
Password:
Login Succeeded
[root@localhost dev-mundotkm]# docker push mundotkm/dev:1.0
The push refers to a repository [docker.io/mundotkm/dev]
8b7d7578f92b: Pushed
4616926d7910: Pushed
5a40f1509988: Pushed
747b61f0642c: Pushed
bed2e8a77ffc: Pushed
9e7ba938274: Pushed
5a57abb4e83b: Pushed
ad9e74a620f4: Pushing [=====>] 9.209 MB/22.12 MB
7ba0e6383d12: Pushing [==>] 3.771 MB/138.1 MB
2346b0cf06bb: Pushing [>] 222.7 Kb/19.58 MB
b599bd310da7: Waiting
9b198ff9ff5b: Waiting
[12-
```



Para descargar esta imagen lo único que debemos hacer es ejecutar docker pull mundotkm/dev:1.0 .

Si queremos compartir la imagen a otro usuario podemos compartir la misma de la página de hub.docker.com / Recordar siempre logear nuestra cuenta de docker para poder acceder al repo.

Instalando Docker Compose

- Es una herramienta para levantar aplicaciones multicontenedor
- Se crea a partir de en un fichero yaml donde se declaran los contenedores a levantar.
- Facilita el trabajo de un desarrollador a la hora de levantar una aplicación multi-contenedor.

En el siguiente link podemos apreciar los diferentes tipos de instalación soportados (Linux, Windows, OSX).

<https://docs.docker.com/compose/install/>

Instalando Compose en Centos 7 / Fedora

```
[root@minion2 ~]# curl -L "https://github.com/docker/compose/releases/download/1.11.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload Upload   Total   Spent    Left   Speed
100 600    0 600    0    0  497    0 --:--:-- 0:00:01 --:--:-- 497
100 8066k 100 8066k    0    0  924k    0 0:00:08 0:00:08 --:--:-- 1703k
[root@minion2 ~]# chmod +x /usr/local/bin/docker-compose
[root@minion2 ~]# mv /usr/local/bin/docker-compose /usr/bin/docker-compose
[root@minion2 ~]# docker-compose
bin/ boot/ dev/ etc/ home/ lib/ lib64/ media/ mnt/ opt/ proc/ root/ run//sbin/ srv/ sys/ tmp/ usr/ var/
[root@minion2 ~]# docker-compose version
docker-compose version 1.11.2, build dfed245
docker-py version: 2.1.0
CPython version: 2.7.13
OpenSSL version: OpenSSL 1.0.1t 3 May 2016
```

Comandos:

docker-compose up : Levanta el archivo docker-compose.yml de nuestro directorio y muestra el output/logs de los contenedores corriendo

docker-compose up -d : Levanta el archivo docker-compose.yml como demonio sin mostrar el output.

docker-compose -f docker-compose-mundo.yml up : Levanta un archivo yml específico.

docker-compose ps: Muestra el estado de los compose levantados.

docker-compose stop: Detener Stack de Compose en ejecución.

docker-compose start: Volver arrancar Stack detenido.

docker-compose logs container: Visualizar Logs de contenedor específico del Stack.

docker-compose logs : Muestra los generales del Stack corriendo.

docker-compose kill : Eliminar contenedores del Stack .

Diferencia entre Outpout y Demonio :

```
[root@localhost dockerfiles]# docker-compose up
Recreating memcache01
Recreating mongod01
Starting mariadb01
Recreating mundotkm1
Attaching to mariadb01, mongod01, mundotkm1
mariadb01 | 170405 17:39:41 mysqld_safe WARNING: Found /var/lib/mysql/my.cnf
mariadb01 | The data directory is a deprecated location for my.cnf, please move it to
mariadb01 | /usr/my.cnf
mongod01 | /usr/bin/mongod --help for help and startup options
mariadb01 | 170405 17:39:41 mysqld_safe Logging to '/var/lib/mysql/mariadb01.err'.
mariadb01 | 170405 17:39:41 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
mongod01 | 2017-04-05T17:39:46.579+0000 [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=mongod01
mundotkm1 | AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.5. Set the 'ServerName' directive globally to suppress this mess
age
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] db version v2.6.12
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] git version: nogitversion
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] build info: Linux buildvm-16.phx2.fedoraproject.org 4.8.12-300.fc25.x86_64 #1 SMP Fri Dec 2 17:52:11 UTC 2016 x86_64 B005
T_LIB_VERSION=1.53
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] allocator: tcmalloc
mongod01 | 2017-04-05T17:39:46.581+0000 [initandlisten] options: {}
mongod01 | 2017-04-05T17:39:46.713+0000 [initandlisten] journal dir=/data/db/journal
mongod01 | 2017-04-05T17:39:46.713+0000 [initandlisten] recover : no journal files present, no recovery needed
mongod01 | 2017-04-05T17:39:50.485+0000 [initandlisten] preallocateIsFaster=true 47.44
mongod01 | 2017-04-05T17:39:53.297+0000 [initandlisten] preallocateIsFaster=true 34.42
mongod01 | 2017-04-05T17:39:57.197+0000 [initandlisten] preallocateIsFaster=true 35.4
mongod01 | 2017-04-05T17:39:57.197+0000 [initandlisten] preallocateIsFaster check took 10.483 secs
mongod01 | 2017-04-05T17:39:57.197+0000 [initandlisten] preallocating a journal file /data/db/journal/prealloc.0
mongod01 | 2017-04-05T17:40:00.006+0000 [initandlisten] File Preallocator Progress: 744488960/1073741824 69%
^CGracefully stopping... (press Ctrl+C again to force)
Stopping mundotkm1 ... done
Stopping memcache01 ... done
Stopping mongod01 ... done
Stopping mariadb01 ... done
[root@localhost dockerfiles]# docker-compose up -d
Starting memcache01
Starting mongod01
Starting mariadb01
Starting mundotkm1
[root@localhost dockerfiles]# docker compose ps
docker: 'compose' is not a docker command.
See 'docker --help'
[root@localhost dockerfiles]# docker-compose ps

```

Name	Command	State	Ports
mariadb01	/entrypoint.sh mysqld_safe	Up	0.0.0.0:3306->3306/tcp
memcache01	memcached -u daemon	Up	0.0.0.0:11211->11211/tcp
mongod01	/usr/bin/mongod	Up	0.0.0.0:27017->27017/tcp
mundotkm1	/entrypoint.sh apachectl	Up	0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp

```
[root@localhost dockerfiles]#
```

Ejemplo de docker-compose.yml | App TKM.

```
tkm:
  build: ./dev-mundotkm
  links:
    - memcache01
    - mariadb01:mundotkm_global
    - mongod01
  volumes:
    - /TKM/Projects/tkm_redisenio/html:/var/www/html
    - /TKM/Projects/tkm_redisenio/env:/var/www/sites.enable/env
    - /TKM/Projects/tkm_redisenio/env:/var/www/env
    - /TKM/Projects/tkm_redisenio/conf/tkm.conf:/etc/httpd/conf.d/vhost-mundotkm.conf
    - /TKM/Projects/tkm_redisenio/conf/apipolls.conf:/etc/httpd/conf.d/vhost-apipolls.conf
    - /TKM/Projects/tkm_redisenio/conf/reacciones.conf:/etc/httpd/conf.d/vhost-reacciones.conf
  container_name: mundotkm1
  hostname: tkm
  ports:
    - 80:80
    - 443:443

mariadb01:
  image: nightmarezel/zs1:10.0
  volumes:
    - /TKM/Projects/tkm_redisenio/mysql:/var/lib/mysql
    - /TKM/Projects/tkm_redisenio/mysql/my.cnf:/etc/my.cnf
  container_name: mariadb01
  hostname: mariadb01
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=mundotkm_global
    - MYSQL_USER=root
    - MYSQL_PASSWORD=root

  ports:
    - 3306:3306

memcache01:
  build: ./memcache
  container_name: memcache01
  hostname: memcache01
  ports:
    - 11211:11211

mongod01:
  build: ./mongodb
  hostname: mongod01
  container_name: mongod01
  ports:
    - 27017:27017
```

Levantando Aplicación TKM:

Clonamos el repositorio y movemos la carpeta Projects a TKM del /

Ingresamos a /TKM/Projects/docker_tkm/dockerfiles

Ejecutamos docker-compose up -d

Comprobamos que se ejecutaron los contenedores correctamente y añadimos tkm.vm a nuestro /etc/hosts con nuestra ip.

```
zz@localhost: /TKM/Projects/docker_tkm/dockerfiles
File Edit View Search Terminal Help
[root@localhost ~]# git clone http://gitlab.grupovi-da.biz/earielli/TKM.git
Cloning into 'TKM'...
Username for 'http://gitlab.grupovi-da.biz': earielli@grupovi-da.com
Password for 'http://earielli@grupovi-da.com@gitlab.grupovi-da.biz':
remote: Counting objects: 9356, done.
remote: Compressing objects: 100% (7010/7010), done.
remote: Total 9356 (delta 2030), reused 9322 (delta 2020)
Receiving objects: 100% (9356/9356), 55.60 MiB | 10.38 MiB/s, done.
Resolving deltas: 100% (2030/2030), done.
Checking connectivity... done.
[root@localhost ~]# ls
l boot dev home lib64 lost+found media opt root sbin sys TKM.tar usr
bin data etc lib local mariadb_data mnt proc run srv TKM tmp var
[root@localhost TKM]# ls
Rea.md TKM
[root@localhost TKM]# cd ..
[root@localhost ~]# ls
l boot dev home lib64 lost+found media opt root sbin sys TKM.tar usr
bin data etc lib local mariadb_data mnt proc run srv TKM tmp var
[root@localhost TKM]# cd TKM/
[root@localhost TKM]# ls
Projects
[root@localhost TKM]# mv Projects/ /TKM/
[root@localhost TKM]# ls
Projects
[root@localhost TKM]# cd ..
[root@localhost TKM]# ls
Projects Rea.md TKM
[root@localhost TKM]# rm TKM/
rm: cannot remove 'TKM/': Is a directory
[root@localhost TKM]# rm -R TKM/
rm: remove directory 'TKM/'? y
[root@localhost TKM]# ls
Projects Rea.md
[root@localhost TKM]# cd Projects/
[root@localhost Projects]# cd docker_tkm/
[root@localhost docker_tkm]# cd dockerfiles/
[root@localhost dockerfiles]# docker-compose up
Creating mongodb01
Creating mariadb01
Creating memcache01
Creating mundotkm1
Attaching to memcache01, mongodb01, mariadb01, mundotkm1
mongodb01 | /usr/bin/mongod -help for help and startup options
mongodb01 | 2017-04-05T18:16:00.532+0000 [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=mongodb01
mariadb01 | 170405 18:16:00 mysqld_safe WARNING: Found /var/lib/mysql/my.cnf
mariadb01 | The data directory is a deprecated location for my.cnf, please move it to
mongodb01 | 2017-04-05T18:16:00.533+0000 [initandlisten] db version v2.6.12
mariadb01 | /usr/my.cnf
mongodb01 | 2017-04-05T18:16:00.533+0000 [initandlisten] git version: nogitversion
mongodb01 | 2017-04-05T18:16:00.533+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
mongodb01 | 2017-04-05T18:16:00.533+0000 [initandlisten] build info: Linux buildvm-16.phx2.fedoraproject.org 4.8.12-300.fc25.x86_64
# SMP Fri Dec 2 17:52:11 UTC 2016 x86_64 BOOST_LIB_VERSION=1.53
```

Validamos nuestra IP

```
root@localhost zz]# ip a
: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid lft forever preferred_lft forever
: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 50:b7:c3:06:36:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.7.74/22 brd 192.168.7.255 scope global dynamic enp2s0
        valid lft 2887sec preferred_lft 2887sec
    inet6 fe80::2235:c2fb:3657:5174/64 scope link
        valid lft forever preferred_lft forever
```

Agregamos en /etc/hosts nuestra ip con el nombre tkm.vm

```
File Edit View Search Terminal Help
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.66.250 contenedor.leamos.com
192.168.7.74 tkm.vm
```


Comprobamos que nuestro Stack de Compose levanto correctamente.

```
[root@localhost dockerfiles]# ls
dev-mundotkm  docker-compose.yml  elasticsearch  mariadb  memcache  mongodb  proceso.sh  ssh
[root@localhost dockerfiles]# docker-compose up -d
Starting memcache01
Starting mariadb01
Starting mongod01
Starting mundotkm1
[root@localhost dockerfiles]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
aac6760934f1	dockerfiles_tkm	"/entrypoint.sh ap..."	16 minutes ago	Up 7 seconds	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp	mundotkm1
9c9687c49668	nightmarezel/zzi:10.0	"/entrypoint.sh my..."	16 minutes ago	Up 9 seconds	0.0.0.0:3306->3306/tcp	mariadb01
953c80818ba0	dockerfiles_memcache01	"memcached -u daemon"	16 minutes ago	Up 10 seconds	0.0.0.0:11211->11211/tcp	memcache01
7d3668b3c7	dockerfiles_mongodb01	"/usr/bin/mongo"	16 minutes ago	Up 9 seconds	0.0.0.0:27017->27017/tcp	

```
[root@localhost dockerfiles]#
```

Luego de comprobar ejecutamos el script Proceso.sh el cual nos realizara un minify-all del tema de TKM.

```

zz@localhost:TKM/Projects/docker_tkm/dockerfiles
File Edit View Search Terminal Help

Complete!
npm WARN deprecated minimatch@2.0.10: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated minimatch@0.2.14: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated graceful-fs@1.2.3: graceful-fs v3.0.0 and before will fail on node releases >= v7.0. Please update to graceful-fs@~4.0.0 as soon as possible. Use
npm ls graceful-fs' to find it in the tree.
tkm_rediseno@1.0.0 /TKM/Projects/tkm_rediseno
├── gulp@3.9.1
│   ├── archy@1.0.0
│   ├── chalk@1.1.3
│   ├── ansi-styles@2.2.1
│   ├── escape-string-regexp@1.0.5
│   ├── has-ansi@2.0.0
│   │   ├── ansi-regex@2.1.1
│   │   └── strip-ansi@3.0.1
│   ├── supports-color@2.0.0
│   ├── deprecated@0.0.1
│   ├── gulp-util@3.0.8
│   ├── array-differ@1.0.0
│   ├── array-uniq@1.0.3
│   ├── beeper@1.1.1
│   ├── dateFormat@2.0.0
│   ├── fancy-log@1.3.0
│   ├── time-stamp@1.0.1
│   ├── gulplog@1.0.0
│   │   └── glogg@1.0.0
│   ├── has-gulplog@1.0
│   ├── sparkles@1.0.0
│   ├── lodash._reescape@3.0.0
│   ├── lodash._reevaluate@3.0.0
│   ├── lodash._reinterpolate@3.0.0
│   ├── lodash.template@3.6.2
│   ├── lodash._basecopy@3.0.1
│   ├── lodash._basetostring@3.0.1
│   ├── lodash._basevalues@3.0.0
│   ├── lodash._isiterateecall@3.0.9
│   ├── lodash._escape@3.2.0
│   ├── lodash._root@3.0.1
│   ├── lodash._keys@3.1.2
│   ├── lodash._getnative@3.9.1
│   ├── lodash._isarguments@3.1.0
│   └── lodash._isarray@3.0.0

```

Finalizado el gulp Ejecutamos en nuestro navegador. <http://tkm.vm>

