**Aplicaciones multicontenedor con Docker Compose**

**Continuando la serie de artículos sobre Docker, otra de las herramientas disponibles en el ecosistema y que nos facilitará enormemente el administrar aplicaciones compuestas por varios contenedores relacionados entre sí es Docker Compose. Definiendo en un archivo los contenedores que forman parte de una aplicación podemos iniciarlos, pararlos, eliminarlos o ver su estado como si de una unidad se tratara.**

Una vez que ya sabemos que nos puede proporcionar una de las herramientas de la que más se está hablando con la [introducción a docker](https://picodotdev.github.io/blog-bitix/2014/10/introduccion-y-caracteristicas-de-docker/), conocemos [como empezar a usar docker](https://picodotdev.github.io/blog-bitix/2014/11/inicio-basico-de-docker/) y sabemos [cómo crear nuestras propias imágenes de docker](https://picodotdev.github.io/blog-bitix/2014/11/como-crear-una-imagen-para-docker-usando-un-dockerfile/) para los propósitos específicos que necesitemos nos resultará interesante algunas de las nuevas herramientas que proporciona docker. El ecosistema de docker ya no solo se compone del entorno de ejecución de los contenedores, explicadas de forma básica son:

* [Docker](https://www.docker.com/): el sistema de contenedores (*runtime*, imágenes, …).
* [Docker Hub](https://registry.hub.docker.com/): el repositorio de imágenes que las diferentes herramientas de forma oficial o a través de usuarios construyen y comparten imágenes que cualquiera puede usar.
* [Docker Compose](https://docs.docker.com/compose/): permite describir un conjunto de contenedores que se relacionan entre ellos.
* [Docker Machine](https://docs.docker.com/machine/): permite construir máquinas virtuales para alojar contenedores. Similar a lo que hace [Vagrant](https://www.vagrantup.com/), al igual que el anterior también permite construir máquinas que ejecuten contenedores con [VirtualBox](https://www.virtualbox.org/) o diferentes sistemas de computación en en la nube como [Amazon EC2](http://aws.amazon.com/es/ec2/), [Digital Ocean](https://www.digitalocean.com/) u otros. Docker Machine puede ser la forma de usar docker en Windows o Mac OS hasta que funcione de forma nativa como en GNU/Linux.
* [Docker Swarm](https://docs.docker.com/swarm/): permite construir un *cluster* de máquinas al que se puede enviar contenedores docker para su ejecución.

Continuando la [serie de artículos sobre docker](https://picodotdev.github.io/blog-bitix/series/docker/) que escribí hace unos meses escribiré sobre cada una de estas nuevas herramientas, empezando en este artículo con Docker Compose en el que explicaré de qué forma nos puede servir, como instalarlo y el descriptor de contenedores con un ejemplo.

**Introducción a Docker Compose**

Las aplicaciones basadas en microservicios se prestan a usar múltiples contenedores cada uno con un servicio, uno puede contener la base de datos [postgresql](https://www.postgresql.org/), otro una base de datos clave/valor [redis](https://redis.io/) o de documentos como [elasticsearch](https://www.elastic.co/products/elasticsearch/) para hacer búsquedas, otro un sistema de mensajería como [rabbitmq](https://www.rabbitmq.com/), otro [tomcat](https://tomcat.apache.org/) o [wildfly](https://wildfly.org/) que use los anteriores y un servidor web como [Nginx](https://nginx.org/). Teniendo múltiples contenedores usar el comando docker run para cada uno de ellos nos resultará incómodo. En este punto entra Docker Compose permitiéndonos definir nuestra aplicación multicontenedor en un archivo con las mismas propiedades que indicaríamos con el comando docker run individualmente. Con un único comando podremos iniciar todos los contenedores y en el orden que los especifiquemos.

El archivo descriptor nos puede servir no solo como forma de iniciar los contenedores en un entorno de desarrollo sino como de documentación de la aplicación en la que veremos qué contenedores, imágenes, volúmenes, enlaces y demás propiedades tienen.

**Instalar Docker Compose**

Tenemos varias formas de instalar Docker Compose. La que más me gusta y la que recomiendo por ser sencilla es [descargar el binario de docker compose](https://github.com/docker/compose/releases) según nuestra plataforma GNU/Linux o Mac. Descargando el binario de Docker Compose deberemos darle permisos de ejecución y si nos interesa colocarlo en la variable de entorno *PATH* del sistema:

|  |  |
| --- | --- |
| 1 | $ chmod +x docker-compose |

[**view rawpermisos.sh**](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/code/permisos.sh)

|  |  |
| --- | --- |
| 1 | PATH=$PATH:~/Software/scripts/docker-compose |

[**view rawbashrc**](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/code/bashrc)

Con el siguiente comando veremos que Docker Compose funciona correctamente y la versión del mismo.

|  |  |
| --- | --- |
| 1  2  3  4 | $ docker-compose --version  docker-compose version: 1.3.1  CPython version: 2.7.9  OpenSSL version: OpenSSL 1.0.1e 11 Feb 2013 |

[**view rawdocker-compose-version.sh**](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/code/docker-compose-version.sh)

**El descriptor de contenedores**

El descriptor de los contenedores a usar con Docker Compose es un archivo de texto con [formato yaml](https://www.yaml.org/) en la que especificamos los diferentes contenedores y sus propiedades, básicamente podemos indicar las mismas propiedades que indicamos arrancando los contenedores individualmente con el comando docker run. En el siguiente ejemplo vemos varios contenedores, dos contenedores de datos para redis y postgresql, los contenedores de redis y postgresql y un contenedor para la aplicación usando tomcat enlazado con los contenedores de redis y postgresql definidos previamente.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | redisdb:  image: busybox  volumes:  - /var/lib/redis  postgresqldb:  image: busybox  volumes:  - /var/lib/postgresql/data  redis:  image: redis:3  volumes\_from:  - redisdb  ports:  - "6379:6379"  postgresql:  image: postgres:9  volumes\_from:  - postgresqldb  environment:  POSTGRES\_USER: posgresql  POSTGRES\_PASSWORD: posgresql  ports:  - "5432:5432"  apps:  image: library/tomcat:8-jre8  links:  - redis  - postgresql  ports:  - "8080:8080" |

[**view rawdocker-compose.yml**](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/code/docker-compose.yml)

La imagen de los contenedores se indica con la propiedad *image*, los contenedores de datos, *redisdb* y *posgresqldb*, usan la propiedad *volumes* con los datos que guardarán y la imagen de *busybox* (se suele usar esta para los contenedores de datos porque es muy pequeña), con la propiedad *hostname* podemos indicar el nombre de la máquina que al usar la propiedad *link* docker hará visible al contenedor que los usen, con *volumes\_from* podemos usar volúmenes, con *links* enlazar contenedores y con *ports* asociar puertos entre los contenedores y la propia máquina anfitrión, en el ejemplo he usado los puertos por defecto de cada uno de los servicios.

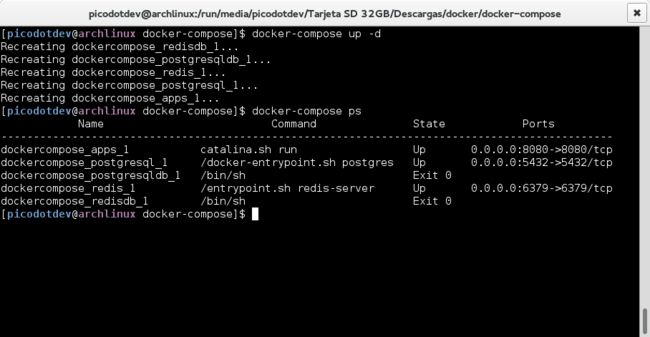
La [descripción completa del formato del archivo de Docker Compose](https://docs.docker.com/compose/yml/) nos da una idea de las opciones que podemos usar, está bastante bien explicado y con ejemplos que nos resultará sencillo entender conociendo los parámetros que usamos con *docker run*.

**Iniciar los contenedores con Docker Compose**

Escrito el archivo de los contenedores y llamándolo *docker-compose.yml* podemos iniciar los contenedores con el comando docker-compose up estando en el mismo directorio de trabajo donde esté ubicado del archivo yml (y previamente habiendo iniciado el servicio de docker). Con *docker-compose ps* podremos ver el estado de los contenedores y de cuales está compuesta la aplicación. Con la opción *–help* podemos ver la lista completa de comandos que podemos usar.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ docker-compose up -d  $ docker-compose ps  $ docker-compose stop  $ docker-compose restart  $ docker-compose logs |

[**view rawdocker-compose-up.sh**](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/code/docker-compose-up.sh)

[](https://picodotdev.github.io/blog-bitix/2015/07/aplicaciones-multicontenedor-con-docker-compose/images/docker-compose.png)Docker Compose

docker-compose inicia los contenedores en el orden que hemos indicado en el archivo de definición, las trazas emitidas de los servicios de los contenedores aparecerán en la terminal si iniciamos los contenedores en primer plano y con *Ctrl+C* se pararán los contenedores. Indicando la opción *-d* los contenedores se iniciarán en segundo plano, con *docker-compose stop* podremos pararlos, con *docker-compose restart* reiniciarlos, *docker-compose rm* para eliminar completamente los contenedores y con *docker-compose logs* veremos las trazas emitidas por los servicios que nos serán de utilizar si iniciamos los contenedores en segundo plano.