**Docker compose es otro proyecto *open source* que permite definir aplicaciones muilti-contenedor de una manera sencilla y declarativa. Es una herramienta ideal para gestionar entornos de desarrollo y de pruebas, o para procesos de integración continua como veremos en la próxima sesión.**

*docker-compose* es una alternativa más cómoda al uso del comando *docker run*, que resulta ingobernable cuando trabajamos con aplicaciones con varios componentes. Con Docker Compose se define un fichero *docker-compose.yml* que tiene esta forma:

```
web:
  build: .
  ports:
   - "5000:5000"
  links:
   - redis
redis:
  image: redis
```

Donde estamos definiendo una aplicación que se compone de un contenedor definido desde un Dockerfile local, que escucha en el puerto 5000, y que hace uso de *redis* como un servicio externo. Dada esta definición, la manera de levantar la aplicación es simplemente:

```
docker-compose up -d
```

*docker-compose* acepta distintos comando, una lista completa puede encontrarse aquí. Destacar los siguientes puntos sobre *docker-compose*:

- `docker-compose up -d` levanta la aplicación en modo demonio, `docker-compose up` la levanta en primer plano, mostrando los logs de los distintos contenedores. La ejecución sucesiva del comando `docker-compose up -d` sólo recrea los contenedores que hayan cambiado su imagen o su definición.

- `docker-compose up -d` no hace el build de las imágenes locales. Si deseas actualizar tu aplicación en base a los últimos cambios de tu código, necesitarás hacer `docker-compose build` antes de ejecutar `docker-compose up -d` nuevamente. Un truco para mejorar este proceso es montar tu código como un volumen en el fichero `docker-compose.yml`, de tal manera que tu container siempre ve los últimos cambios en tu código fuente.

*docker-compose* permite definir prácticamente todos los flags que soporta el comando *docker run*, pero *docker-compose* es mucho más fácil de utilizar. Las opciones más comunes son:

- *build*: para indicar que el container se construye desde un Dockerfile local.
- *image*: para indicar que el container corre un imagen remota.
- *command*: para redefinir el comando que ejecuta el container en lugar del comando definido en la imagen.
- *environment*: para definir variables de entorno en el contenedor. Se pueden pasar haciendo referencia a un fichero usando la propiedad *env_file*. Si la variable no tiene un valor dado, su valor se cogerá del entorno de shell que ejecuta el `docker-compose up` , lo que puede ser útil para pasar claves, por ejemplo.
- *links*: para definir relaciones entre contenedores.
- *ports*: para mapear los puertos donde el contenedor acepta conexiones.
- *volumes*: para definirir volúmes en el contenedor.
- *volumes_from*: para reusar los volúmenes de otro contenedor.

Aquí tenéis una lista completa y actualizada de las opciones que permite *docker-compose*.