# Que es docker

Docker", el software de TI, es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux.

Con DOCKER, puede usar los contenedores como máquinas virtuales extremadamente livianas y modulares. Además, obtiene flexibilidad con estos contenedores: puede crearlos, implementarlos, copiarlos y moverlos de un entorno a otro, lo cual le permite optimizar sus aplicaciones para la nube.

## Comandos de docker

Comandos imágenes

|  |  |
| --- | --- |
| **Construir una imagen** | docker build --build-arg -t <nombre\_imagen> . |
| **Construir una imagen detrás de un proxy** | docker build --build-arg http\_proxy=http://<proxy>:<port> --build-arg https\_proxy=http://<proxy>:<port> -t <nombre\_imagen> . |
| **Hacer pull de una imagen ya construida** | docker pull nginx:1.13-alpine |
| **Añadir un TAG a una imagen** | docker tag <nombre\_imagen> <nombre\_imagen>:<tag> |
| **Inspeccionar una imagen** | docker inspect <nombre\_imagen> |
| **Guardar imagen en un fichero TAR** | docker save <nombre\_imagen> > <fichero>.tar |
| **Eliminar imagen** | docker rmi <nombre\_imagen>:<TAG> |
| **Eliminar todas las imágenes** | docker rmi `docker images -q` |

Comandos contendores

|  |  |
| --- | --- |
| Ejecutar contenedor | docker run -v <unidad\_host>:<unidad\_docker> -it --name <nombre\_contenedor> -d <nombre\_imagen> |
| Eliminar contenedor | docker rm <nombre\_contenedor> |
| **Eliminar todos los contenedores parados** | docker rm `docker ps -a -q` |

|  |  |
| --- | --- |
| **Lista de imágenes:** | docker images |
| **Ver contenido de una imagen:** | docker run -it image\_name sh |
| **Lista contenedores en ejecución:** | docker ps -a |
| **Conectar a un contenedor en ejecución:** | docker exec -i -t a9ddb677957f /bin/bash |

|  |  |
| --- | --- |
| **Build Docker-Compose:** | docker-compose build |
| **Configurar Dockerfile para mantener el docker en ejecución:** | CMD tail -f /dev/null |
| **Up Docker-Compose segundo plano:** | docker-compose up -d |
| **Ver salida de Docker-Compose:** | docker-compose logs -f -t |
| **Detener un Docker-Compose:** | docker-compose down |
| **Detener un Docker-Compose y eliminar volúmenes:** | docker-compose down -v |
| **Ver Docker-Compose en ejecución:** | docker-compose ps |
| **Ejecutar un comando en un conenedor de docker-compose:** | docker-compose exec <nombre-servicio> /bin/bash |
| **Eliminar Docker-Compose detenidos:** | docker-compose rm |

# Docker mysql

|  |  |
| --- | --- |
| Paso 1: Traer mysql | docker pull mysql |
| Aqui traemos la imagen de docker, pero esta imagen no hace nada aun.  Las imágenes no guardan los datos al terminar su ejecución por ello crearemos una carpeta para guardar los datos de mysql y usamos un comando para guardar ese directorio. | Docker run -v  /*home/*felipe/Escritorio/Docker/Databases/mysql:/var/lib/mysql  -p 3306:3306  --name mysql  -e MYSQL\_ROOT\_PASSWORD=secret  -d mysql  --character-set-server=utf8  --collation-server=utf8\_unicode\_ci |
| Paso 2: -p es para el puerto, primero el puerto del contenedor y de ahí el puerto del anfitrión, que puede ser el puerto que uno decida | Paso 3: podemos asignarle un nombre con –name , en este caso –name mysql |
| Paso 4: asignamos la contraseña:  -e MYSQL\_ROOT\_PASSWORD=  contraseñacualquiera | Paso 5: elegimos un -d y la imagen de docker que elijamos en este caso al imagen de mysql  -d mysql |
| Paso 6: ponemos que sea utf8  --character-set-server=utf8 | Paso 7: se puede elegir una codificación pa la colección de datos ej: la Ñ  --collation-server=utf8\_unicode\_ci |
| Una vez puesto el comando se genera un hash ,  si se hace un docker ps muestra los contenedores que están corriendo y debería salir ese. | Para conectarse al contenedor ponemos: docker exec -it y el nombre o hash, mas el comando que se ejecutara, en este caso mysql.  Docker exec it mysql mysql -uroot -p |

Usando docker

Para este ejemplo usaremos el framework rails + mysql

<https://docs.docker.com/compose/rails/>(rails+ postgres)

Paso 1) Creamos un directorio donde vamos a trabajar. Luego creamos los siguientes archivos: Dockerfile, docker-compose.yml, Gemfile , Gemfile.lock, entrypoint.sh, database.yml.

Paso 2) En el archivo Dockerfile pondremos lo siguiente:

|  |  |
| --- | --- |
| FROM ruby:2.5  RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs  WORKDIR /myapp  COPY Gemfile /myapp/Gemfile  COPY Gemfile.lock /myapp/Gemfile.lock  RUN bundle install  COPY . /myapp  # Add a script to be executed every time the container starts.  COPY entrypoint.sh /usr/bin/  RUN chmod +x /usr/bin/entrypoint.sh  ENTRYPOINT ["entrypoint.sh"]  EXPOSE 3000  # Start the main process.  CMD ["rails", "server", "-b", "0.0.0.0"] | Aqui definimos las dependencias del contenedor docker.  Eso podria el codigo de la aplicación dentro de una imagen la cual construira el contendor con Ruby, Bundler y todas als dependencias dentro del contenedor.  Referencias  [Docker user guide](https://docs.docker.com/get-started/)  [Dockerfile reference](https://docs.docker.com/engine/reference/builder/). |

En el archivo docker-compose.yml: (nota: debe estar identado o no funciona)

|  |  |
| --- | --- |
| version: "2"  services:  db:  image: mysql:5.7  restart: always  environment:  MYSQL\_ROOT\_PASSWORD: password  MYSQL\_DATABASE: app  MYSQL\_USER: user  MYSQL\_PASSWORD: password  DB\_HOST: db  ports:  - "3306:3306"  web:  build: .  command: bash -c "rm -f tmp/pids/server.pid && bundle exec rails s -p 3000 -b '0.0.0.0'"  volumes:  - .:/myapp  ports:  - "3000:3000"  depends\_on:  - db  links:  - db  environment:  DB\_USER: root  DB\_NAME: app  DB\_PASSWORD: password  DB\_HOST: db | Finally, docker-compose.yml is where the magic happens. This file describes the services that comprise your app (a database and a web app), how to get each one’s Docker image (the database just runs on a pre-made PostgreSQL image, and the web app is built from the current directory), and the configuration needed to link them together and expose the web app’s port. |

En entrypoint.sh:

|  |  |
| --- | --- |
| #!/bin/bash  set -e  # Remove a potentially pre-existing server.pid for Rails.  rm -f /myapp/tmp/pids/server.pid  # Then exec the container's main process (what's set as CMD in the Dockerfile).  exec "$@" | Next, provide an entrypoint script to fix a Rails-specific issue that prevents the server from restarting when a certain server.pid file pre-exists. This script will be executed every time the container gets started. entrypoint.sh consists of: |

En database.yml:

default: &default

adapter: mysql2

encoding: utf8

pool: 5

username: root

password:

host: localhost

database: <%= ENV['DB\_NAME'] %>

username: <%= ENV['DB\_USER'] %>

password: <%= ENV['DB\_PASSWORD'] %>

host: <%= ENV['DB\_HOST'] %>

development:

<<: \*default

test:

<<: \*default

production:

<<: \*default

En Gemfile:

|  |  |
| --- | --- |
| source 'https://rubygems.org'  gem 'rails', '~>5' | Next, create a bootstrap Gemfile which just loads Rails. It’ll be overwritten in a moment by rails new. |

Y en Gemfile.lock nada.

Paso 3) Usamos el comando: docker-compose run --no-deps web rails new . --force --database=mysql

Paso 4) Usamos el comando “ sudo chown -R $USER:$felipe . “ para dar permisos en el contenedor.

Paso 5) Usamos el comando: docker-compose build

Paso 6) Copiamos el contenido del archivo database.yml o lo reemplazamos en config database.yml.

Paso 7) Usamos los comandos:

docker-compose up

docker-compose run web rake db:create (aveces este no es necesario ya que crea la bd en anterior)

Para ver la página web entramos a la siguiente url.

http://localhost:3000

En el caso de la conexión a la base de datos el puerto seria 3306, en caso que no conecte es el puerto 3307.