

MANUAL TÉCNICO

Contenido

Docker	2
Pasos para realizar una contenerización:.....	2
Instalar Docker, Esto podemos hacerlo con los siguientes comandos en Linux:	2
<i>FORMA 1. Usando el repositorio</i>	2
<i>FORMA 2. instalar el Docker engine-comunity</i>	3
Instalar compose	3
Crear archivos Dockerfile	4
Crear archivo Docker Compose .yaml	4
Explicación del archivo Docker-Compose:	5
1. Cabeceras	5
Explicación de los Docker files.	6

Docker:

Es una herramienta que automatiza el despliegue de aplicaciones dentro de contenedores de software proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos, con lo cual logramos que nuestras aplicaciones sean portables y puedan correr en cualquier sistema operativo siempre y cuando tengamos Docker instalado.

Pasos para realizar una contenerización:

Instalar Docker, Esto podemos hacerlo con los siguientes comandos en Linux:

FORMA 1. Usando el repositorio

- a. Actualizar el índice del paquete apt

```
sudo apt-get update
```

- b. Instalar paquetes para permitir que apt use un repositorio sobre HTTPS

```
sudo apt-get install \  
    apt-transport-https \  
    ca-certificates \  
    curl \  
    gnupg-agent \  
    software-properties-common
```

- c. Añadir la llave oficial de Docker GPG

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- d. Verificar que se tenga la llave con la huella 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

```
$ sudo apt-key fingerprint 0EBFCD88
```

```
pub   rsa4096 2017-02-22 [SCEA]  
      9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>  
sub   rsa4096 2017-02-22 [S]
```

e. configurar el repositorio estable con el siguiente comando

```
$ sudo add-apt-repository \  
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
    $(lsb_release -cs) \  
    stable"
```

FORMA 2. instalar el Docker engine-community

a. Actualizar el índice del paquete apt

```
sudo apt-get update
```

b. Instalar la última versión de Docker engine

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Instalar compose

f. Obtener la versión estable de Docker compose

```
$ sudo curl -L \  
    "https://github.com/docker/compose/releases/download/1.24.1/docker-compose- \  
    $(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

g. Darle permisos al binario ejecutable

```
sudo chmod +x /usr/local/bin/docker-compose
```

h. Pruebe la instalación

```
$ docker-compose --version  
  
docker-compose version 1.24.1, build 1110ad01
```

10/10/2019

Crear archivos Dockerfile

Se creara uno para el backend y otro para el frontend.

```
docker-compose.yml x Dockerfile — Frontend ●
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /code2
WORKDIR /code2
COPY requirements.txt /code2/
RUN pip install -r requirements.txt
COPY . /code2/
```

```
Dockerfile — Backend ● docker-compose.yml x
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
EXPOSE 8000
```

Crear archivo Docker Compose .yml

Cabe mencionar que por cada servicio en Services se tendrá un contenedor, por tal motivo nuestro proyecto contara con tres contenedores los cuales son:

1. db -> Base de datos en la cual usamos una imagen de postgres
2. frontend -> realizado en Python con el framework Django
3. backend -> realizado en Python con el framework Django

```
version: '3'
services:
  db:
    image: postgres
  backend:
    build: ./Backend
    command: >
      sh -c "python Backend/manage.py makemigrations &&
        python Backend/manage.py migrate &&
        python Backend/manage.py runserver 0.0.0.0:8000"
    volumes:
      - ./code
    ports:
      - "8000:8000"
    depends_on:
      - db
  frontend:
    build: ./Frontend
    command: python Frontend/manage.py runserver 0.0.0.0:8001
    volumes:
      - ./code2
    ports:
      - "8001:8001"
    depends_on:
      - backend
```

Explicación del archivo Docker-Compose:

Contiene las configuraciones de todos nuestros contenedores como cuales dependen de cuáles, los puertos que utilizan la ubicación de los Docker files entre otras cosas.

1. Cabeceras:

- a. Versión: Los archivos del compose son versionados por lo que es necesario indicar la versión de las instrucciones que queremos darle. Recordemos que siempre habrá compatibilidad hacia atrás.
- b. Services: Aquí irán todos los contenedores que vayamos a crear en nuestro caso solo creamos tres especificados mas adelante.
- c. Image: Se pone la imagen de Docker que se utilizara en el contenedor
- d. Build: especificamos la ruta donde se encuentra nuestro dockerfile que utilizaremos para el contenedor.
- e. Command: Los comandos aquí ingresados se ejecutarán en la imagen de nuestro contenedor.
- f. Volumes: Hacemos que el directorio actual se mapee directamente con el de /code y /code2 de nuestra aplicación.
- g. Ports: mapeamos los puertos locales 8000 y 8001 al servidor del host. Esto permitirá que accediendo a Localhost:8000 o Localhost:8001 podamos probar el sitio generado.
- h. Depends_on: expresa una dependencia entre los servicios de la siguiente manera el frontend depende del backend y el backend depende de la base de datos.
- i. Cabeceras con nombres de los contenedores:
 - db
 - frontend
 - backend

Explicación de los Docker files.

Contiene la receta para la creación automática de las imágenes que utilizaremos.

```
# indicamos la imagen base sobre la que se construirá la app dentro del contenedor.  
FROM python:3  
  
#indicamos el enviroment donde generaremos nuestra app con django  
#cabe resaltar que debe de ser el mismo env para el frontend que para el backend.  
ENV PYTHONUNBUFFERED 1  
  
#le indicamos que cree la carpeta /code  
RUN mkdir /code  
  
#Ahora debemos indicar la carpeta sobre la que vamos a trabajar  
WORKDIR /code  
  
#copiamos el archivo requirements.txt a la carpeta que hemos creado “/code/”  
COPY requirements.txt /code/  
  
#corremos el commando pip sobre cada instruccion que se encuentra en nistro archive requirements.  
RUN pip install -r requirements.txt  
  
#copiamos todo el contenido de la carpeta /code/  
COPY . /code/  
  
#le decimos sobre que puerto vamos a trabajar  
EXPOSE 8000
```