

# Programación Python para Big Data - Tarea lección 2

Kevin Martínez García

8 de junio de 2022

## Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. MongoDB</b>	<b>2</b>
2.1. MongoDB con Docker . . . . .	2
2.2. MongoDB con docker-compose . . . . .	3
<b>3. PostgreSQL</b>	<b>3</b>
3.1. PostgreSQL con Docker . . . . .	3
3.2. PostgreSQL con docker-compose . . . . .	5

# 1. Introducción

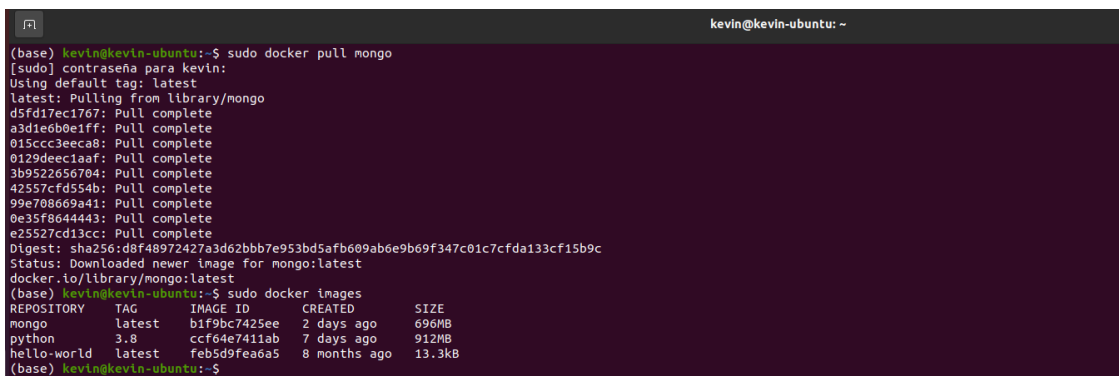
En la actividad relacionada a la Lección 2, se nos solicitó generar un manual de instalación de los servicios *MongoDB* y *PostgreSQL* usando las herramientas *Docker* y *docker-compose* vistas en sesiones de teoría. Para hacerlo, haremos uso de imágenes previamente diseñadas disponibles en *DockerHub* y adjuntaremos capturas del procedimiento que llevemos a cabo.

## 2. MongoDB

MongoDB es un sistema de base de datos *NoSQL*, orientado a documentos y de código abierto que, en lugar de usar tablas para almacenar la información, utiliza documentos de tipo BSON con un esquema dinámico [1]. En la actualidad, se trata de un servicio ampliamente utilizado en diferentes ámbitos tecnológicos y resulta de especial interés para esta asignatura. Vamos a comenzar exponiendo su despliegue mediante el uso de Docker en un sistema Ubuntu.

### 2.1. MongoDB con Docker

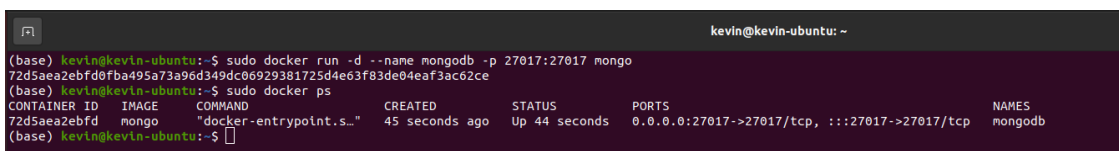
Para comenzar con el despliegue de MongoDB con Docker, acudimos a su respectiva página en DockerHub [2]. Aquí podemos ver como podemos obtener una imagen ya diseñada mediante el comando `docker pull mongo`. Abrimos un terminal, ejecutamos el comando y esperamos a que la imagen se descargue en nuestro sistema. Si al finalizar ejecutamos el comando `docker images` deberíamos poder verla disponible, tal y como aparece en la Figura 1 a continuación.



```
(base) kevin@kevin-ubuntu:~$ sudo docker pull mongo
[sudo] contraseña para kevin:
Using default tag: latest
latest: Pulling from library/mongo
d5fd17ec1767: Pull complete
a3d1e6b0e1ff: Pull complete
015ccc3eeca8: Pull complete
0129deec1aaf: Pull complete
3b9522656704: Pull complete
42557cfd554b: Pull complete
99e708669a41: Pull complete
0e35f8644443: Pull complete
e25527cd13cc: Pull complete
Digest: sha256:d8f48972427a3d62bbb7e953bd5afb609ab6e9b69f347c01c7cfda133cf15b9c
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
(base) kevin@kevin-ubuntu:~$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
mongo                latest          b1f9bc7425ee   2 days ago     696MB
python               3.8             ccf64e7411ab   7 days ago     912MB
hello-world          latest          feb5d9fea6a5   8 months ago   13.3kB
(base) kevin@kevin-ubuntu:~$
```

Figura 1: Servicio MongoDB obtenido a través de DockerHub

A continuación, podemos arrancar el servicio haciendo uso de `docker run`. Para ello, lanzamos la instrucción `docker run -d --name mongoddb -p 27017:27017 mongo` lo que ejecutaría nuestro servicio en segundo plano. Si ahora ejecutamos la instrucción `docker ps` podríamos ver como efectivamente nuestro servicio está ejecutándose tal y como aparece en la Figura 2.



```
(base) kevin@kevin-ubuntu:~$ sudo docker run -d --name mongoddb -p 27017:27017 mongo
72d5aea2ebfd0fba495a73a96d349dc06929381725d4e63f83de04eaf3ac62ce
(base) kevin@kevin-ubuntu:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
72d5aea2ebfd   mongo    "docker-entrypoint.s..." 45 seconds ago Up 44 seconds 0.0.0.0:27017->27017/tcp, :::27017->27017/tcp   mongoddb
(base) kevin@kevin-ubuntu:~$
```

Figura 2: Servicio MongoDB obtenido a través de DockerHub

Antes de proceder con el siguiente apartado, nos aseguramos de parar el servicio utilizando el comando `docker kill mongoddb`.

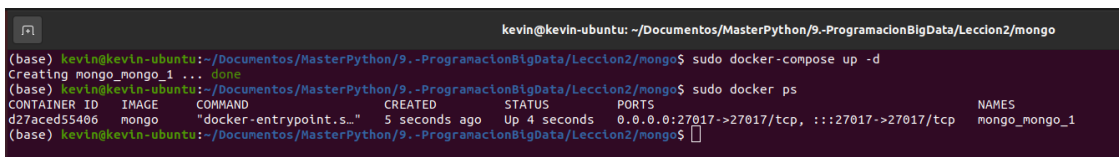
## 2.2. MongoDB con docker-compose

Para lanzar nuestro servicio desde *docker-compose* debemos generar el fichero `docker-compose.yml` con el contenido que aparece en la Figura 3 a continuación.

```
# Use root/example as user/password credentials
version: '3.1'
services:
  mongo:
    image: mongo
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
```

Figura 3: `docker-compose.yml` para MongoDB

En este caso, como ya disponemos de la imagen de MongoDB descargada desde DockerHub, no necesitamos ejecutar `docker-compose build` y podemos pasar directamente a lanzar el servicio. Para ello, nos ubicamos en el directorio dónde guardemos el fichero `docker-compose.yml` y ejecutamos la instrucción `docker-compose up -d`, lo que lanzaría el servicio en segundo plano. Para comprobar que efectivamente se encuentra activo, ejecutamos `docker ps` tal y como aparece en la Figura 4.



```
kevin@kevin-ubuntu: ~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/mongo
(base) kevin@kevin-ubuntu:~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/mongo$ sudo docker-compose up -d
Creating mongo_mongo_1 ... done
(base) kevin@kevin-ubuntu:~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/mongo$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
d27aced55406   mongo         "docker-entrypoint.s..." 5 seconds ago  Up 4 seconds  0.0.0.0:27017->27017/tcp, :::27017->27017/tcp  mongo_mongo_1
(base) kevin@kevin-ubuntu:~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/mongo$
```

Figura 4: Servicio MongoDB ejecutado con `docker-compose`

De nuevo, podemos terminar la ejecución del servicio mediante la instrucción `docker kill mongo_mongo_1`.

## 3. PostgreSQL

PostgreSQL es un potente sistema de base de datos relacional de objetos de código abierto con más de 30 años de desarrollo activo que le ha valido una sólida reputación de fiabilidad, solidez de las funciones y rendimiento [3]. De nuevo, comenzaremos exponiendo su despliegue haciendo uso de Docker.

### 3.1. PostgreSQL con Docker

Empezamos acudiendo a la página de PostgreSQL en DockerHub [4] y usamos la instrucción `docker pull postgres` para descargarla en nuestro sistema. Cuando finalice, ejecutamos el comando `docker images` y deberíamos poder verla disponible como se muestra en la Figura 5 a continuación.

```
kevin@kevin-ubuntu: ~  
(base) kevin@kevin-ubuntu:~$ sudo docker pull postgres  
Using default tag: latest  
latest: Pulling from library/postgres  
42c077c10790: Pull complete  
3c2043bc3122: Pull complete  
12e1d6a2dd60: Pull complete  
9ae1101c4060: Pull complete  
fb05d2fd4701: Pull complete  
9785a964a677: Pull complete  
16fc798b0e72: Pull complete  
f1a0bfa2327a: Pull complete  
fd2d68720749: Pull complete  
83b23beac012: Pull complete  
7962517582d4: Pull complete  
6b4a569b8013: Pull complete  
ad029fbc8984: Pull complete  
Digest: sha256:2d1e636f07781d4799b3f2edbff78a0a5494f24c4512cb56a83ebfd0e04ec074  
Status: Downloaded newer image for postgres:latest  
docker.io/library/postgres:latest  
(base) kevin@kevin-ubuntu:~$ sudo docker images  
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE  
nongp                latest          b1f9bc7425ee    2 days ago     696MB  
python               3.8             ccf64e7411ab    7 days ago     912MB  
postgres             latest          5b21e2e06aab    7 days ago     376MB  
hello-world          latest          feb5d9fea6a5    8 months ago   13.3kB  
(base) kevin@kevin-ubuntu:~$
```

Figura 5: Servicio PostgreSQL obtenido a través de DockerHub

A continuación, arrancamos el servicio haciendo uso de `docker run`. Para ello, lanzamos la instrucción `docker run -d --name postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 postgres` lo que ejecutaría el servicio en segundo plano. Si ahora ejecutamos la instrucción `docker ps` podríamos ver como efectivamente el servicio se encuentra en ejecución tal y como aparece en la Figura 6.

```
kevin@kevin-ubuntu: ~  
(base) kevin@kevin-ubuntu:~$ sudo docker run -d --name postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 postgres  
9358ee20f52124ccb08ad77e7f252621a7327b9b54eb158f38910d3003f57dc1  
(base) kevin@kevin-ubuntu:~$ sudo docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES  
9358ee20f521   postgres "docker-entrypoint.s..." 42 seconds ago Up 41 seconds 0.0.0.0:5432->5432/tcp, :::5432->5432/tcp postgres  
(base) kevin@kevin-ubuntu:~$
```

Figura 6: Servicio PostgreSQL obtenido a través de DockerHub

Antes de proceder con el siguiente apartado, nos aseguramos de parar el servicio utilizando el comando `docker kill postgres`.

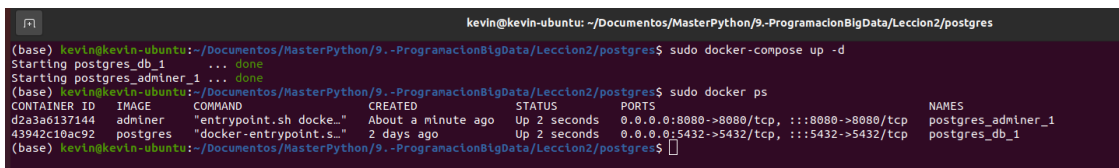
### 3.2. PostgreSQL con docker-compose

Para finalizar, realizamos el despliegue de PostgreSQL haciendo uso de docker-compose. De nuevo, acudimos a la página DockerHub de PostgreSQL y generamos el fichero `docker-compose.yml` como aparece en la Figura 7. Lanzaremos también el servicio `adminer` ya que haremos uso del mismo en próximas lecciones.

```
# Use postgres/example user/password credentials
version: '3.1'
services:
  db:
    image: postgres
    restart: always
    ports:
      - 5432:5432
    environment:
      POSTGRES_PASSWORD: example
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

Figura 7: docker-compose.yml para PostgreSQL

En este caso, como ya disponemos de la imagen de PostgreSQL descargada desde DockerHub, no necesitamos ejecutar `docker-compose build` y podemos pasar directamente a lanzar el servicio. Para ello, nos ubicamos en el directorio dónde guardemos el fichero `docker-compose.yml` y ejecutamos la instrucción `docker-compose up -d`, lo que lanzaría el servicio en segundo plano. Para comprobar que efectivamente se encuentra activo, ejecutamos `docker ps` tal y como aparece en la Figura 8.



```
kevin@kevin-ubuntu: ~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/postgres
(base) kevin@kevin-ubuntu:~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/postgres$ sudo docker-compose up -d
Starting postgres_db_1 ... done
Starting postgres_adminer_1 ... done
(base) kevin@kevin-ubuntu:~/Documentos/MasterPython/9.-ProgramacionBigData/Leccion2/postgres$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                               NAMES
d2a3a6137144   adminer    "entrypoint.sh docke..." About a minute ago Up 2 seconds    0.0.0.0:8080->8080/tcp, :::8080->8080/tcp postgres_adminer_1
43942c10ac92   postgres  "docker-entrypoint.s..." 2 days ago      Up 2 seconds    0.0.0.0:5432->5432/tcp, :::5432->5432/tcp postgres_db_1
```

Figura 8: Servicio PostgreSQL ejecutado con docker-compose

Para terminar, podemos parar la ejecución del servicio con `docker kill postgres_db_1` y `docker kill postgres_adminer_1`.

## Referencias

- [1] Wikipedia. (2020). MongoDB. [online] Available at: <https://es.wikipedia.org/wiki/MongoDB>.
- [2] hub.docker.com. (n.d.). Docker Hub. [online] Available at: [https://hub.docker.com/\\_/mongo](https://hub.docker.com/_/mongo).
- [3] The PostgreSQL Global Development Group (2019). PostgreSQL: The world's most advanced open source database. [online] Postgresql.org. Available at: <https://www.postgresql.org/>.
- [4] hub.docker.com. (n.d.). Docker Hub. [online] Available at: [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres).