

# Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones



## Desarrollo De Aplicaciones Para Sistemas Ubicuos

Docker con MQTT

**Estudiante**

Juan Sebastián Realpe González

**Código**

100616021378

**Profesor:**

Ing. Julian Andrés Bolaños

Popayán, Cauca

2021

# Índice

<b>1. Práctica 5 Docker Inicio</b>	<b>3</b>
1.1. Instalación . . . . .	3
<b>2. Práctica 6 Docker Fundamentals</b>	<b>4</b>
2.1. Manejo de imágenes . . . . .	4
2.2. Dockerfile . . . . .	6
<b>3. Tareas</b>	<b>8</b>
3.1. MQTT pub-sub . . . . .	8
3.2. MYSQL . . . . .	9

# 1. Práctica 5 Docker Inicio

Esta práctica tiene como objetivo introducir al estudiante en el manejo de herramientas para el despliegue y empaquetamiento de aplicaciones. La herramienta a trabajar será Docker, la cual se basa en la factorización de software y la optimización del hardware disponible, haciendo una analogía a las máquinas virtuales las cuales dependen de la implementación completa de una instancia de sí misma. Por el lado de Docker, se trata de un concepto novedoso al componerse de imágenes que comparten muchas veces el mismo kernel de Linux y en muchos casos hasta el sistema operativo.

## 1.1. Instalación

Para proceder a la instalación de Docker es necesario contar con el sistema operativo LINUX, en cualquiera de sus distribuciones, ya que el mismo está construido para este tipo de distribuciones. Para sistemas operativos Windows es posible bien sea montar una máquina virtual utilizando VirtualBOX, o manejar el software “Docker Desktop” que permite manejar Docker en WSL2 con una máquina virtual simulada.

- Se descarga el instalador en <https://docs.docker.com/engine/install/>
- Se realiza la instalación de WSL2 para el correcto funcionamiento de Docker Desktop siguiendo los pasos de <https://docs.microsoft.com/en-us/windows/wsl/install>
- En una ventana de comandos o en powershell se escribe el comando `wsl -install`
- Se obtiene la siguiente ventana

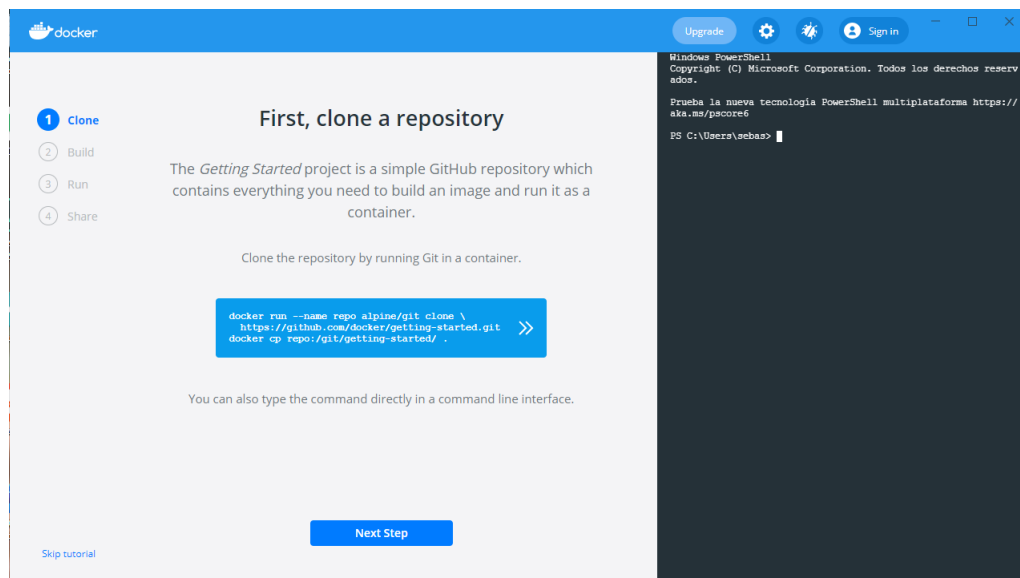


Figura 1: Ventana de Docker

- Para comprobar el funcionamiento de Docker, se corre el siguiente comando `docker run hello-world`

```
C:\WINDOWS\system32\cmd.exe
C:\Users\sebas>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2d029716123e: Pull complete
Digest: sha256:393b81f0ea5a98a7335d7ad44be96fe76ca8eb2aaa76950eb8c989ebf2b78ec0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

C:\Users\sebas>
```

Figura 2: Hola mundo en Docker

## 2. Práctica 6 Docker Fundamentals

Esta práctica tiene como objetivo introducir al estudiante en el manejo de herramientas para el despliegue y empaquetamiento de aplicaciones. Se revisarán algunos de los aspectos fundamentales para introducir al estudiante en el entendimiento de la herramienta Docker para el manejo de aplicaciones.

### 2.1. Manejo de imágenes

Docker es una comunidad abierta, y en el Docker hub existen múltiples imágenes que se pueden descargar, usar y modificar.

Para descargar esta y cualquier imagen se debe tener en cuenta el siguiente comando:

```
docker pull postgres
```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\sebas>docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
f841ed8bac72: Pull complete
858331474a69: Pull complete
4ee930e806cf: Pull complete
6748b551eaab: Pull complete
3885f0ed3c9a: Pull complete
f99ac1b6637a: Pull complete
c67f57ae254c: Pull complete
1cbe9dcd337: Pull complete
1a3eccc077fb: Pull complete
d2d31d2dbf98: Pull complete
c407080f1477: Pull complete
57838d6b0005: Pull complete
25c560526375: Pull complete
Digest: sha256:90c1dddb38152f7260014bc4d3d10209cebbd5e25346cea07b00efd8e36028e
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest

C:\Users\sebas>
```

Figura 3: Creación imagen postgres

Para correr la imagen:

```
docker run -e POSTGRES_PASSWORD = password postgres
```

```
C:\WINDOWS\system32\cmd.exe - docker run -e POSTGRES_PASSWORD=password postgres

C:\Users\sebas>docker run -e POSTGRES_PASSWORD=password postgres
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Etc/UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
ok

Success. You can now start the database server using:

    pg_ctl -D /var/lib/postgresql/data -l logfile start

waiting for server to start....2021-09-27 21:47:08.915 UTC [48] LOG: starting PostgreSQL 13.4 (Debian 13.4-1.pgdg110+1)
on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2021-09-27 21:47:08.918 UTC [48] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-09-27 21:47:08.927 UTC [49] LOG: database system was shut down at 2021-09-27 21:47:08 UTC
2021-09-27 21:47:08.933 UTC [48] LOG: database system is ready to accept connections
done
server started

/usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

2021-09-27 21:47:09.054 UTC [48] LOG: received fast shutdown request
waiting for server to shut down....2021-09-27 21:47:09.058 UTC [48] LOG: aborting any active transactions
2021-09-27 21:47:09.060 UTC [48] LOG: background worker "logical replication launcher" (PID 55) exited with exit code 1
PostgreSQL init process complete; ready for start up.

2021-09-27 21:47:09.182 UTC [1] LOG: starting PostgreSQL 13.4 (Debian 13.4-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2021-09-27 21:47:09.182 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
2021-09-27 21:47:09.182 UTC [1] LOG: listening on IPv6 address "::", port 5432
2021-09-27 21:47:09.189 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-09-27 21:47:09.196 UTC [67] LOG: database system was shut down at 2021-09-27 21:47:09 UTC
2021-09-27 21:47:09.203 UTC [1] LOG: database system is ready to accept connections
```

Figura 4: Correr imagen postgres

Con el comando `docker images` se puede ver las imágenes creadas.

```
C:\Users\sebas>docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            latest             5861c038d674       3 days ago         371MB
hello-world         latest            feb5d9fea6a5       3 days ago         13.3kB
```

Figura 5: Imágenes creadas en Docker

Con el comando `docker ps` se puede ver los containers creados

```
C:\Users\sebas>docker ps

CONTAINER ID   IMAGE      COMMAND                  CREATED             STATUS              PORTS              NAMES
d73b7a39028a   postgres  "docker-entrypoint.s..." About a minute ago  Up About a minute  5432/tcp           hopeful_rhodes
```

Figura 6: Containers creados en Docker

## 2.2. Dockerfile

El uso de dockerfile puede ser adecuado para lleva a cabo implementaciones de nuestra aplicación en tiempo real, tomando el ejemplo de la guía se obtiene en `localhost:8060` lo siguiente:

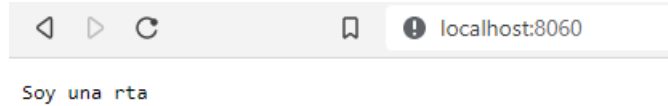


Figura 7: Salida localhost:8060, ejemplo 1

Se corre el comando `docker build -t imagenode .` para crear una imagen con el tag `imagenode` que podremos correr posteriormente con el comando `docker run imagenode` como se muestra en las siguientes figuras.

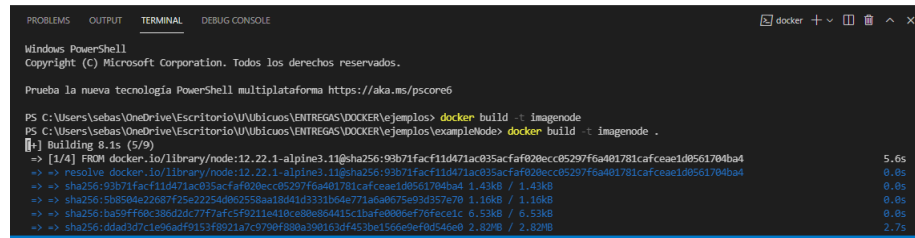


Figura 8: Crear imagen con etiqueta

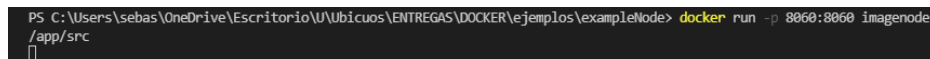


Figura 9: Correr imagen con etiqueta

En la ventana de docker se muestra lo siguiente:

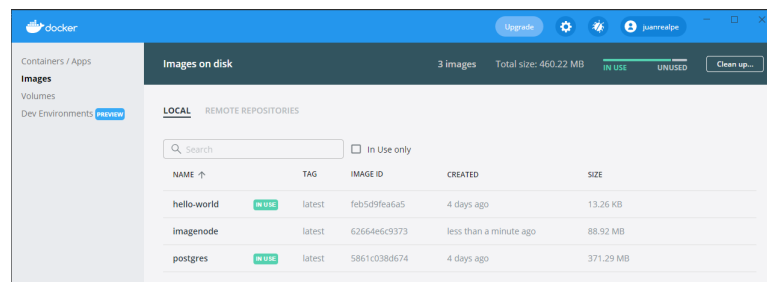


Figura 10: Ventana docker

En localhost:8060, se muestra el siguiente mensaje:

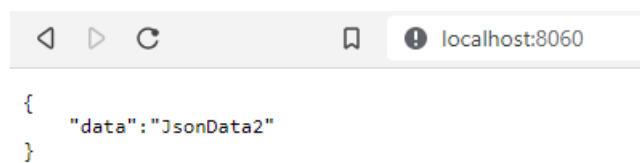


Figura 11: Salida localhost:8060, ejemplo 2

Para que sea posible observar desde el contenedor un archivo cualquiera o un directorio es necesario el uso de volúmenes, para este caso se creó una nueva imagen llamada *imagenode-tres*, y se probó de la siguiente forma:



Figura 12: Uso de volúmenes

En localhost:8060, se muestra el siguiente mensaje:

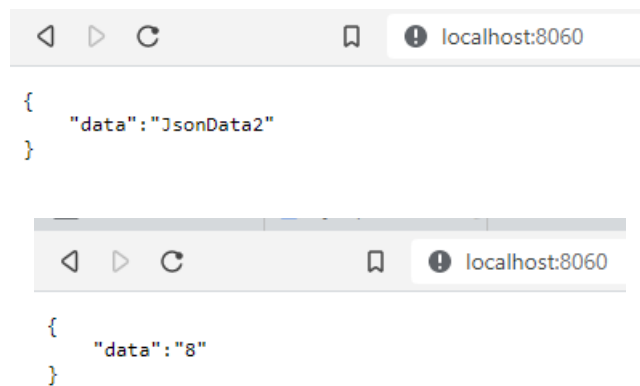


Figura 13: Salida localhost:8060, ejemplo 3

### 3. Tareas

#### 3.1. MQTT pub-sub

Se usa la imagen ruimarinho-mosquitto para implementar un nuevo ejemplo, con el siguiente comando

`docker pull postgres :`

```
C:\Program Files\mosquitto>docker run --rm -it ruimarinho/mosquitto

1632793015: mosquitto version 1.4.14 (build date 2018-02-26 21:40:04+0000) starting
1632793015: Using default config.
1632793015: Opening ipv4 listen socket on port 1883.
1632793015: Opening ipv6 listen socket on port 1883.
1632793226: New connection from 172.17.0.3 on port 1883.
1632793226: New client connected from 172.17.0.3 as mosqsub|1-0055e4e0e585 (c1, k60).
1632793346: New connection from 172.17.0.4 on port 1883.
1632793346: New client connected from 172.17.0.4 as mosqpub|1-5a390a1bb163 (c1, k60).
1632793346: Client mosqpub|1-5a390a1bb163 disconnected.
```

Figura 14: Imagen ruimarinho-mosquitto

Luego, se procede a crear probar el método subscribe y publish, para este caso, se usaron los siguientes comandos:

- Para subscribe

```
Administrador: Símbolo del sistema - docker run --rm -it --link agitated_burnell ruimarinho/mosquitto mosquitto_sub -h agitated_burnell -t "#"

C:\WINDOWS\system32>docker run --rm -it --link agitated_burnell ruimarinho/mosquitto mosquitto_sub -h agitated_burnell -t "#"
}
TV is ON
```

Figura 15: Comando subscribe

- Para publish



```
Administrador: Símbolo del sistema
C:\WINDOWS\system32>docker run --rm -it --link agitated_burnell ruimarinho/mosquitto mosquitto_pub -h agitated_burnell -t home-assistant/switch/1/on -m "TV is ON"

C:\WINDOWS\system32>
```

Figura 16: Comando publish

Se debe tener en cuenta que el nombre de la imagen corresponda a la imagen creada en Docker.

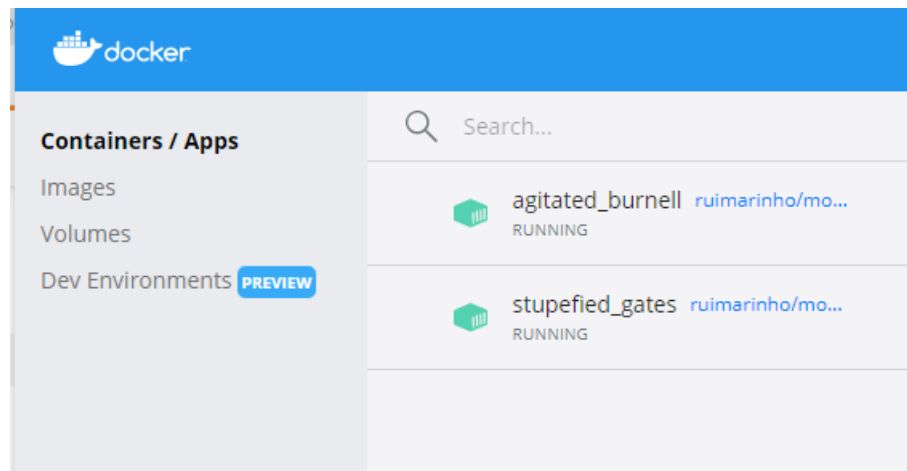


Figura 17: Ventana Docker

Y dentro de la otra imagen, debe mostrarse el mensaje publicado.

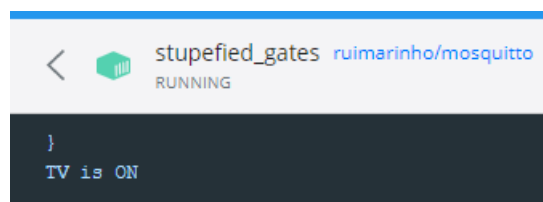


Figura 18: Imagen "stupefied\_gates"

### 3.2. MYSQL

Para MYSQL, se hace uso del archivo docker-compose.yml, y se hace la respectivas correcciones, para posteriormente probarlo.

The screenshot shows the Visual Studio Code interface with a Docker Compose file named `docker-compose.yml` open in the editor. The file defines two services: `app` and `mysql`. The `app` service uses the `imagenode-tres` image and maps port 8060 on the host to port 8060 in the container. It also sets environment variables for MySQL connection. The `mysql` service uses the `mysql:5.7` image and sets environment variables for the root user password and database name. A comment at the bottom of the file shows the command to run the containers: `# docker run -d --network todo-app --network-alias mysql -e MYSQL_ROOT_PASSWORD=secret`. The terminal at the bottom shows the output of the `docker-compose up` command, displaying warnings about deprecated TLS versions and a self-signed certificate.

```
1 version: "3.7"
2 services:
3   #docker run -dp 3000:3000 --network todo-app -e MYSQL_HOST=mysql -e MYSQL_USER=r
4   app:
5     image: imagenode-tres
6     ports:
7       - 8060:8060
8     environment:
9       MYSQL_HOST: mysql
10      MYSQL_USER: root
11      MYSQL_PASSWORD: test
12      MYSQL_DB: test
13     volumes:
14       - C:\Users\sebas\OneDrive\Escritorio\U\Ubicuos\ENTREGAS\DOCKER\ejemplos\ex
15 # docker run -d --network todo-app --network-alias mysql -e MYSQL_ROOT_PASSI
16 mysql:
17   image: mysql:5.7
18   environment:
19     MYSQL_ROOT_PASSWORD: secret
20     MYSQL_DATABASE: test
```

Terminal output:

```
app_1 | /app/src
mysql_1 | 2021-09-28T02:19:56.637332Z 0 [Warning] A deprecated TLS version TLSv1 is enab
mysql_1 | 2021-09-28T02:19:56.637374Z 0 [Warning] A deprecated TLS version TLSv1.1 is en
mysql_1 | 2021-09-28T02:19:56.637870Z 0 [Warning] CA certificate ca.pem is self signed.
mysql_1 | 2021-09-28T02:19:56.829308Z 1 [Warning] root@localhost is created with an empt
y password ! Please consider switching off the --initialize-insecure option.
```

Figura 19: Archivo docker-compose.yml

Y se puede observar en `localhost:8060` el mensaje que se uso en los anteriores puntos, gracias a la imagen *imagenode-tres*.

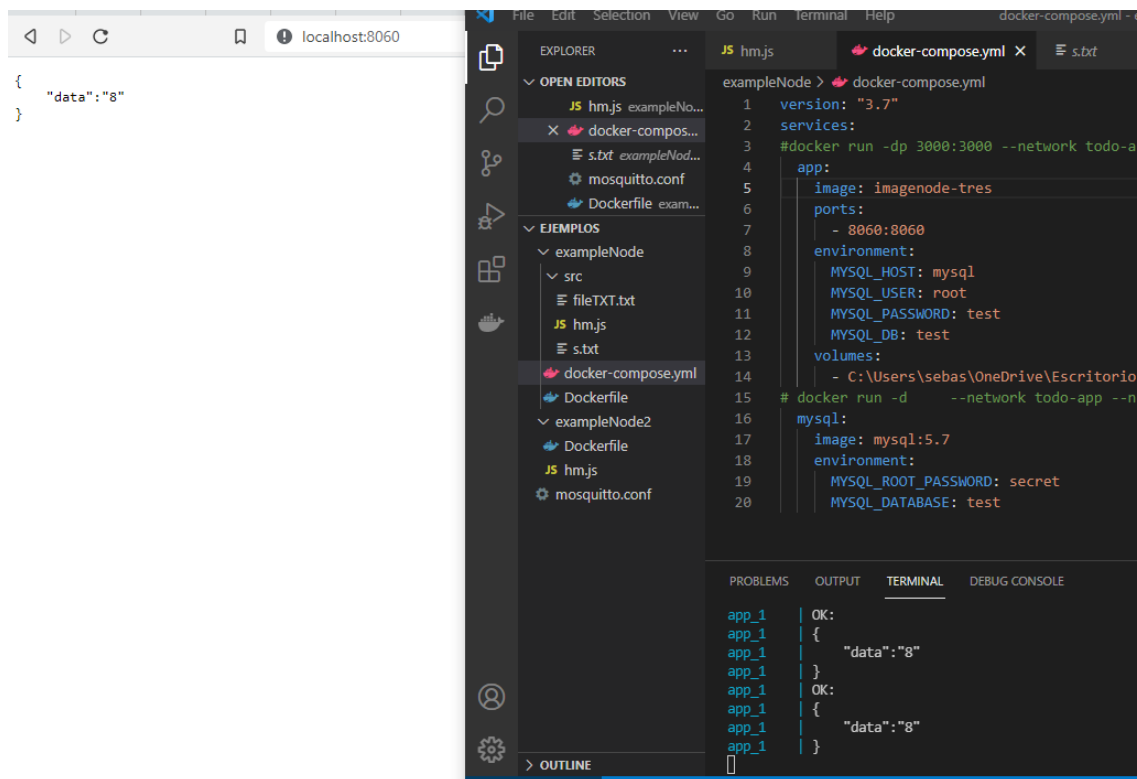


Figura 20: Salida localhost:8060 Tarea