

*Universidad de San Carlos de Guatemala*  
*Facultad de Ingeniería*  
*Escuela de Ciencias y Sistemas*  
*Análisis y Diseño de Sistemas 2*  
*Sección A-*  
*Ing. Ivonne Aldana*  
*Aux. Byron López*  
*2 semestre 2019*



## PRÁCTICA #1

Docker, Docker-compose  
Manual Técnico

Elmer Edgardo Alay Yupe  
201212945  
Guatemala, 6 de octubre de 2019

## INTRODUCCIÓN

Docker es una herramienta muy utilizada actualmente en el mundo DevOps ya que proporciona una manera de manejar la arquitectura de una aplicación a través de código dando la facilidad de versionar y estandarizar entornos repetibles de desarrollo, pruebas y producción.

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones de software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

Un ambiente ya sea de desarrollo, pruebas o producción comúnmente consisten en varios contenedores que en su conjunto forman un sistema unificado y totalmente aislado de la máquina host como de otros contenedores.

En esta aplicación lo que se busca es “contenerizar” una aplicación web la cual simulará a pequeña escala el funcionamiento de Tweeter. Para ello utilizarán herramientas para manejo de contenedores tales como Docker y Docker-compose para lograrlo.

# OBJETIVOS

## Generales

- Introducir al lector a las tecnologías de contenerización

## Específicos

- Aprender la estructura de un archivo Dockerfile para describir contenedores y entornos de ejecución.
- Aprender la estructura de un archivo de configuración de DockerCompose y describir satisfactoriamente un entorno de ejecución multi-contenedor.
- Crear una API REST

# REQUISITOS DEL SISTEMA

Esta aplicación fue desarrollada en una pc con las siguientes especificaciones:

- Procesador Intel Core i7-7500U
- RAM 16GB
- Sistema Operativo Windows 10 Pro

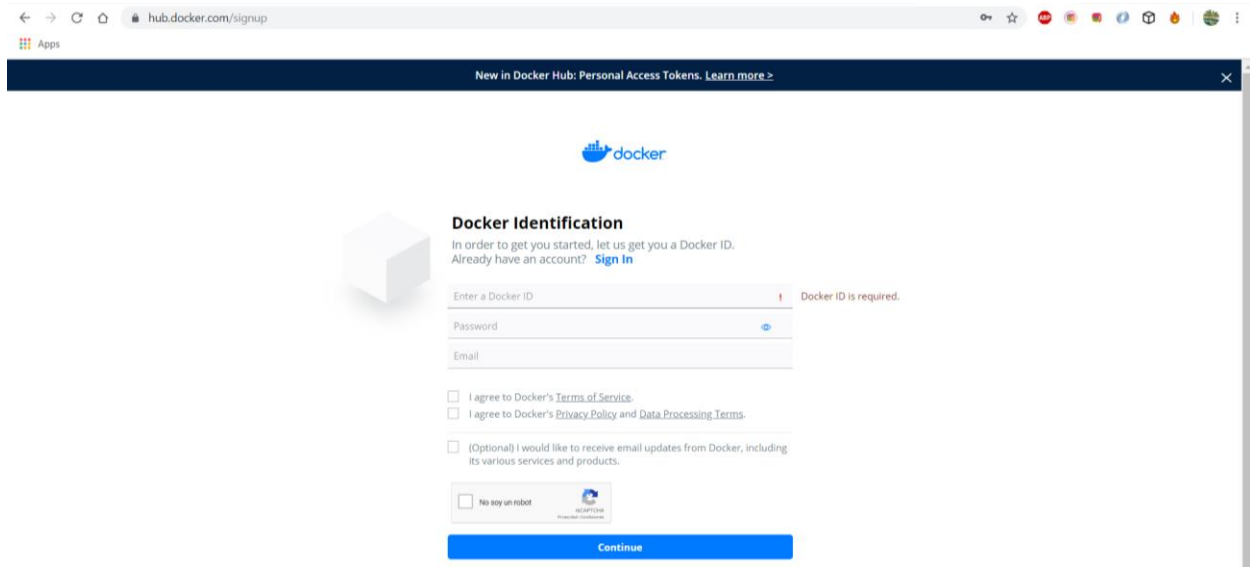
Para la correcta ejecución de esta aplicación se debe contar con los siguientes requisitos:

- Conexión a Internet
- Tener Docker instalado
- Tener Docker Compose instalado
- Disponer de un navegador web

# PROCESO GENERAL

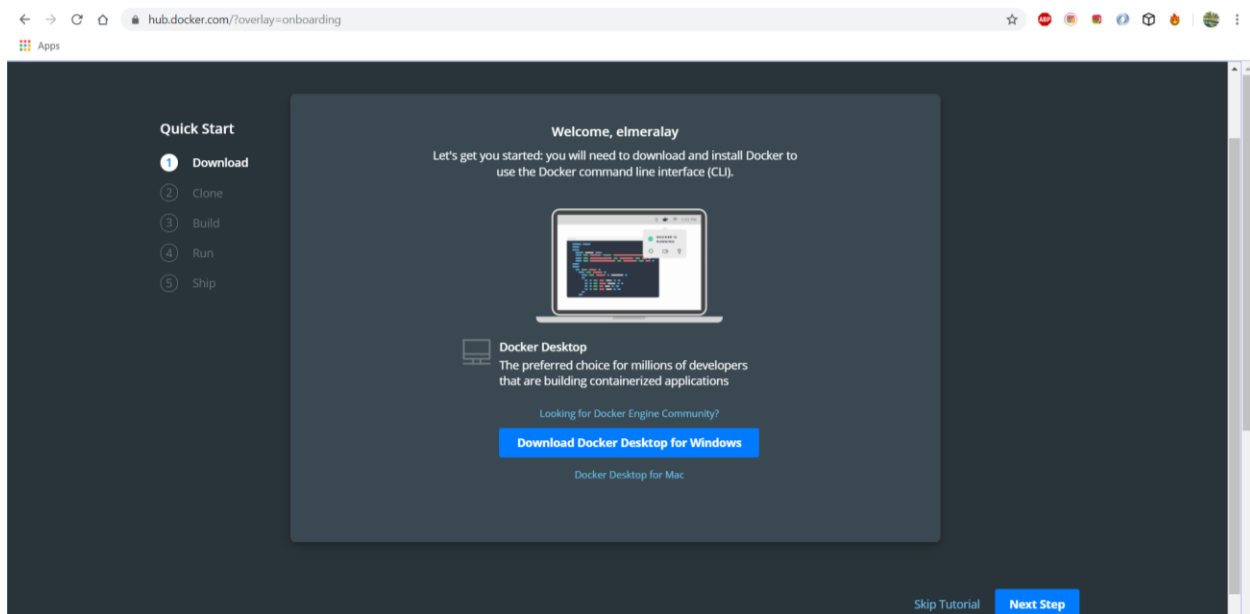
## Instalación de Docker

Para la instalación de Docker comenzaremos ingresando desde nuestro navegador preferido a la página oficial de Docker <https://hub.docker.com/>. Es necesario crear una cuenta en esta página porque sino no podremos instalar Docker. Para esto daremos click en “Get Started” ingresamos los datos que solicitan y damos click en continuar y seguimos hasta que tengamos la cuenta creada.

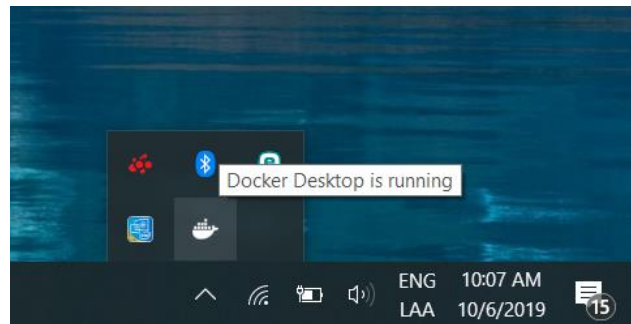


The screenshot shows the Docker Hub signup page. At the top, there's a navigation bar with the Docker logo and a link to "New in Docker Hub: Personal Access Tokens. Learn more >". Below the logo, the heading "Docker Identification" is followed by the text "In order to get you started, let us get you a Docker ID. Already have an account? [Sign In](#)". The form includes fields for "Enter a Docker ID", "Password", and "Email". A red error message "Docker ID is required." is visible next to the Docker ID field. Below the form, there are three checkboxes: "I agree to Docker's [Terms of Service](#)", "I agree to Docker's [Privacy Policy](#) and [Data Processing Terms](#)", and "(Optional) I would like to receive email updates from Docker, including its various services and products." At the bottom, there's a "No spam, un robot" checkbox and a "Continue" button.

Iniciamos sesión, y en la página principal seleccionamos la opción “Download Docker Desktop”, seleccionamos nuestro sistema operativo y le damos a descargar



Luego de la instalación deberíamos tener un ícono en la barra de tareas que indique que Docker ya está corriendo.



## Instalación de Docker Compose

Si instalaste Docker en Windows ya tienes instalado Docker Compose. Si lo instalaste en otro sistema operativo debes buscar en la página oficial los pasos a seguir.

## Composición de la aplicación

La aplicación es un simulador de Tweeter y para lograr su correcto funcionamiento se necesitan lo siguiente:

- Una base de datos
- Una API Rest
- Un frontend que consuma la api

Todo el código utilizado para la realización de la aplicación está disponible en un repositorio de github, el cuál su dirección es:

[https://github.com/ElmerAlay/tweet\\_app](https://github.com/ElmerAlay/tweet_app)

## Archivos Dockerfile

Un archivo dockerfile nos ayudará a colocar todas las instrucciones necesarias para obtener imágenes desde el repositorio de Dockerhub de una manera más sencilla que en consola de comandos. En otras palabras, es una receta que nos permite resumir los pasos para crear una imagen y colocarle ciertos parámetros y por medio de un solo comando se creará la imagen.

## Base de Datos

Se ha utilizado mysql como gestor de base de datos. Para crear el contenedor de la base de datos se utilizó un archivo Dockerfile. El cuerpo del archivo debe contener lo siguiente:

- *Instrucción From mysql:5.7*  
Significa que Docker descargará la imagen de mysql en su versión 5.7. Para conocer qué otras versiones existen visitar la página oficial de Docker hub.
- *Instrucción ENV MYSQL\_DATABASE db\_tweet\_app*  
Significa que se creará en mysql una base de datos de nombre db\_tweet\_app. Aquí podemos colocar cualquier nombre que deseemos.

- *Instrucción ENV MYSQL\_ROOT\_PASSWORD 1234*

Al descargar la imagen de mysql se crea por defecto el usuario root, Docker genera un password para este usuario, pero si lo queremos cambiar colocamos esta instrucción. Para la aplicación se utilizó la contraseña para root 1234.

- *Instrucción COPY ./tweetApp\_script.sql/ /docker-entrypoint-initdb.d/*

Para la bd a utilizar se utilizó un script donde vienen todas las creaciones de las tablas y demás. Con esta instrucción estamos copiando ese script en un archivo llamado Docker-entrypoint-initdb.d dentro del contenedor. Este archivo lo que hace es que cuando se construye el dockerfile automáticamente se ejecuta el script y se crean las tablas.

## API

Para la API Rest se utilizó el lenguaje nodejs con express. También se ha creado un archivo Dockerfile para este parte de la aplicación. Contiene lo siguiente:

- *Instrucción From node:12*

Le indicamos a Docker que queremos utilizar la imagen de node js en su versión 12.

- *Instrucción WORKDIR /usr/src/app*

Le indicamos a Docker que crearemos una carpeta llamada app dentro del directorio /usr/src en el contenedor y nos moveremos a ella.

- *Instrucción COPY package\*.json ./*

Aquí indicamos que copiaremos todos nuestros archivos que comiencen con package... y terminen con .json al directorio en donde nos hemos movido.

- *Instrucción RUN npm install*

Run ejecuta un comando, en este caso le estamos indicando que queremos que instale todos las dependencias necesarias para nuestra api en base a nuestros archivos package.json

- *Instrucción COPY . .*

Al igual que en el paso 3 ahora le indicamos que copiaremos todo lo que está dentro de nuestro directorio que contiene el dockerfile a la carpeta actual del contenedor. Es una forma más resumida de hacerlo.

- *Instrucción CMD ["npm", "start"]*

Por último ejecutamos nuestra aplicación con el comando CMD

## WEB

Para la realización de la página web y consumir la API se utilizó javascript y html. Las instrucciones de su archivo dockerfile son las siguientes:

- *Instrucción From php:apache*

Estamos indicando que queremos utilizar la imagen de php con el servidor apache.

- *Instrucción COPY ..web/ /var/www/html*

Aquí copiaremos todo nuestro código a la carpeta var/www/html del contenedor de apache para que nuestra página web funcione correctamente.

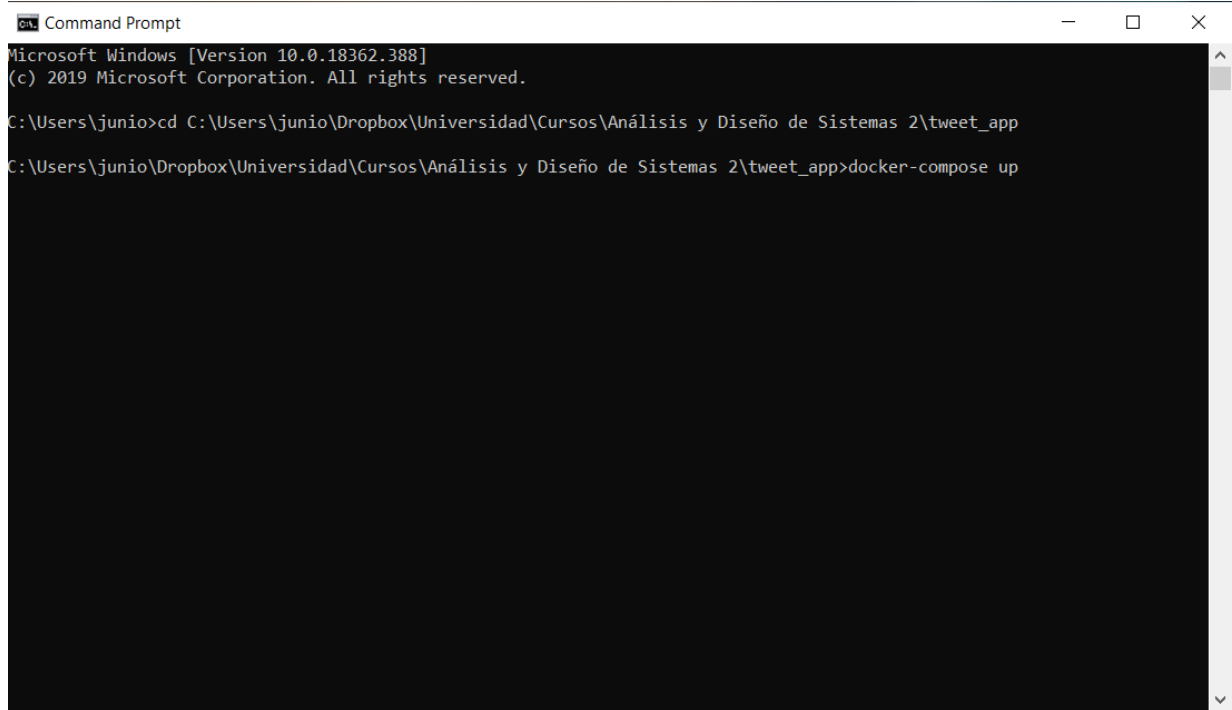
## Archivo DockerCompose

Un archivo Docker-compose nos permite ejecutar varios micro-servicios al mismo tiempo. Primero descarga la imagen que se necesita y luego crea un contenedor, o también, nos permite ejecutar varios archivos dockerfile. Por medio de este archivo hemos ejecutado nuestra aplicación y sus instrucciones son las siguientes:

- **Instrucción versión: "3"**  
Aquí indicamos la versión de dockercompose que vamos a utilizar. Para ver todas las versiones disponibles consultar la página oficial de Docker-hub.
- **Instrucción service**  
Aquí le indicamos que comensaremos a crear los micro-servicios necesarios para la aplicación.
- **Instrucción nombre\_servicio**  
Este nombre es el que se les colocará a nuestros servicios. En nuestro caso hemos utilizado 3, uno para la bd, uno para el api y uno para la web. Cada uno se le coloco el nombre correspondiente.
- **Instrucción build**  
Aquí colocaremos la ruta de nuestros Dockerfile para cada uno de nuestros servicios.
- **Instrucción Ports**  
Aquí colocaremos el puerto que queremos habilitar dentro de nuestra computadora y también el puerto del que está permitiendo la salida nuestro contenedor. De una forma más entendible debe quedar de la siguiente manera:  
Ports: "puerto de nuestra pc":"puerto del contenedor"
- **Instrucción volumes**  
Esta instrucción nos ayuda a persistir nuestra data de la base de datos. Si no usamos volúmenes, cada vez que se levanta el contenedor se pierde la información, entonces es necesario. Su correcta utilización es la siguiente:  
Volumes:
  - "carpeta donde queremos guardar nuestra data":/var/lib/mysqlLa carpeta /var/lib/mysql es la que contiene la data pero dentro del contenedor.
- **Instrucción environment**  
Cuando vamos a utilizar variables de entorno es necesaria esta instrucción, aquí permite definir las variables que queremos utilizar dentro de nuestra aplicación. Para nuestro caso sólo la utilizamos para establecer el host de nuestra bd en la api.
- **Instrucción image**  
Cuando no vamos a utilizar un archivo dockerfile para la construcción de nuestro micro servicio podemos utilizar la imagen directamente con esta instrucción. En nuestro caso la utilizamos para la imagen de apache.
- **Instrucción restart**  
Cuando queremos que nuestro microservicio se reinicie por x razón se utiliza esta instrucción, en nuestro caso queremos que la api se reinicie en caso de fallos, entonces quedaría de la siguiente manera:  
Restart: on-failure

## Ejecutar la aplicación

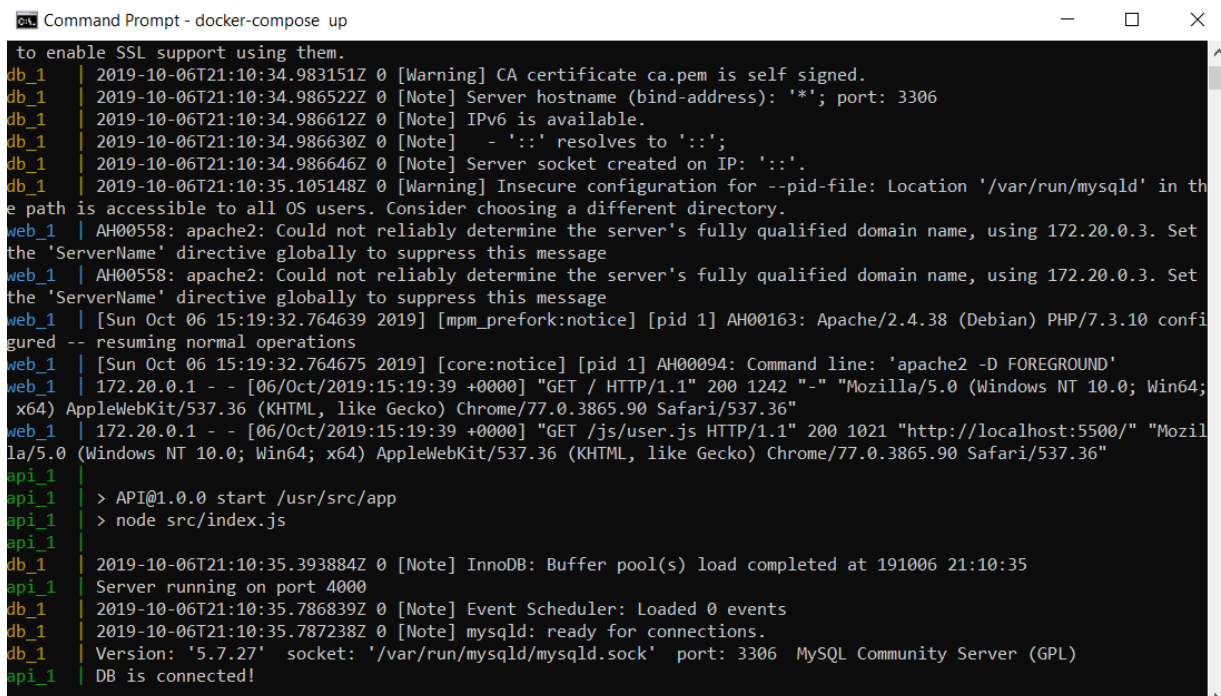
Para ejecutar la aplicación primero nos ubicamos en el directorio que contiene al archivo dockercompose.yml. Luego abrimos desde esa ubicación una consola de comandos y ejecutamos el comando Docker-compose up.



```
Command Prompt
Microsoft Windows [Version 10.0.18362.388]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\junio>cd C:\Users\junio\Dropbox\Universidad\Cursos\Análisis y Diseño de Sistemas 2\tweet_app
C:\Users\junio\Dropbox\Universidad\Cursos\Análisis y Diseño de Sistemas 2\tweet_app>docker-compose up
```

Obtenemos la una respuesta similar a esta



```
Command Prompt - docker-compose up
to enable SSL support using them.
db_1 | 2019-10-06T21:10:34.983151Z 0 [Warning] CA certificate ca.pem is self signed.
db_1 | 2019-10-06T21:10:34.986522Z 0 [Note] Server hostname (bind-address): '*'; port: 3306
db_1 | 2019-10-06T21:10:34.986612Z 0 [Note] IPv6 is available.
db_1 | 2019-10-06T21:10:34.986630Z 0 [Note] - '::' resolves to '::';
db_1 | 2019-10-06T21:10:34.986646Z 0 [Note] Server socket created on IP: '::'.
db_1 | 2019-10-06T21:10:35.105148Z 0 [Warning] Insecure configuration for --pid-file: Location '/var/run/mysqld' in th
e path is accessible to all OS users. Consider choosing a different directory.
web_1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.20.0.3. Set
the 'ServerName' directive globally to suppress this message
web_1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.20.0.3. Set
the 'ServerName' directive globally to suppress this message
web_1 | [Sun Oct 06 15:19:32.764639 2019] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.38 (Debian) PHP/7.3.10 confi
gured -- resuming normal operations
web_1 | [Sun Oct 06 15:19:32.764675 2019] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
web_1 | 172.20.0.1 - - [06/Oct/2019:15:19:39 +0000] "GET / HTTP/1.1" 200 1242 "-" "Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36"
web_1 | 172.20.0.1 - - [06/Oct/2019:15:19:39 +0000] "GET /js/user.js HTTP/1.1" 200 1021 "http://localhost:5500/" "Mozil
la/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36"
api_1 |
api_1 | > API@1.0.0 start /usr/src/app
api_1 | > node src/index.js
api_1 |
db_1 | 2019-10-06T21:10:35.393884Z 0 [Note] InnoDB: Buffer pool(s) load completed at 191006 21:10:35
api_1 | Server running on port 4000
db_1 | 2019-10-06T21:10:35.786839Z 0 [Note] Event Scheduler: Loaded 0 events
db_1 | 2019-10-06T21:10:35.787238Z 0 [Note] mysqld: ready for connections.
db_1 | Version: '5.7.27' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server (GPL)
api_1 | DB is connected!
```



Y podemos entrar a nuestro navegador y colocar localhost:el puerto que decidimos en el archivo dockercompose para nuestra app web. En nuestro caso 5500

