



Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías

Ejemplo básico utilizando Docker

Alumno: Bryan De Anda Reyes

Código de alumno: 216195537

Materia: Computación Tolerante a Fallas (D06)

Horario: lunes y miércoles (11:00 – 13:00)

Carrera: INCO

Profesor: Michel Emanuel López Franco

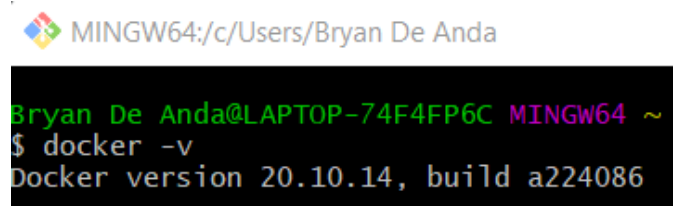
Instrucciones

El objetivo de esta actividad es realizar un ejemplo básico para poner en práctica la utilización de la herramienta Docker. Docker es un proyecto de código abierto 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.

Desarrollo

Ejemplo básico utilizando Docker

Para el desarrollo de esta actividad, primero fue necesario realizar la instalación de Docker desde su sitio oficial, y para verificar que se instaló correctamente, desde el PowerShell se escribe la línea de comandos “docker -v” la cual nos mostrara la versión de Docker instalada.



```
MINGW64:/c/Users/Bryan De Anda  
Bryan De Anda@LAPTOP-74F4FP6C MINGW64 ~  
$ docker -v  
Docker version 20.10.14, build a224086
```

Ahora bien, para que Docker funcione correctamente también es necesario instalar el subsistema de Windows para Linux (WSL) en su versión 2, si todo esto se hizo correctamente, en la aplicación de Docker de nuestro computador debe abrirse sin ningún inconveniente.

Hablando a grandes rasgos de lo que hace el ejemplo, creará una aplicación Web en Python “Hola mundo” utilizando Flask framework y lo que va a hacer este contenedor es empaquetar la aplicación con todas las partes necesarias, por lo que, todas las bibliotecas y demás dependencias también se empaquetaran.

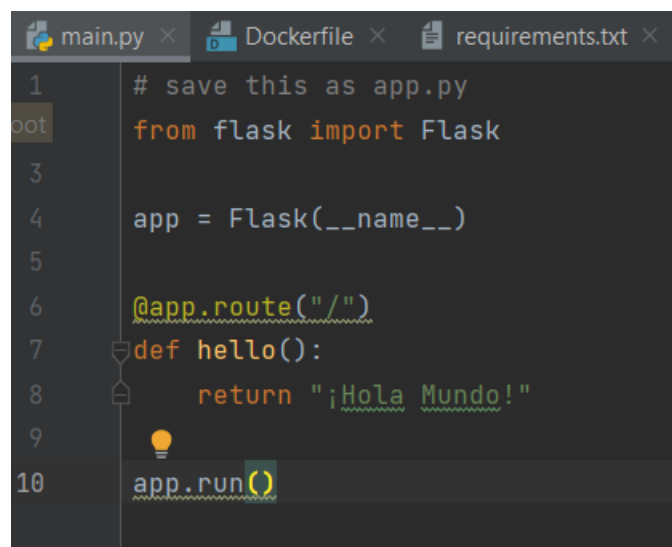
El primer paso para nuestra aplicación es generar un archivo Docker que tiene que ser llamado “Dockerfile”, y para generar el contenedor se necesitan tres cosas indispensables, la primera es el Dockerfile que ya se mencionó, la segunda es una imagen Docker y por último el contenedor en sí.

Lo que hace el Dockerfile es generar imágenes Docker, la imagen es una plantilla para ejecutar contenedores y el contenedor es el proceso en ejecución real donde tenemos nuestro proyecto empaquetado.

También, creamos un archivo llamado "requirements.txt" el cual contendrá las dependencias de nuestro archivo Python y también creamos el "main.py" que contendrá el código fuente.

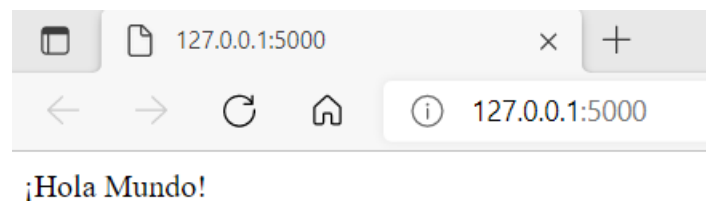
La actividad pide la realización de un ejemplo básico y el objetivo principal de esta actividad es aprender a utilizar Docker. Por esta razón el código de la página Web es un hola mundo utilizando el framework Flask de python, para instalarlo se ingresa la siguiente instrucción en la terminal de python, "pip install Flask".

En la siguiente captura se muestra el código que generará el sitio web con el hola mundo.



```
1 # save this as app.py
2 from flask import Flask
3
4 app = Flask(__name__)
5
6 @app.route("/")
7 def hello():
8     return "¡Hola Mundo!"
9
10 app.run()
```

Si corremos este código nos genera el sitio en el localhost con la información correspondiente definida en el código.



Ya que tenemos todo lo anterior bien definido, procedemos a crear la imagen Docker con la instrucción “docker build -t app-flask .” esta imagen nos permitirá generar el contenedor de nuestro API para después poder ejecutarlo.

```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject4> docker build -t app-flask .
[+] Building 2.1s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 32B                                                0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.8.5-alpine3.11        1.7s
=> [internal] load build context                                                  0.1s
=> => transferring context: 78.36kB                                              0.1s
```

Utilizando el comando “docker images”, podemos darnos cuenta de todas las imágenes que se crearon y ahí podemos encontrar la que creamos con el nombre “app-flask” mostrándonos diversa información, como su ID, su tamaño, etc.

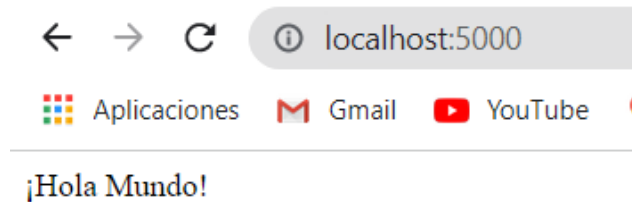
```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject4> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
app-flask	latest	c6d5b2cbfddd	About a minute ago	67.1MB
nginx_imagen1	latest	fe2549f5bb04	5 hours ago	58.1MB

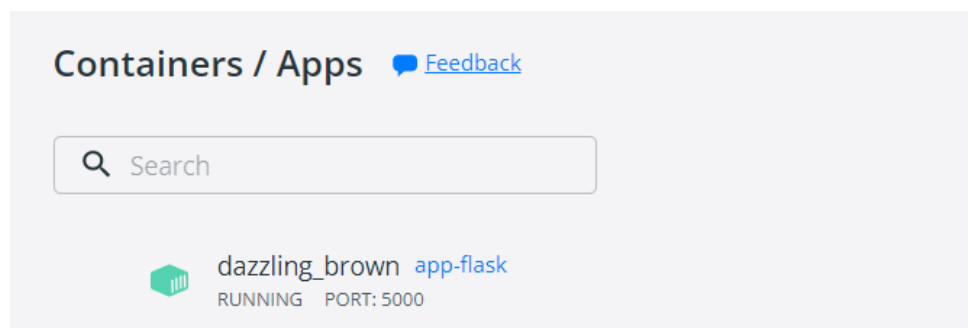
Después de la creación de la imagen que se le asigno el nombre de “app-flask” se ejecuta el siguiente comando en la terminal “docker run -it -p 5000:5000 NAME_IMAGE”, indicando los puertos adecuados.

```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject4> docker run -it -p 5000:5000 ap
p-flask
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000 (Press CTRL+C to quit)
```

Si intentamos ingresar en un navegador con la dirección “localhost:5000”, el Hola Mundo que escribimos en Python, con esto podemos comprobar que el contenedor se esa ejecutando correctamente.



También podemos verificar la ejecución en la aplicación de Docker, ahí encontraremos todos los contenedores y en este caso aparece debajo del contenedor con el nombre de la imagen que creamos el mensaje de “RUNNING”.



Ahora, si entramos al contenedor podemos apreciar diversa información, como por ejemplo los tiempos en los que estuvo en ejecución



Otra manera para comprobar la ejecución es desde la terminal si escribimos el comando, “docker ps -a” en estatus podemos ver que el contenedor esta arriba, es decir, está corriendo.

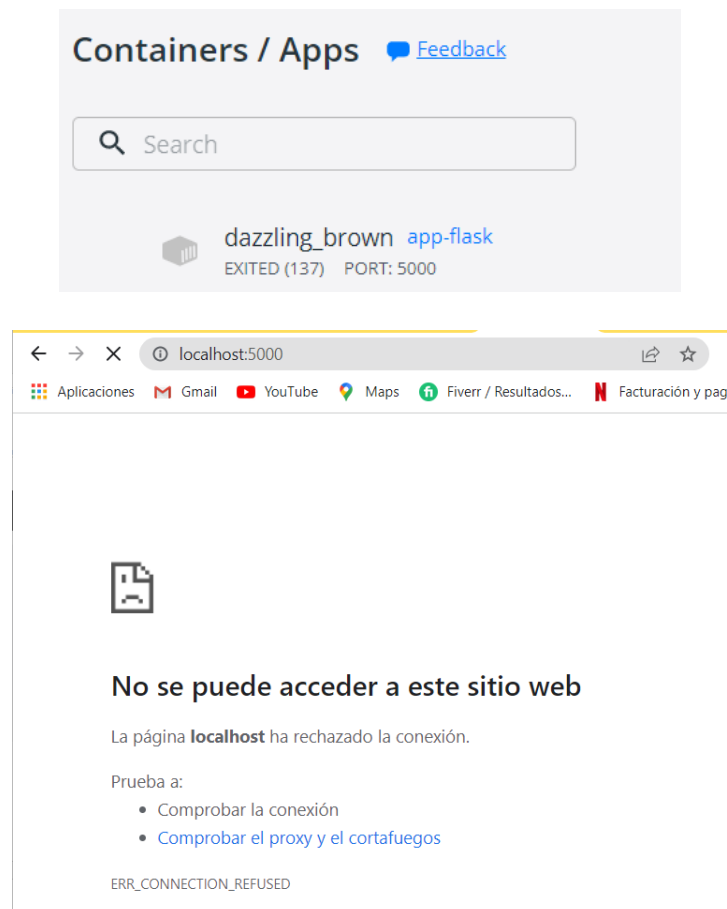
```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject4> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f81802323a6c	app-flask	"/bin/sh -c 'python ..."	2 minutes ago	Up 2 minutes	0.0.0.0:5000->5000/tcp	dazzling_brown
d714d0be38e	apache/singflow-2.2.5	"/usr/bin/dumb-init ..."	3 days ago	Exited (137) About a minute ago		singflow-deckn...

Después, si queremos detener la ejecución podemos hacerlo desde la consola escribiendo el siguiente comando “Docker stop CONTAINER_ID”, o bien podemos hacerlo desde la aplicación de Docker presionando el botón con el cuadrado negro.

```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject4> docker stop f81802323a6c
f81802323a6c
```

Desde la aplicación podemos ver que el contenedor ya no está corriendo, mostrando el mensaje “EXITED”, o también podemos ingresar al sitio web con la dirección localhost:5000 comprobando que ya no se puede acceder a ese sitio.



Enlace al código en el repositorio: <https://github.com/BryanDeAnda/Ejemplo-basico-utilizando-Docker..git>

Conclusión

La realización de esta actividad me pareció muy interesante de realizar, ya que implementé nuevas herramientas que no había utilizado anteriormente, como lo fue crear una API con Python utilizando la librería Flask.

En cuanto a Docker se me presentaron más problemas al momento de su instalación porque funciona con un subsistema de Windows para Linux, pero en cuanto al manejo de Docker no me costó tanto trabajo al ya haber trabajado con el en la practica pasada y en la práctica de Airflow.

Bibliografía

C. (2022, 8 abril). *Pasos de instalación manual para versiones anteriores de WSL*. Microsoft Docs. Recuperado 2022, de <https://docs.microsoft.com/es-mx/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>

Flask. (2022, 30 marzo). PyPI. Recuperado 2022, de <https://pypi.org/project/Flask/>