Ejemplo básico utilizando Docker

Nombre: Diana Fernanda Castillo Rebolledo

Materia: Computación Tolerante a Fallas

Horario: L-I 11:00 am-1:00 pm

Profesor: Dr. Michel Emanuel López Franco

Sección: D06

¿Qué es Docker?

Docker es una plataforma de software que permite crear y ejecutar aplicaciones de una manera rápida. Esta plataforma empaqueta software en unidades que son llamadas contenedores, estos incluyen todo lo necesario para que funcione la aplicación, como bibliotecas y otras herramientas. De esta manera se pueden implementar las aplicaciones en cualquier entorno con la certeza de que estas funcionarán al momento de su ejecución.

Dentro del contenedor se definen una serie de instrucciones (dockerfile) que permiten crear una imagen con la que poder arrancar a dicho contenedor.

- *Dockerfile* es el documento de texto sobre el que se pueden agrupar una serie de comandos con el fin de que se ejecuten todos a la vez evitando así tener que ejecutarlos uno a uno manualmente con el fin de que el proceso de crear una imagen de Docker sea mucho más rápido y eficiente.
- Una *imagen de Docker* contiene las librerías, junto al código de la aplicación que contiene todo lo necesario para ejecutar nuestra aplicación.
- Un *contenedor* es una imagen de Docker cuando empieza a funcionar, es decir, cuando cobra vida.

Una vez definido lo anterior, se explicará un ejemplo utilizando Docker.

Primeramente, para su instalación se necesita actualizar WSL (Windows Subsystem for Linux) a WSL 2 en la distribución de Linux que se tenga, en mi caso Ubuntu20.04LTS:

```
wsl --set-version Ubuntu20.04LTS 2
```

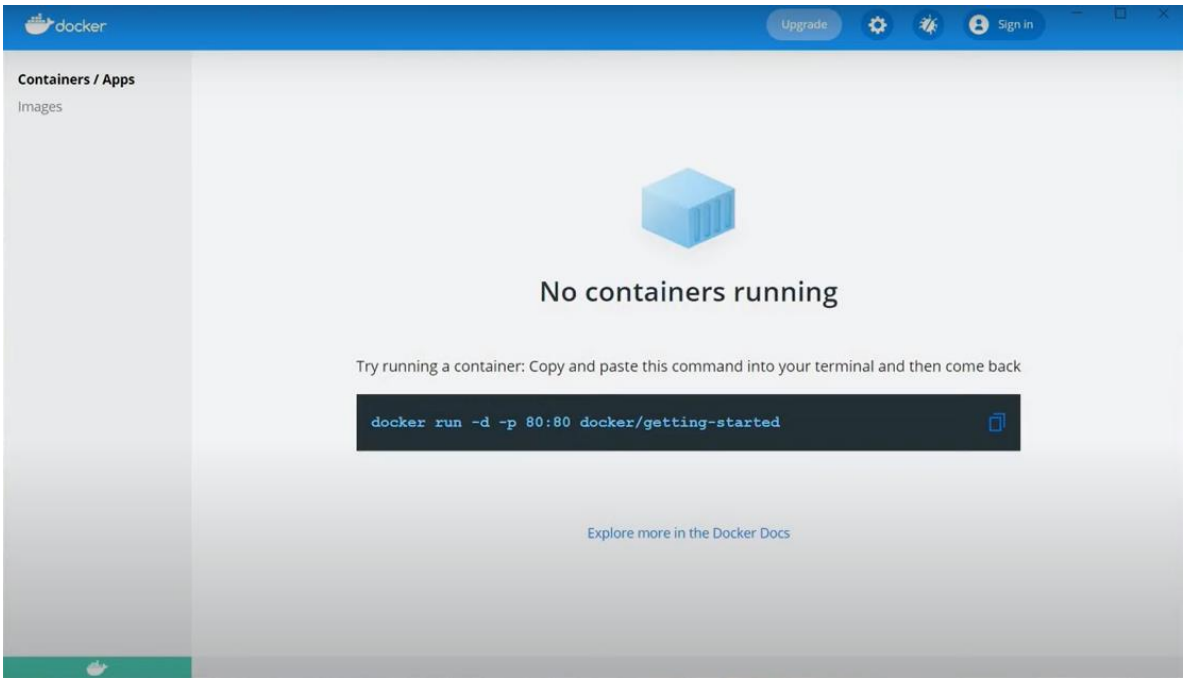
Después colocar la versión 2 de WSL por defecto en el sistema.

```
wsl --set-default-version 2
```

Una vez hecho esto se debe colocar por defecto Ubuntu como la distribución por defecto en docker desktop en la cual se van a ejecutar los contenedores.

```
wsl --set-default Ubuntu20.04LTS
```

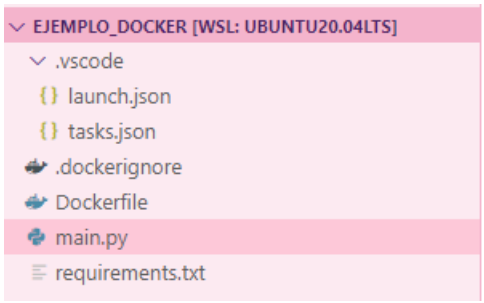
Luego de terminar esto, ya se puede descargar e instalar Docker desde su página oficial, en mi caso para Windows, y ya se podrá ejecutar.



Yo utilicé Docker para ejecutar un programa en Python en Visual Studio Code, así que instalé dos extensiones que hicieron el proceso más sencillo, una es “Docker” que ayuda en la creación y gestión de aplicaciones en contenedores, la otra extensión es “Remote - WSL” la cual permite abrir carpetas en WSL y conectarse de forma remota al WSL en VS Code tal y como funcionaría al trabajar en Windows. Una vez hecho esto, abrí una nueva ventana de VS Code conectada a WSL2 y cree una carpeta en la que generé mi aplicación.

El programa que generé es un flujo de prefect que se ejecuta cada dos minutos con un proceso ETL que extrae datos del link “https://jsonplaceholder.cypress.io/todos” el cual simula información sobre varios usuarios, esto en una lista de objetos json, después en transform por cada objeto dentro de la lista, se crea un diccionario usando los valores de cada objeto, pero modificando las keys, y cada uno de estos nuevos diccionarios se agregan a una lista nueva y esta se retorna como un DataFrame con ayuda de Pandas para posteriormente guardar ese DataFrame en un archivo.csv.

Después de generar este código, añadí archivos de Docker al espacio de trabajo con ayuda de la extensión de Docker, la cual tiene esta opción para agregar los archivos que usualmente se necesitan para ejecutar un contenedor.



El archivo Dockerfile es el que contiene todas las instrucciones sobre cómo generar la imagen de Docker. En el archivo requirements.txt que se encuentra vacío, es donde se colocan los nombres de los paquetes de bibliotecas u otras extensiones en caso de que estas se necesiten para poder ejecutar la aplicación, es decir, herramientas que normalmente se instalarían con pip en caso de ser para Python como en este programa, y con esto se instalarán estos paquetes automáticamente al generar la imagen de Docker. Lo que se busca con esto es que siempre se pueda ejecutar la aplicación sin importar el entorno en el que se encuentre el contenedor. Para esta aplicación necesito la biblioteca de Prefect y Pandas:

main.py

```python
import json
import requests
import pandas
import datetime
from prefect import task, Flow, Parameter
from prefect.schedules import IntervalSchedule

@task(max_retries=10, retry_delay=datetime.timedelta(seconds=10))
def extract(url):
    resultado = requests.get(url)
    if not resultado:
        raise Exception('Datos no obtenidos.')
    return json.loads(resultado.content)

@task
def transform(datos):
    usuarios = []
    for usuario in datos:
        usuarios.append({
            'UsuarioID': usuario['userId'],
            'ID': usuario['id'],
            'Titulo': usuario['title'],
            'Completado': usuario['completed']
        })
    return pandas.DataFrame(usuarios)

@task
def load(datos, path):
    datos.to_csv(path, index=False)

schedule = IntervalSchedule(interval=datetime.timedelta(minutes=2))
```

Por lo que los coloqué en requirements.txt:

main.py | requirements.txt | Dockerfile

requirements.txt

```
prefect
pandas
```

En el archivo de Dockerfile venían varios comandos por defecto, la primera línea es la que permite escribir código básico en Python, la siguiente línea es para ignorar los archivos.pyc (archivos que contienen el código de bytes compilado para un programa en Python) para que estos no se estén generando en el contenedor, el tercer comando es para desactivar el almacenamiento en búfer para facilitar el registro de contenedores, las siguientes dos líneas son para instalar los paquetes de pip, ahí es donde se utiliza el archivo de requirements.txt donde se especificaron los paquetes necesarios. Las siguientes dos líneas configuran un directorio de trabajo y copian todos los archivos que hay en la carpeta al nuevo directorio de aplicaciones (imágenes) de docker. Después se agrega un nuevo usuario sin contraseña y con algunos privilegios limitados que solo son necesarios para ejecutar la aplicación por cuestiones de seguridad, y el último comando ejecutará el comando Python con el archivo main.py que es donde se encuentra el programa.

Dockerfile > ...

```
1   # For more information, please refer to https://aka.ms/vscode-docker-python
2   FROM python:3.8-slim
3
4   # Keeps Python from generating .pyc files in the container
5   ENV PYTHONDONTWRITEBYTECODE=1
6
7   # Turns off buffering for easier container logging
8   ENV PYTHONUNBUFFERED=1
9
10  # Install pip requirements
11  COPY requirements.txt .
12  RUN python -m pip install -r requirements.txt
13
14  WORKDIR /app
15  COPY . /app
16
17  # Creates a non-root user with an explicit UID and adds permission to access the /app folder
18  # For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
19  RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
20  USER appuser
21
22  # During debugging, this entry point will be overridden. For more information, please refer t
23  CMD ["python", "main.py"]
```

Después se tiene que construir la imagen del Dockfile con el comando docker build:

```
TERMINAL   PROBLEMS   OUTPUT   DEBUG CONSOLE

> Executing task: docker-build <

> docker build --rm --pull -f "/home/diana/ejemplo_docker/Dockerfile" --label "com.microsoft.created-by=visual-studio-
code" -t "ejemplodocker:latest" "/home/diana/ejemplo_docker" <

#2 [internal] load .dockerignore
#2 sha256:25423383493ea010d2b3b82bc1b730c0c06081d9428dbd2f428b91b028a69ec6
#2 transferring context:
#2 transferring context: 35B 0.5s done
#2 DONE 2.5s

#1 [internal] load build definition from Dockerfile
#1 sha256:ced736df5eda6ad91882f842f5449cabe6b9acd23ed550d664b76f617ed7dd1e
#1 transferring dockerfile: 38B 0.5s done
#1 DONE 4.0s

#3 [internal] load metadata for docker.io/library/python:3.8-slim
#3 sha256:a94c831f6b826f8ffd4325edd7e9f6408caa85fd7f03e2b107cc6a9d249f7065
#3 DONE 29.0s

#5 [internal] load build context
#5 sha256:36d07234e7345868f783021762ae1200daeaac8f08952367a51bb76bb39e5692
#5 DONE 0.0s

#4 [1/6] FROM docker.io/library/python:3.8-slim@sha256:09919e06ecceca5cca99ab3622758b11f52abe7dcfc9bdb945d6bfe7482a0e8
2
#4 sha256:06cf7d837b5ae0bbf303a0147b8a9893dd15a19f8b4c710861aa2c6dbf0c9019
#4 DONE 0.1s

#5 [internal] load build context
#5 sha256:36d07234e7345868f783021762ae1200daeaac8f08952367a51bb76bb39e5692
#5 transferring context:
#5 transferring context: 64B 0.2s done
```

Después con el comando “docker image list” se puede ver una lista de las imágenes que hay en Docker, y está la imagen recién creada con el nombre de la carpeta en la que se está trabajando:

```
diana@user-PC:~/ejemplo_docker$ docker image list
REPOSITORY                TAG       IMAGE ID       CREATED        SIZE
ejemplodocker             latest    57c08ce17264   5 hours ago    338MB
docker/getting-started    latest    bd9a9f733898   5 weeks ago    28.8MB
```

Y se puede ejecutar con “docker run”:

```
diana@user-PC:~/ejemplo_docker$ docker run ejemplodocker
[2022-03-20 07:33:07+0000] INFO - prefect.etl_flow | Waiting for next scheduled run at 2022-03-20T07:34:00+00:00
[2022-03-20 07:34:00+0000] INFO - prefect.FlowRunner | Beginning Flow run for 'etl_flow'
[2022-03-20 07:34:00+0000] INFO - prefect.TaskRunner | Task 'parametro_url': Starting task run...
[2022-03-20 07:34:03+0000] INFO - prefect.TaskRunner | Task 'parametro_url': Finished task run for task with final state: 'Success'
[2022-03-20 07:34:03+0000] INFO - prefect.TaskRunner | Task 'extract': Starting task run...
[2022-03-20 07:34:07+0000] INFO - prefect.TaskRunner | Task 'extract': Finished task run for task with final state: 'Success'
[2022-03-20 07:34:07+0000] INFO - prefect.TaskRunner | Task 'transform': Starting task run...
[2022-03-20 07:34:08+0000] INFO - prefect.TaskRunner | Task 'transform': Finished task run for task with final state: 'Success'
[2022-03-20 07:34:08+0000] INFO - prefect.TaskRunner | Task 'load': Starting task run...
[2022-03-20 07:34:08+0000] INFO - prefect.TaskRunner | Task 'load': Finished task run for task with final state: 'Success'
[2022-03-20 07:34:08+0000] INFO - prefect.FlowRunner | Flow run SUCCESS: all reference tasks succeeded
[2022-03-20 07:34:08+0000] INFO - prefect.etl_flow | Waiting for next scheduled run at 2022-03-20T07:36:00+00:00
```

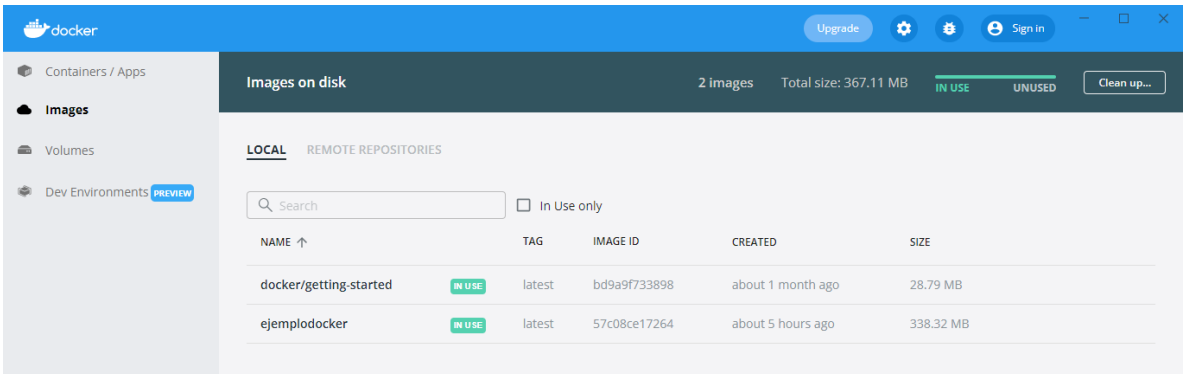
Archivo.csv creado:

main.py | requirements.txt | Dockerfile | datos.csv

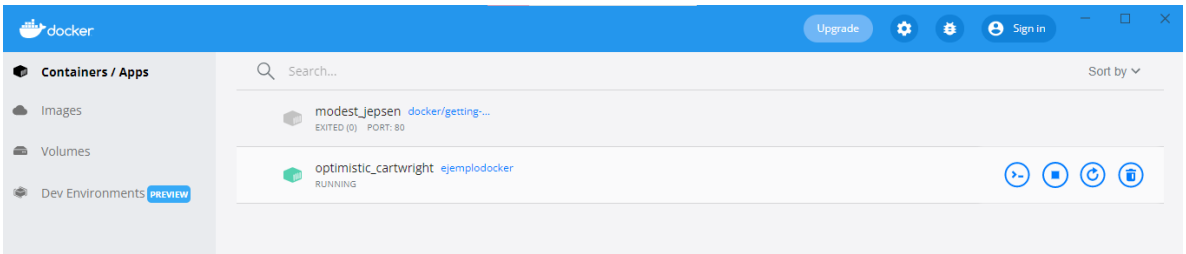
docker-desktop-data > version-pack-data > community > docker > overlay2 > 99cc8445f6d8d38a01000bfbbec9c

```
UsuarioID,ID,Titulo,Completado
1,1,delectus aut autem,False
1,2,quis ut nam facilis et officia qui,False
1,3,fugiat veniam minus,False
1,4,et porro tempora,True
1,5,laboriosam mollitia et enim quasi adipisci quia provident illum,False
1,6,qui ullam ratione quibusdam voluptatem quia omnis,False
1,7,illo expedita consequatur quia in,False
1,8,quo adipisci enim quam ut ab,True
1,9,molestiae perspiciatis ipsa,False
1,10,illo est ratione doloremque quia maiores aut,True
1,11,vero rerum temporibus dolor,True
1,12,ipsa repellendus fugit nisi,True
1,13,et doloremque nulla,False
1,14,repellendus sunt dolores architecto voluptatum,True
1,15,ab voluptatum amet voluptas,True
1,16,accusamus eos facilis sint et aut voluptatem,True
1,17,quo laboriosam deleniti aut qui,True
1,18,dolorum est consequatur ea mollitia in culpa,False
1,19,molestiae ipsa aut voluptatibus pariatur dolor nihil,True
1,20,ullam nobis libero sapiente ad optio sint,True
2,21,suscipit repellat esse quibusdam voluptatem incidunt,False
2,22,distinctio vitae autem nihil ut molestias quo,True
2,23,et itaque necessitatibus maxime molestiae qui quas velit,False
2,24,adipisci non ad dicta qui amet quaerat doloribus ea,False
2,25,voluptas quo tenetur perspiciatis explicabo natus,True
2,26,aliquam aut quasi,True
2,27,veritatis pariatur delectus,True
2,28,nesciunt totam sit blanditiis sit,False
2,29,laborum aut in quam,False
2,30,nemo perspiciatis repellat ut dolor libero commodi blanditiis omnis,True
2,31,repudiandae totam in est sint facere fuga,False
2,32,earum doloribus ea doloremque quis,False
```

De igual manera en la interfaz de Docker se pueden ver las imágenes que hay:



Así como los contenedores, los cuales se pueden pausar, ejecutar, o eliminar:



Código en GitHub: https://github.com/diana-castillo/Docker.git

Conclusiones:

Docker me pareció una herramienta bastante útil y práctica para poder generar y ejecutar aplicaciones, en especial cuando nos encontramos en un entorno en el cual no se tiene la certeza de que estén instalados todos los paquetes de herramientas extras que se necesitan para poder hacer funcionar una determinada aplicación o si es que esta se quiere ejecutar rápidamente en un entorno distinto al que se creó, pues las imágenes de docker son sencillas de construir y más con la ayuda de las extensiones para Docker que hay en Visual Studio Code, los contenedores resultan muy cómodos ya que sólo contienen lo que la aplicación necesita, y la interfaz de Docker es muy útil para verificar qué imágenes y contenedores se tienen y poder ejecutarlos desde ahí en caso de que se requiera.

Referencias:

Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. Recuperado 18 de marzo de 2022, de https://aws.amazon.com/es/docker/

Bernal González, D. (2021, 2 agosto). *¿Qué es Docker y para qué sirve?* Profile Software Services. Recuperado 19 de marzo de 2022, de https://profile.es/blog/que_es_docker/

Tech Journey. (2021, 4 septiembre). *Aprende a instalar Docker en Windows* [Vídeo]. YouTube. https://www.youtube.com/watch?v=9awV3Y-rpI0

The Digital Life. (2021, 10 mayo). *Docker VSCode Python Tutorial ▶ Run your App in a Container* [Vídeo]. YouTube. https://www.youtube.com/watch?v=jtBVppyfDbE