

# Ejemplo básico utilizando Docker

---

**Alumno:** Oswaldo Gael Cervantes Castoño

**Código:** 219747468

**Computación Tolerante a Fallas**

**Sección D06**

**Profesor:** Michel Emanuel López Franco



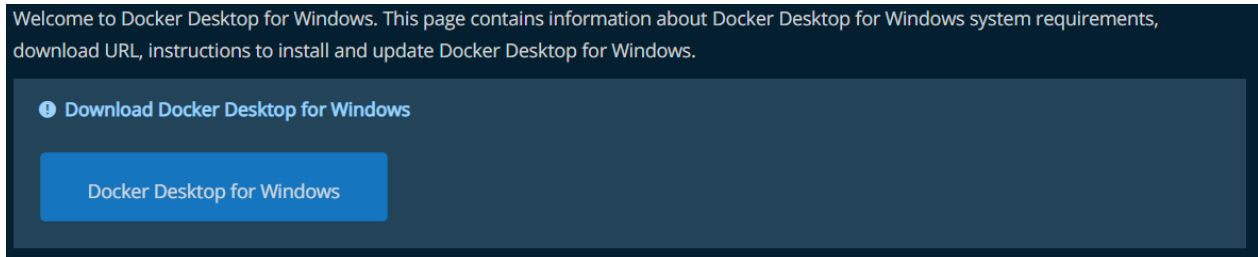
Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

## Objetivo:

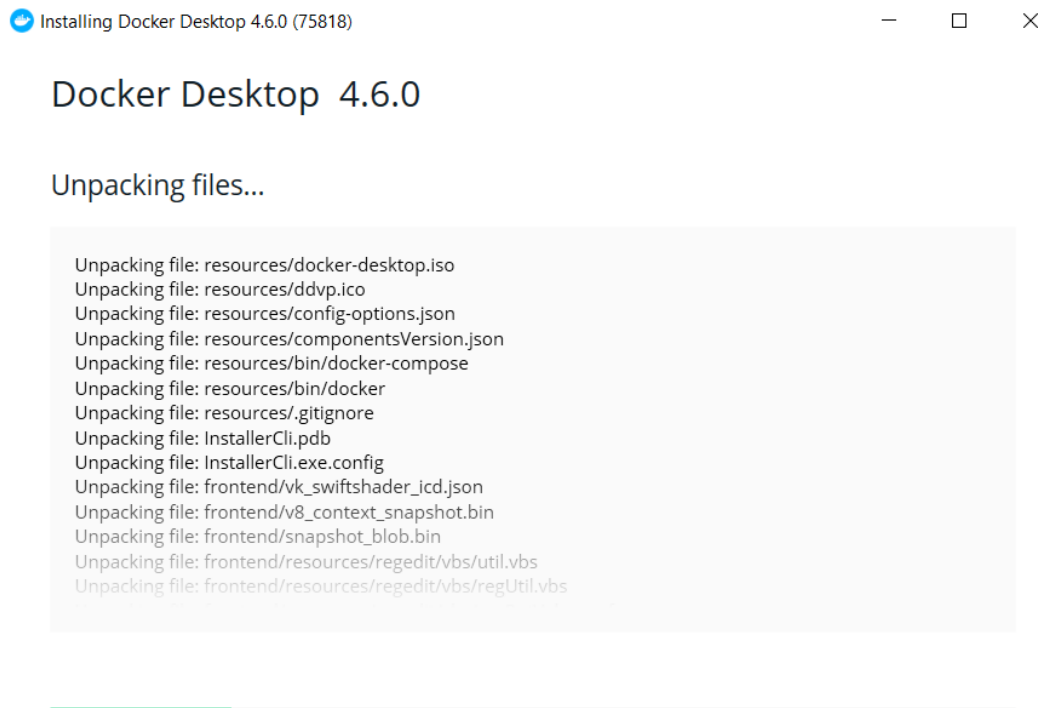
Genera un programa en el cual utilices Docker.

## Desarrollo:

El primer paso para el desarrollo de esta actividad es descargar Docker para escritorio, el cual podemos encontrarlo en: <https://docs.docker.com/desktop/windows/install/>:



Una vez que ejecutamos el archivo descargado esperamos a que se descompriman los archivos:



Esperamos a que se instalen los paquetes:

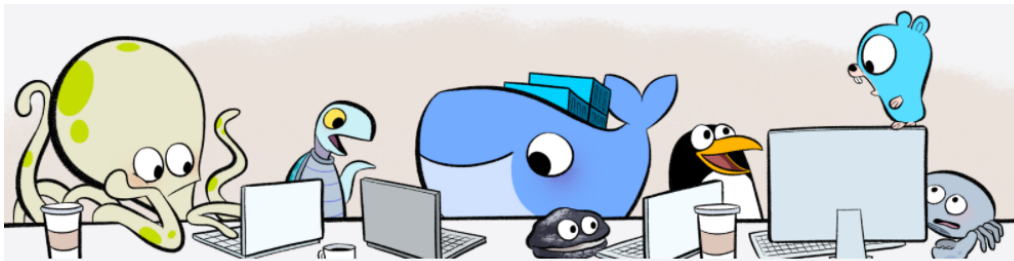
Installing Docker Desktop 4.6.0 (75818)

## Docker Desktop 4.6.0

Installing...

```
Deploying component: Install required Windows components for WSL 2
Deploying component: Add user to docker-users group
Deploying component: Create docker-users group
Installing components
Unpacking file: System.Xml.XPath.XDocument.dll
Unpacking file: System.Xml.XPath.dll
Unpacking file: System.Xml.XmlSerializer.dll
Unpacking file: System.Xml.XmlDocument.dll
Unpacking file: System.Xml.XDocument.dll
Unpacking file: System.Xml.ReaderWriter.dll
Unpacking file: System.Web.Http.Owin.dll
Unpacking file: System.Web.Http.dll
Unpacking file: System.ValueTuple.dll
Unpacking file: System.Threading.Timer.dll
```

Aceptamos los términos y condiciones:




### Our Service Agreement has Changed

We've updated the [Docker Subscription Service Agreement](#). Please read the [Blog](#) and [FAQs](#) to learn how companies using Docker Desktop may be affected. By checking "I accept the terms" you agree to the [Subscription Service Agreement](#), the [Data Processing Agreement](#), and the [Data Privacy Policy](#).

Here's a summary of key changes:

- Our Docker Subscription Service Agreement include a change to the terms of use for Docker Desktop.
  - It **remains free** for small businesses (fewer than 250 employees AND less than \$10 million in annual revenue), personal use, education, and non-commercial open source projects.
  - It requires a paid subscription for professional use in larger enterprises.
- The effective date of these terms is August 31, 2021. There was a **grace period** until January 31, 2022 for those that require a paid subscription to use Docker Desktop. Docker trusts our customers to be in compliance and Docker Desktop will continue to function normally after January 31st, but this is a

I accept the terms ☒

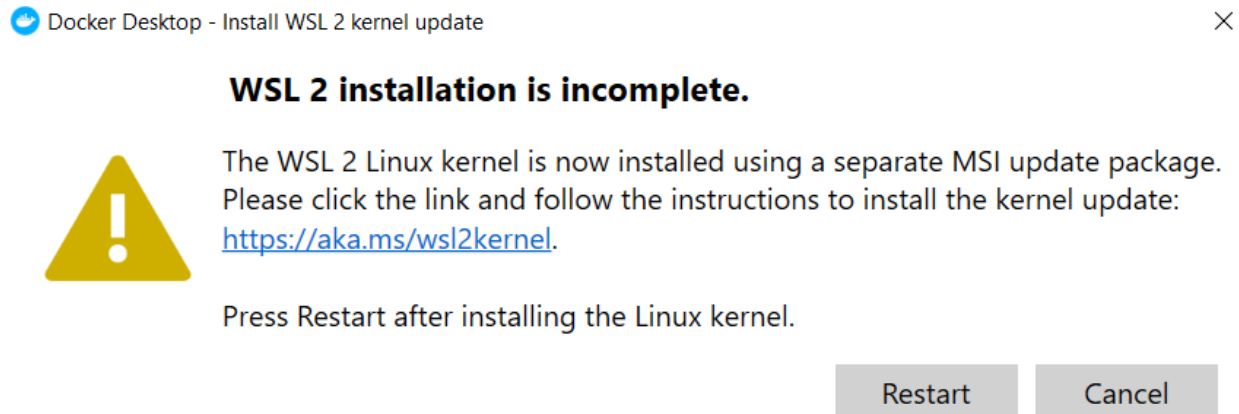
[View Full Terms](#) 

[Decline and Close Application](#) [Accept](#)

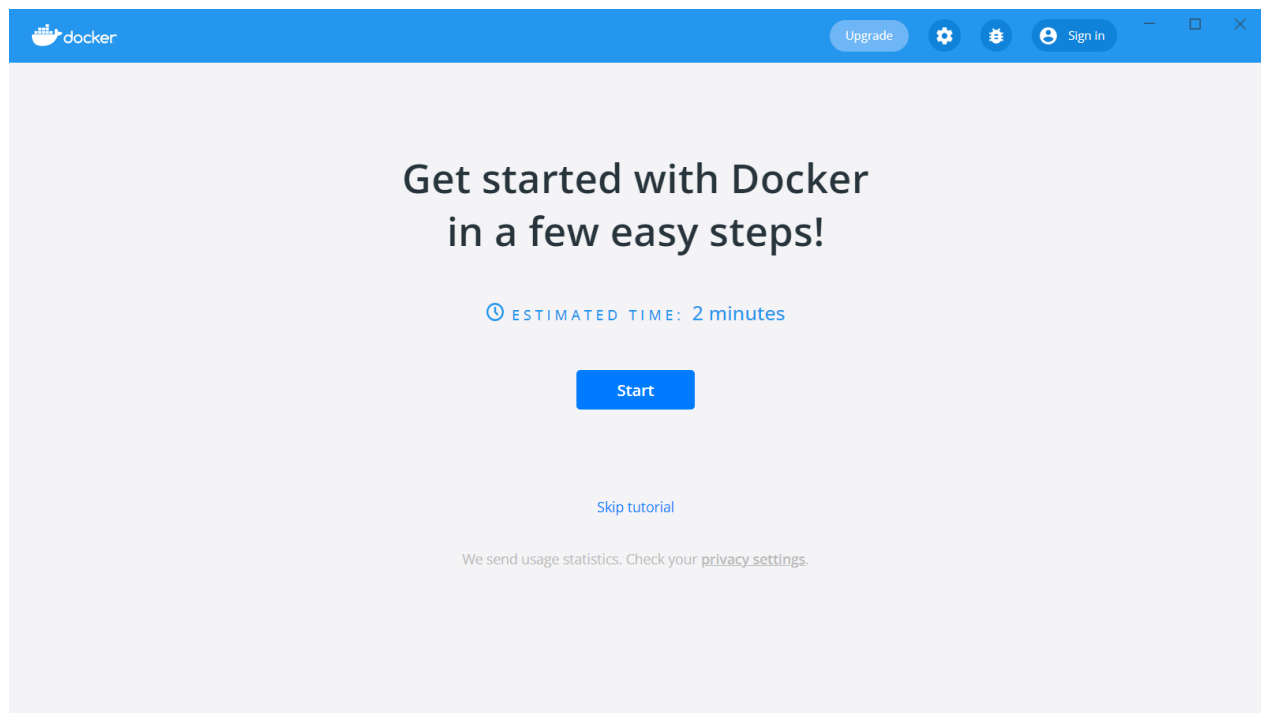
Para iniciar Docker se nos solicita reiniciar la computadora, así que procedemos y lo que vemos a continuación es una ventana como la de la siguiente imagen.

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Docker nos lanza una advertencia de que no se han instalado todos los componentes necesarios para utilizar Windows Subsystem for Linux, así que ingresamos al enlace sugerido:

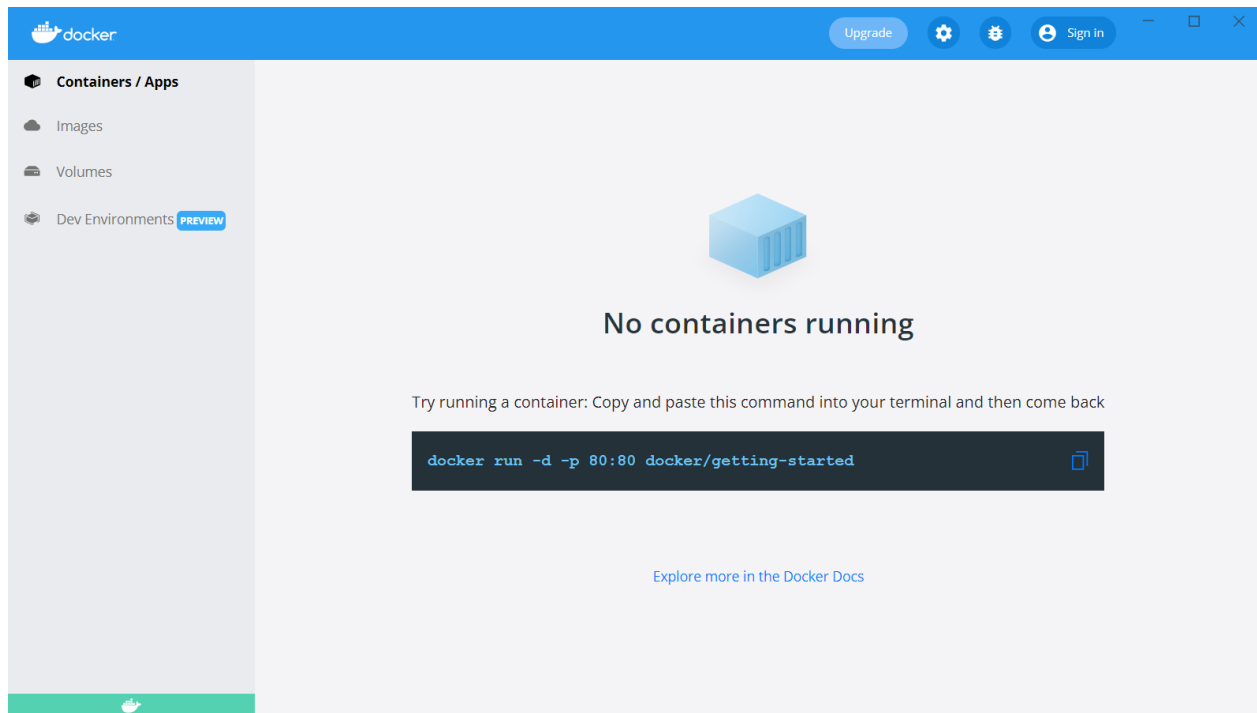


Una vez que descargamos e instalamos el paquete faltante, presionamos la opción “Restart” de la imagen anterior y con ello ya estamos listos para utilizar Docker. Se nos muestra entonces una imagen como la siguiente, pero seleccionamos la opción de saltar el tutorial para ir directamente a lo que nos concierne:



Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Lo que vemos a continuación es la ventana principal de Docker para escritorio, así que lo siguiente es trabajar sobre la configuración inicial de nuestro entorno de trabajo:



Para corroborar que la instalación se llevó a cabo con éxito, podemos ejecutar el comando “docker --version” y si la información desplegada en la terminal es parecida a la de la siguiente imagen, significa que logramos instalar Docker para escritorio de forma correcta:

```

C:\> Símbolo del sistema
Microsoft Windows [Versión 10.0.19044.1586]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gael>docker --version
Docker version 20.10.13, build a224086

C:\Users\Gael>
```

Ya que vamos a trabajar con un proyecto que pretendemos trabajar en computadoras con entornos de ejecución diferentes al nuestro, para comenzar a trabajar sobre el código primero tenemos que iniciar un entorno virtual, para lo cual tenemos que instalar por medio de la terminal el paquete “virtualenv” haciendo uso del comando “pip install virtualenv”:

```
Microsoft Windows [Versión 10.0.19044.1586]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\python-flask-docker>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.13.4-py2.py3-none-any.whl (8.7 MB)
    | 8.7 MB 328 kB/s
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.4-py2.py3-none-any.whl (461 kB)
    | 461 kB 297 kB/s
Collecting filelock<4,>=3.2
  Downloading filelock-3.6.0-py3-none-any.whl (10.0 kB)
Collecting platformdirs<3,>=2
  Downloading platformdirs-2.5.1-py3-none-any.whl (14 kB)
Requirement already satisfied: six<2,>=1.9.0 in c:\users\gael\appdata\local\programs\python\python310\lib\site-packages (from virtualenv) (1.16.0)
Installing collected packages: platformdirs, filelock, distlib, virtualenv
Successfully installed distlib-0.3.4 filelock-3.6.0 platformdirs-2.5.1 virtualenv-20.13.4
WARNING: You are using pip version 21.2.4; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\Gael\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.

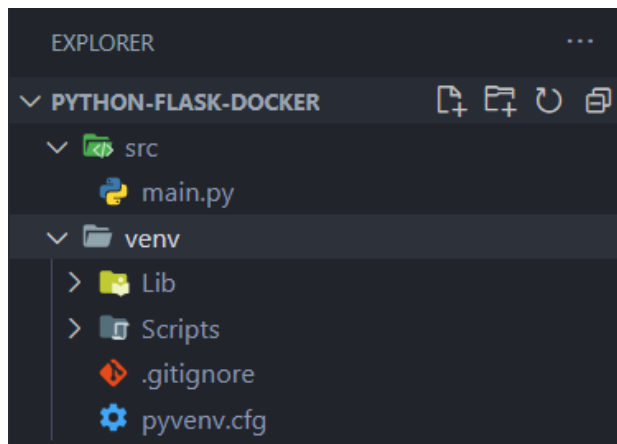
D:\python-flask-docker>
```

Lo siguiente es ejecutar el comando “virtualenv venv”, donde “venv” es el nombre del directorio que se va a crear para colocar todos los archivos necesarios para el entorno virtual:

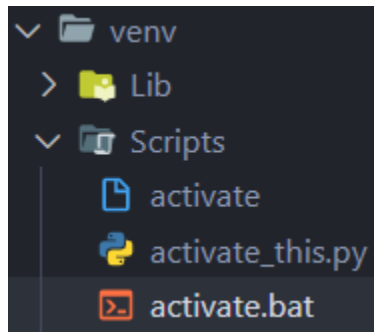
```
D:\python-flask-docker>virtualenv venv
created virtual environment CPython3.10.2.final.0-64 in 12251ms
  creator CPython3Windows(dest=D:\python-flask-docker\venv, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\Gael\AppData\Local\pypa\virtualenv)
    added seed packages: pip==22.0.4, setuptools==60.10.0, wheel==0.37.1
    activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator

D:\python-flask-docker>
```

Podemos observar que se creó la carpeta “venv” con algunos recursos y subdirectorios que se añadieron de forma automática. La carpeta “src” la creé yo para crear los archivos necesarios para mi RESTful API con Flask y Python:



Dentro de la carpeta “Scripts” en “venv/Scripts”, podemos encontrar un archivo llamado “actíivate.bat”, y el último paso previo a comenzar con la codificación de la API es ejecutar este archivo:



Ingresamos a la ruta mencionada en el paso anterior por medio de la terminal y ejecutamos el archivo escribiendo su nombre y extensión. Podemos observar que una vez que se ejecuta aparece un prefijo en el nombre de la ruta actual en la terminal, lo cual significa que ya estamos trabajando sobre nuestro entorno virtual:

```
D:\python-flask-docker>cd venv/Scripts  
D:\python-flask-docker\venv\Scripts>activate.bat  
(venv) D:\python-flask-docker\venv\Scripts>
```

Verificamos la versión de Python con la que estaremos trabajando:

```
(venv) D:\python-flask-docker\venv\Scripts>python --version  
Python 3.10.2
```

Seguimos ahora con la instalación de Flask en nuestro entorno virtual por medio de “pip install flask”

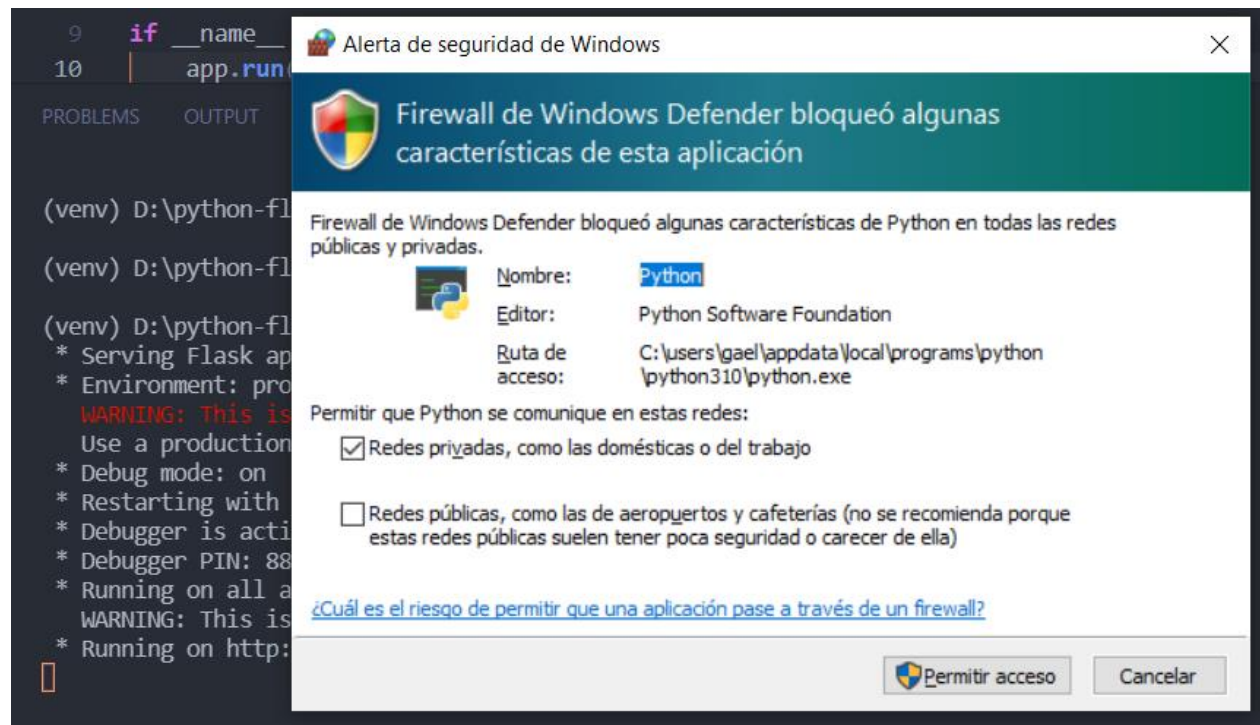
```
(venv) D:\python-flask-docker\venv\Scripts>pip install flask  
Collecting flask  
  Downloading Flask-2.0.3-py3-none-any.whl (95 kB)  
    95.6/95.6 KB 260.2 kB/s eta 0:00:00  
Collecting itsdangerous>=2.0  
  Downloading itsdangerous-2.1.1-py3-none-any.whl (15 kB)  
Collecting click>=7.1.2  
  Using cached click-8.0.4-py3-none-any.whl (97 kB)  
Collecting Werkzeug>=2.0  
  Downloading Werkzeug-2.0.3-py3-none-any.whl (289 kB)  
    289.2/289.2 KB 324.4 kB/s eta 0:00:00  
Collecting Jinja2>=3.0  
  Using cached Jinja2-3.0.3-py3-none-any.whl (133 kB)  
Collecting colorama  
  Using cached colorama-0.4.4-py2.py3-none-any.whl (16 kB)  
Collecting MarkupSafe>=2.0  
  Downloading MarkupSafe-2.1.1-cp310-cp310-win_amd64.whl (17 kB)  
Installing collected packages: Werkzeug, MarkupSafe, itsdangerous, colorama, Jinja2, click, flask  
Successfully installed Jinja2-3.0.3 MarkupSafe-2.1.1 Werkzeug-2.0.3 click-8.0.4 colorama-0.4.4 flask-2.0.3 itsdangerous-2.1.1
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Salimos dos niveles de directorio hasta llegar al directorio raíz y ahora ejecutamos nuestro archivo main, el cual se encuentra en la ruta "src/main.py":

```
(venv) D:\python-flask-docker\venv\Scripts>cd ..  
(venv) D:\python-flask-docker\venv>cd ..  
(venv) D:\python-flask-docker>python src/main.py  
* 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: on  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 883-516-899  
* Running on all addresses.  
  WARNING: This is a development server. Do not use it in a production deployment.  
* Running on http://192.168.100.43:4000/ (Press CTRL+C to quit)
```

Se muestra un aviso sobre el cortafuegos de Windows, lo cual es un buen indicio de que la API funciona. Aceptamos:





El código en este caso es una aplicación de Flask muy simple: importamos la clase “Flask” y el método “jsonify” del módulo “flask”, creamos una aplicación llamada “app”, añadimos una ruta raíz con un método “GET” la cual está asociada con una función que retorna un objeto en formato JSON y finalmente corremos la aplicación señalando como parámetros el host y puerto a utilizar e indicamos el atributo “debug=” como “True” para reflejar cualquier cambio al momento:

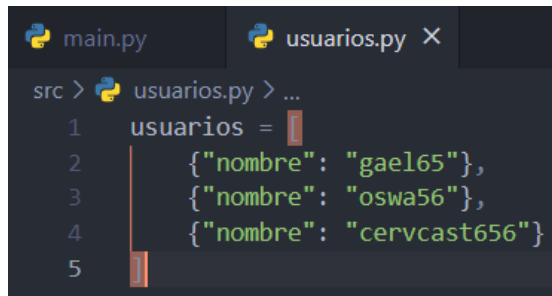
```
main.py ×
src > main.py > ...
1  from flask import Flask, jsonify
2
3  app = Flask(__name__)
4
5  @app.route('/', methods=['GET'])
6  def get():
7      return jsonify({"response": "hola mundo"})
8
9  if __name__ == "__main__":
10     app.run(host="0.0.0.0", port=4000, debug=True)
```

Abrimos la ruta en nuestro navegador indicando el puerto que especificamos en el código y podemos observar que se despliega la información del objeto JSON tal como se indica en el bloque de código correspondiente a la función asociada con la ruta:



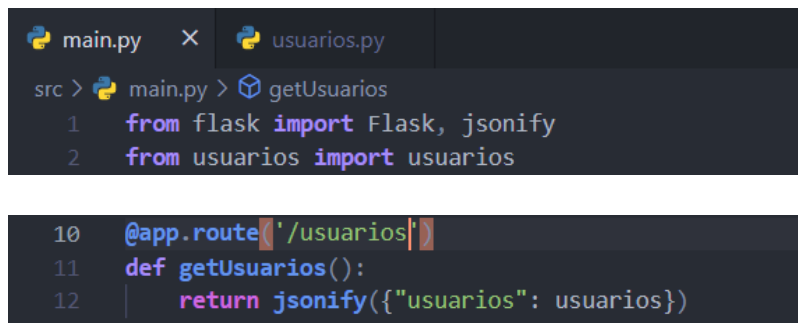
```
{
  "response": "hola mundo"
}
```

El siguiente paso es crear una segunda ruta, para lo cual necesitamos crear primero un segundo archivo con una lista de objetos con el formato de diccionarios para poder tratarlos como objetos JSON:



```
src > usuarios.py > ...
1 usuarios = [
2     {"nombre": "gae165"},
3     {"nombre": "oswa56"},
4     {"nombre": "cervcast656"}
5 ]
```

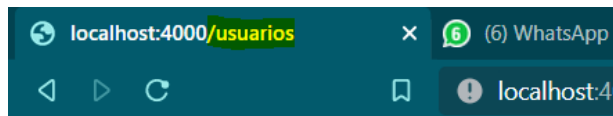
Lo siguiente es importar dicha lista en nuestro archivo “main.py” y crear la ruta, la cual del mismo modo que la otra ruta definida anteriormente toma retorna un objeto en formato JSON en el cual el valor de la clave “usuarios” es el objeto “usuarios” (lista de usuarios) definida en el archivo “usuarios.py”:



```
src > main.py > getUsuarios
1 from flask import Flask, jsonify
2 from usuarios import usuarios

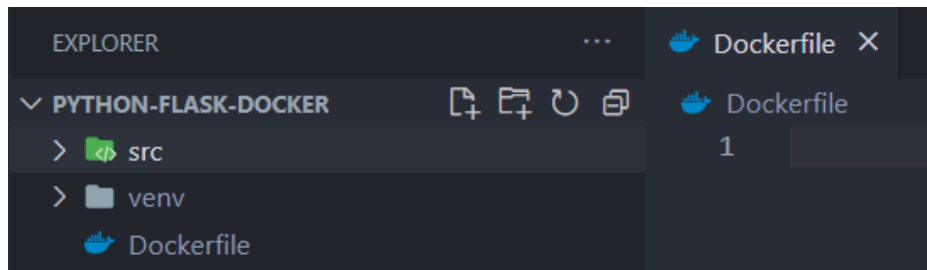
10 @app.route('/usuarios')
11 def getUsuarios():
12     return jsonify({"usuarios": usuarios})
```

Buscamos la ruta en nuestro navegador y podemos visualizar el resultado que se ve en la siguiente imagen:

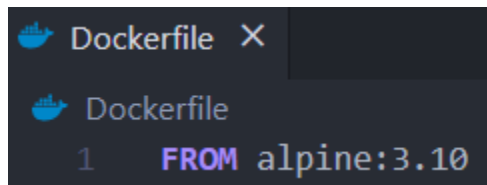


```
{
  "usuarios": [
    {
      "nombre": "gae165"
    },
    {
      "nombre": "oswa56"
    },
    {
      "nombre": "cervcast656"
    }
  ]
}
```

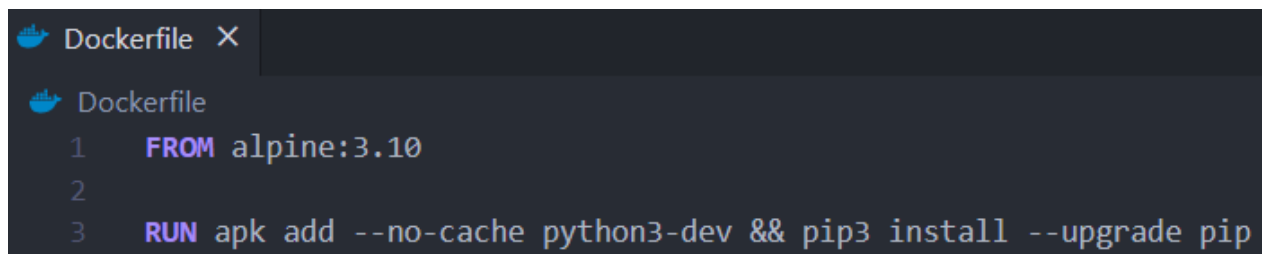
Ahora sí comenzamos con la migración de nuestro proyecto a Docker, para lo cual el primer paso es crear un archivo en nuestro directorio raíz llamado “Dockerfile”:



Ya que dentro de este archivo se van a definir todos los recursos necesarios para inicializar un entorno en el que se pueda ejecutar nuestro proyecto y dicho entorno se basa en un sistema operativo, indicamos entonces como primer paso en nuestro archivo Docker que utilizaremos Alpine, una distribución de Linux que en este caso funge como imagen para poder ejecutar nuestro proyecto:



El siguiente paso es definir en nuestro archivo de Docker una instrucción del sistema operativo que tenemos (Alpine) para instalar Python 3 junto con pip (para eso se especifica con “-dev”), dicha instrucción se define como un comando para administrar paquetes. Lo que se encuentra después de “&&” indica que se va a actualizar pip a la versión más reciente que se encuentre:



Con lo anterior ya podemos crear nuestra imagen, para lo cual indicamos el comando “docker build” seguido de la bandera “-t” y el nombre que tendrá nuestra imagen, seguido de eso indicamos “-f Docker” para hacer referencia a nuestro archivo de Docker y con el

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

“.” Estamos dando la orden de que se copien los recursos que ya están contenidos en las carpetas mostradas en pasos anteriores.

Verificamos que se creó la imagen por medio del comando “docker images” y en la siguiente imagen podemos observar que se creó con éxito:

```
(venv) D:\python-flask-docker>docker build -t mydockerimage -f Dockerfile .
[+] Building 2.0s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 31B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/alpine:3.10
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [1/2] FROM docker.io/library/alpine:3.10@sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
=> CACHED [2/2] RUN apk add --no-cache python3-dev && pip3 install --upgrade pip
=> exporting to image
=> => exporting layers
=> => writing image sha256:0d4cf5b1e84ecaf54b8b43889ca36f901cf81cc69b89e596fe53e8f5b6afa0ef
=> => naming to docker.io/library/mydockerimage

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

(venv) D:\python-flask-docker>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mydockerimage       latest             0d4cf5b1e84e       27 minutes ago     111MB

(venv) D:\python-flask-docker>
```

Del mismo modo la podemos visualizar en la interfaz de Docker for desktop:

NAME ↑		TAG	IMAGE ID	CREATED	SIZE
mydockerimage	IN USE	latest	0d4cf5b1e84e	22 minutes ago	110.55 MB

Para correr nuestra imagen debemos hacer uso del comando “docker run -it mydockerimage /bin/sh”, con “-it” indicamos que se va a ejecutar en modo interactivo y con “/bin/sh” indicamos donde se deben buscar los ejecutables necesarios, esto último ya no será necesario más adelante:

```
(venv) D:\python-flask-docker>docker run -it mydockerimage /bin/sh
/ # ls
bin      dev      etc      home     lib      media   mnt      opt      proc     root     run      sbin     srv      sys      tmp      usr      var
/ #
```

Dentro del modo interactivo podemos revisar información relacionada a los recursos instalados hasta este momento como por ejemplo Python y Pip. Una vez hecho esto salimos con “exit” para seguir configurando nuestra imagen:

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

```
/ # python3 --version
Python 3.7.10
/ # pip --version
pip 22.0.4 from /usr/lib/python3.7/site-packages/pip (python 3.7)
/ # exit

(venv) D:\python-flask-docker>
```

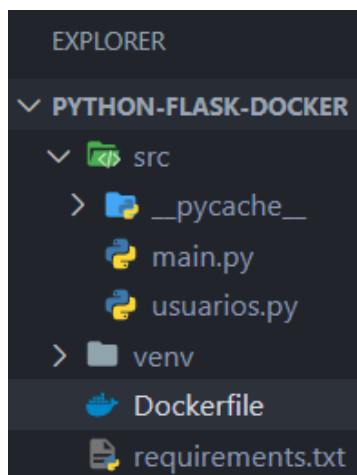
Por medio del comando “pip freeze” podemos mostrar en la terminal todos los recursos necesarios para poder ejecutar nuestra aplicación al momento, y si a eso le añadimos “>” seguido de un nombre de archivo con extensión “.txt”, podemos vaciar esta información en un archivo de texto:

```
(venv) D:\python-flask-docker>pip freeze
click==8.0.4
colorama==0.4.4
Flask==2.0.3
itsdangerous==2.1.1
Jinja2==3.0.3
MarkupSafe==2.1.1
Werkzeug==2.0.3

(venv) D:\python-flask-docker>pip freeze > requirements.txt

(venv) D:\python-flask-docker>
```

El archivo que se crea se puede visualizar al momento en el explorador de archivos de Visual Studio Code:



Finalmente, nuestro archivo Dockerfile queda de la siguiente manera:

```
Dockerfile X
Dockerfile
1 FROM alpine:3.10
2
3 RUN apk add --no-cache python3-dev && pip3 install --upgrade pip
4
5 WORKDIR /app
6
7 COPY . /app
8
9 RUN pip3 --no-cache-dir install -r requirements.txt
10
11 CMD ["python3", "src/main.py"]
```

Con “WORKDIR” indicamos el directorio sobre el cual se va a trabajar y con “COPY” indicamos lo mismo que cuando creamos la imagen por primera vez: en el directorio se copiarán los archivos que tenemos disponibles al momento. Con “RUN” en el mismo caso que el explicado anteriormente, es una instrucción del sistema operativo (Alpine) para indicar que ahora se van a instalar todos los recursos que se encuentren contenidos en el archivo “requirements.txt”.

Finalmente volvemos a construir nuestra imagen por medio del comando “docker build -t mydockerimage -f Dockerfile .”:

```
(venv) D:\python-flask-docker>docker build -t mydockerimage -f Dockerfile .
[+] Building 13.0s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 243B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/alpine:3.10
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [1/5] FROM docker.io/library/alpine:3.10@sha256:451eee8bedcb2f029756dc3e9d73bab0e7943c1ac55cff3a4861c52a0fdd3e98
=> [internal] load build context
=> => transferring context: 20.21MB
=> CACHED [2/5] RUN apk add --no-cache python3-dev && pip3 install --upgrade pip
=> [3/5] WORKDIR /app
=> [4/5] COPY . /app
=> [5/5] RUN pip3 --no-cache-dir install -r requirements.txt
=> exporting to image
=> => exporting layers
=> => writing image sha256:c004e5d9cef7756a4c79e3a8258c0165c3df80dab0a19813b9184e89aefc7888
=> => naming to docker.io/library/mydockerimage

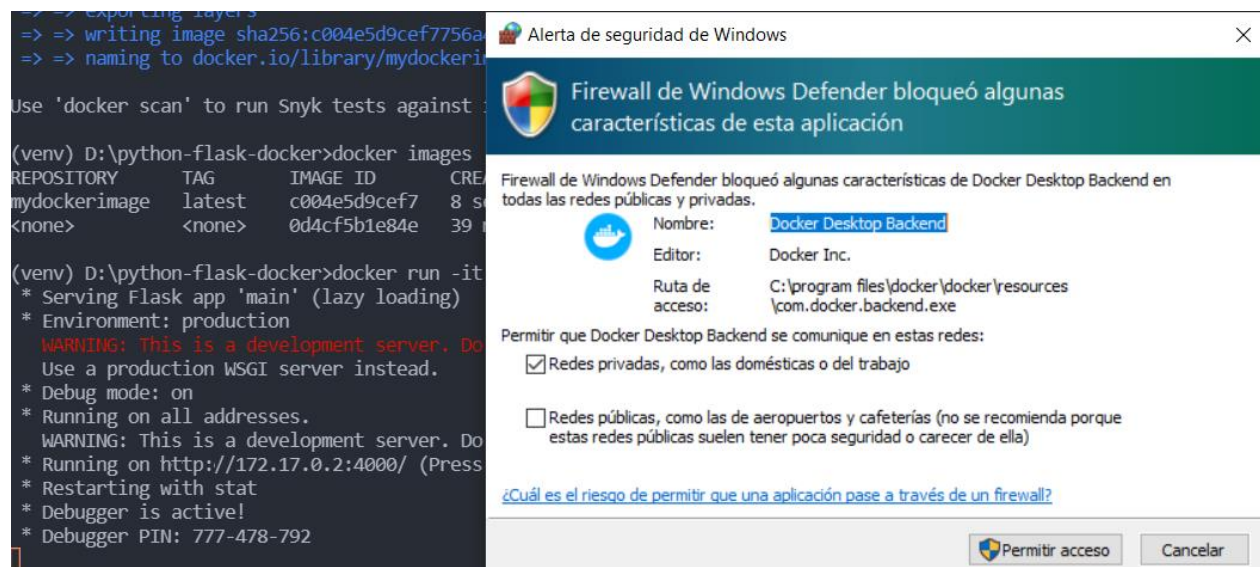
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

(venv) D:\python-flask-docker>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
mydockerimage       latest          c004e5d9cef7    8 seconds ago  135MB
<none>              <none>         0d4cf5b1e84e    39 minutes ago 111MB

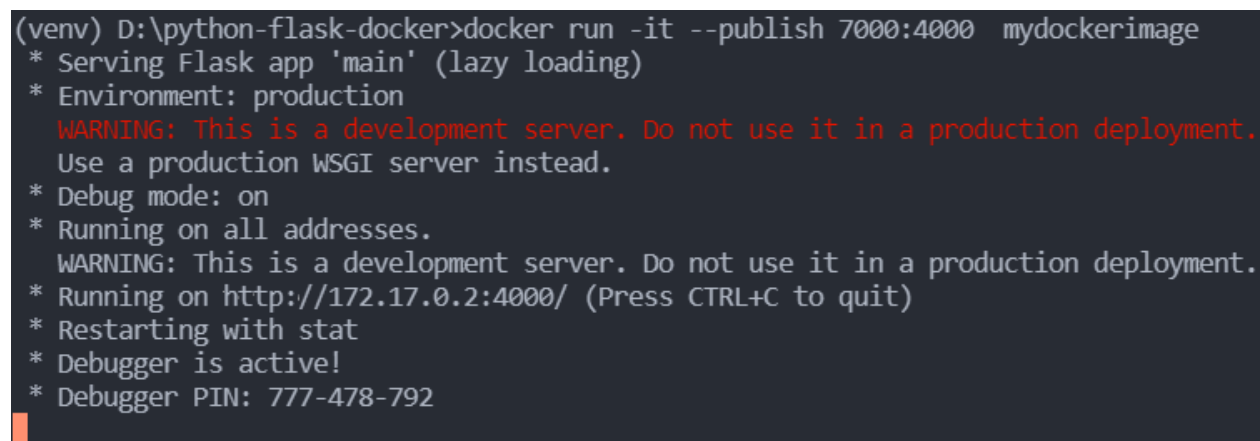
(venv) D:\python-flask-docker>
```

Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

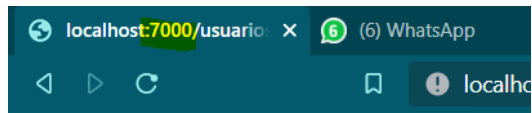
Se nos vuelve a mostrar el aviso asociado con el cortafuegos, lo cual indica que la aplicación sí va a correr de la forma esperada. Aceptamos nuevamente:



El comando utilizado para correr la aplicación esta vez es “docker run -it --publish 7000:4000 mydockerimage”. Con “--publish 7000:4000” indicamos que a pesar de que de forma local el puerto donde se exhibe nuestra aplicación es el 4000, en caso de que dicho puerto no esté disponible en la computadora externa donde se ejecute se puede mover a cualquier otro a gusto del desarrollador. Notamos que ya no es necesario utilizar “/bin/sh” puesto que la aplicación ya no necesita archivos binarios ya que ahora se está ejecutando como imagen, no de forma local:



Si accedemos a la ruta especificando el puerto que indicamos en el comando para ejecutar la imagen, podemos observar que efectivamente se ha ejecutado con éxito:



```
{
  "usuarios": [
    {
      "nombre": "gael65"
    },
    {
      "nombre": "oswa56"
    },
    {
      "nombre": "cervcast656"
    }
  ]
}
```

Observamos que la aplicación funciona de manera normal, se siguen realizando las peticiones con normalidad y para terminar la ejecución basta con presionar “CTRL+C”.

```
(venv) D:\python-flask-docker>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
mydockerimage       latest         c004e5d9cef7   8 seconds ago  135MB
<none>              <none>        0d4cf5b1e84e   39 minutes ago 111MB

(venv) D:\python-flask-docker>docker run -it --publish 7000:4000 mydockerimage
* 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: on
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:4000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 777-478-792
172.17.0.1 - - [21/Mar/2022 22:43:45] "GET /usuarios HTTP/1.1" 200 -
172.17.0.1 - - [21/Mar/2022 22:43:45] "GET /favicon.ico HTTP/1.1" 404 -
```

**Enlace al repositorio con el código fuente:**

<https://github.com/gaeltcervantes65/Docker.git>



## Conclusiones

Como conclusiones para la realización de esta actividad puedo mencionar que pude aprender muchas cosas nuevas ya que en lo personal nunca antes había utilizado Docker y tampoco sabía qué es.

Me pareció muy interesante el haberme dado cuenta de los alcances que tienen este tipo de herramientas y de lo útiles que pueden llegar a ser ya que presentan la solución a un problema muy grande y muy recurrente en el mundo del desarrollo de software, la compatibilidad. Cuando desarrollas algún proyecto de forma local en tu computadora con el software y herramientas en determinadas versiones o con ciertas características y después se la compartes a alguien para que la pruebe y falla, lo primero que se dice son comentarios como “en mi computadora sí funcionaba”, lo cual nunca brinda una solución a este problema y creo que al utilizar herramientas como Docker nos podemos ahorrar muchas horas intentando hacer que una aplicación funcione en diferentes entornos de ejecución ya que de forma rápida una imagen de Docker inicializa el entorno de trabajo requerido para cualquier producto de software.

## Fuentes de información

- Code, F. (2019, agosto 13). *Docker, Instalación en Windows10* [Video]. YouTube. <https://www.youtube.com/watch?v=BK-C2RofmTE&feature=youtu.be>
- Code, F. (2019, 20 agosto). *Docker & Python Flask. Contenedores con Python* [Video]. YouTube. <https://www.youtube.com/watch?v=YENw-bNHZwg&feature=youtu.be>