UNIVERSIDAD NACIONAL DE INGENIERÍA

Facultad de Ciencias
Escuela Profesional de Computación

Curso: Administración de Redes CC312-A

Laboratorio 6a : Crear una aplicación web de muestra en un contenedor Docker

Alumno: Vega Soldevilla Miguel Angel 20200570J

Catedrático: Yuri Ccoicca

ÍNDICE

1. Iniciar la Máquina virtual de DEVASC **2**

2. Crear un script Bash simple **2**

3. Crear una aplicación web de muestra **3**

4. Configurar la aplicación web para utilizar archivos de sitio web **5**

5. Crear un script de Bash para compilar y ejecutar un contenedor Docker **7**

6. Construir, ejecutar y verificar el contenedor Docker **8**

7. Conclusiones **13**

1. Iniciar la Máquina virtual de DEVASC

Ejecutamos la máquina virtual DEVASC.

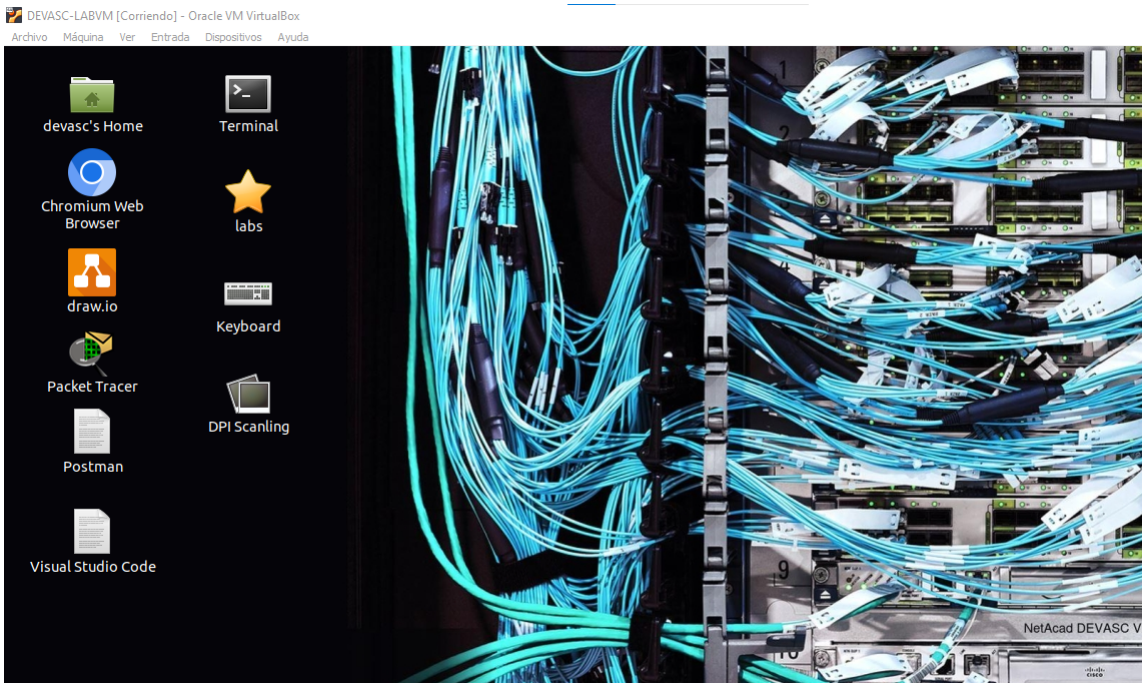

Figura 1

2. Crear un script Bash simple

Los scripts de Bash ayudan a los programadores a automatizar una variedad de tareas en un archivo de script.
En primera instancia, cambiamos nuestro directorio de trabajo al siguiente /**labs/devnet-src/sample-app** y agregamos un nuevo archivo con el comando **touch user-input.sh**, mostramos el contenido de la carpeta **sample-app** con **ls** y abrimos el archivo creado con **nano user-input.sh**.

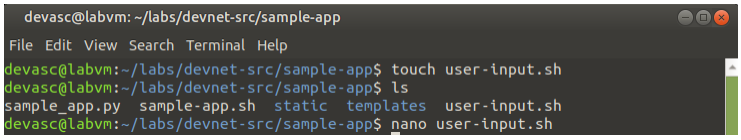

Figura 2

Luego, escribimos en el editor de texto lo que aparece en la imagen a continuación, presionamos CTRL + X, luego Y, luego ENTER para salir de nano y guardar el script.

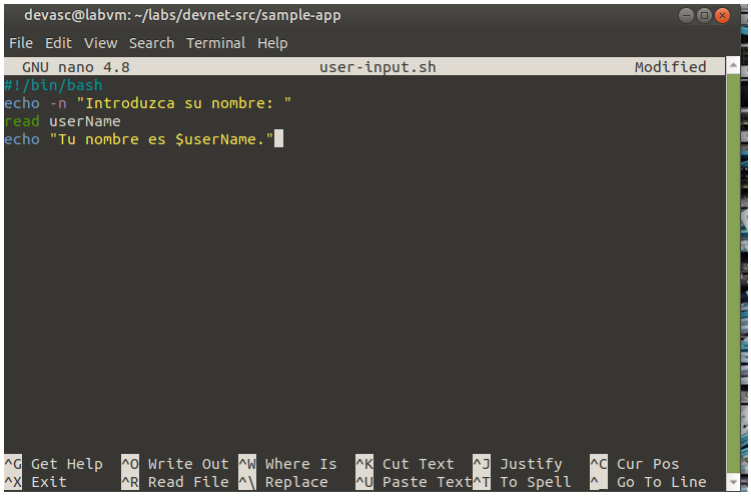

Figura 3

Ejecutamos directamente el script anterior desde la línea de comandos usando **bash user-input.sh**

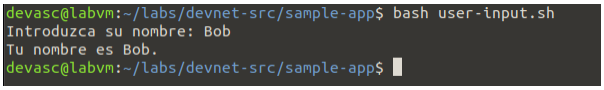

Figura 4

Modificamos el uso del script para que sea un archivo ejecutable para todos los usuarios mediante el comando **chmod** como se puede apreciar en la siguiente imagen.

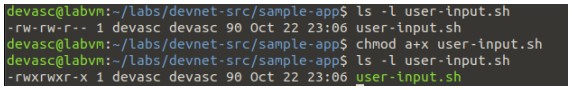

Figura 5

Podemos cambiar el nombre del archivo para quitar la extensión de modo que los usuarios no tengan que agregar **.sh** al comando, para ellos ejecutamos **mv (nombre del archivo con .sh) (nombre el archivo sin .sh)**. Posteriormente para ejecutar el comando usamos ./(nombre del archivo sin .sh).

Figura 6

3. CREAR UNA APLICACIÓN WEB DE MUESTRA

Antes de poder lanzar una aplicación en un contenedor Docker, primero necesitamos tener la aplicación. En esta parte, crearemos un script Python muy simple que mostrará la dirección IP del cliente cuando el cliente visite la página web.

Flask es un marco de aplicación web escrito en Python. Flask recibe solicitudes y luego proporciona una respuesta al usuario en la aplicación web. Esto es útil para aplicaciones web dinámicas porque permite la interacción del usuario y el contenido dinámico. Lo que hace que la aplicación web de muestra sea dinámica es que mostrará la dirección IP del cliente

Por lo cual, 6. instalamos Flask.

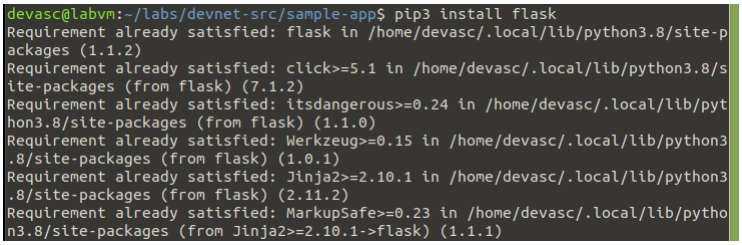

Figura 7

Mostramos el contenido del archivo **sample_app.py**, luego abrimos el archivo con el editor de texto de línea de comandos **nano**.

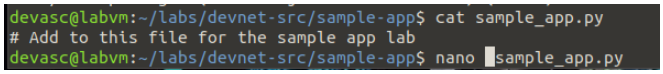

Figura 8

Escribimos las líneas de código según la guía de laboratorio para poder crear una aplicación web de muestra. Así como en la siguiente imagen. Posteriormente guardamos con Ctrl + X, luego escribimos Y y presionamos ENTER

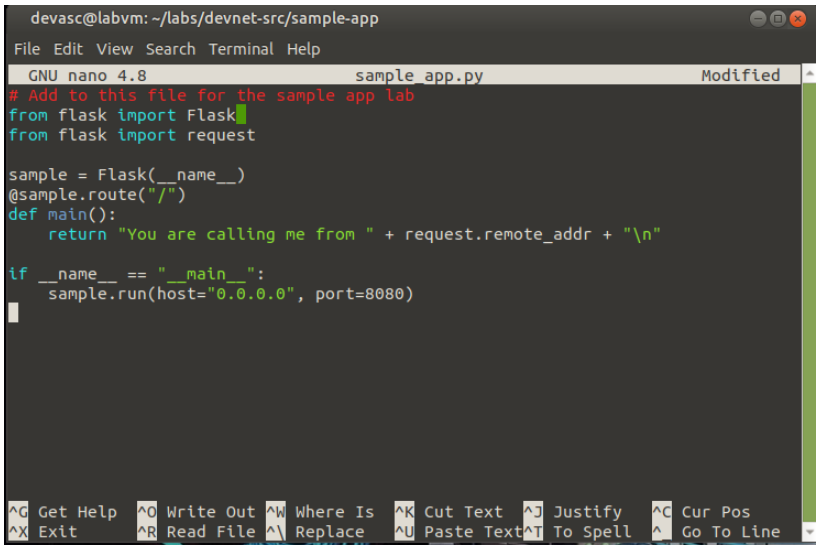

Figura 9

Ejecutamos el script con **python3 sample_app.py**. Ahora si vemos el siguiente mensaje entonces esto indica que el servidor de sample-app se está ejecutando y esta en espera de solicitudes entrantes.

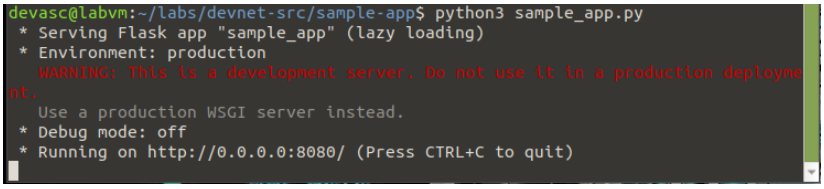

Figura 10

Abrimos Chromium, introducimos 0.0.0.0:8080 en el campo URL y presionamos enter.

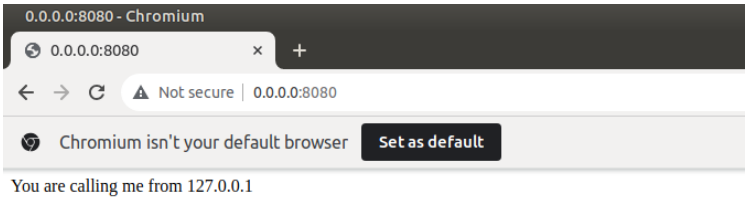

Figura 11

Esto resulta en la creación de una solicitud GET a través del protocolo HTTP. lo cual el servidor responde de manera exitosa.

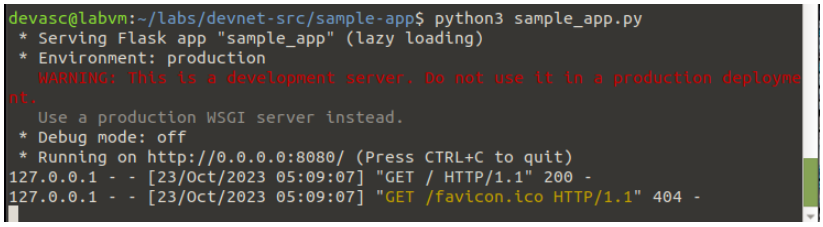

Figura 12

Tambien, introducimos en un nuevo terminal el siguiente comando curl http://0.0.0.0:8080 para verificar la respuesta del servidor.

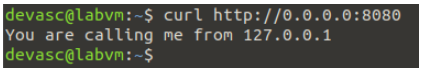

Figura 13

Se puede apreciar las solicitudes recibidas por el servidor. Ademas, podemos detener el servidor entrando a la terminal donde se encuentra ejecutando el servidor presionando CTRL + C

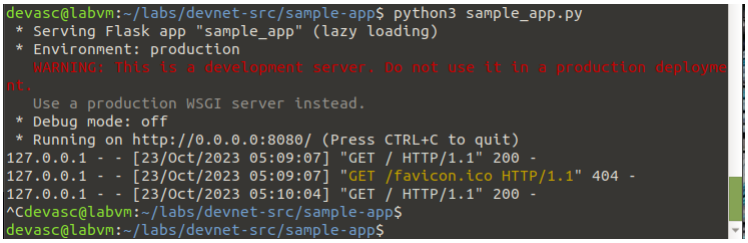

Figura 14

4. CONFIGURAR LA APLICACIÓN WEB PARA UTILIZAR ARCHIVOS DE SITIO WEB

Mostramos los archivos presentes en nuestra carpeta de trabajo. Luego, visualizamos el contenido de los archivos **index.html** y **style.css** utilizando el comando **cat**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ ls
sample_app.py  sample-app.sh  static  templates  user-input
devasc@labvm:~/labs/devnet-src/sample-app$ cat templates/index.html
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>You are calling me from {{request.remote_addr}}</h1>
</body>
</html>
devasc@labvm:~/labs/devnet-src/sample-app$ cat static/style.css
body {background: lightsteelblue;}
devasc@labvm:~/labs/devnet-src/sample-app$
```

Figura 15

Modificaremos el archivo **sample_app.py**, para ello utilizaremos el editor de texto **nano**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ nano sample_app.py
```

Figura 16

Modificamos el archivo según la guía de laboratorio.

```
devasc@labvm: ~/labs/devnet-src/sample-app
File  Edit  View  Search  Terminal  Help
  GNU nano 4.8                    sample_app.py                    Modified
# Add to this file for the sample app lab
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)
@sample.route("/")
def main():
        return render_template ("index.html")

if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```

Figura 17

Guardamos y ejecutamos con el script con **python3 sample_app.py**. Deberíamos obtener una salida como la siguiente.

```
devasc@labvm:~/labs/devnet-src/sample-app$ python3 sample_app.py
 * Serving Flask app "sample_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Figura 18

Abrimos el navegador web Chromium e introducimos 0.0.0.0:8080 en el campo URL. Debería obtener la misma salida que antes. Sin embargo, su fondo será azul metálico claro y el texto tendrá el formato H1

Figura 19

Abrimos otra ventana en la termina e utilizamos el comando **curl http://0.0.0.0:8080**. El código dinámico de Python se reemplazará por el valor de **request.remote_addr**.

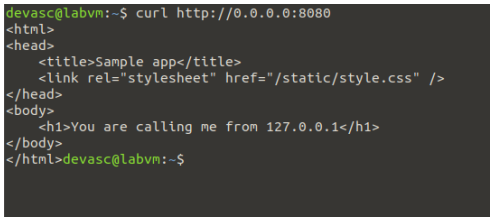

Figura 20

Volvemos a la ventana de la terminal para ver las solicitudes recibidas por el servidor y presionamos CTRL+C para detener el servidor

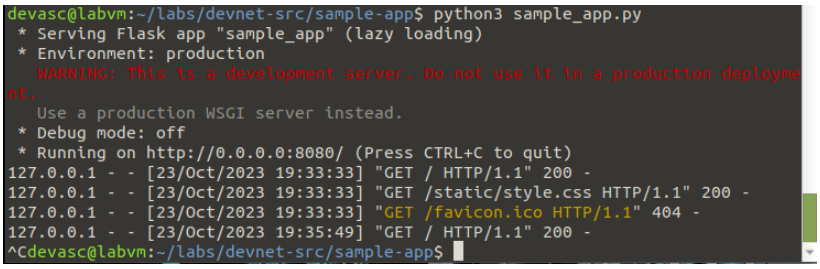

Figura 21

5. Crear un script de Bash para compilar y ejecutar un contenedor Docker

Abrimos el archivo de script **sample-app.sh** con el editor de texto nano.

Figura 22

Editamos el script según la guía de laboratorio. Y posteriormente guardamos el script bash. El script automatiza la creación de una estructura de directorios, copia archivos y crea un archivo Dockerfile. Luego, construye una imagen de Docker y ejecuta un contenedor basado en esa imagen

sample-app.sh

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile
echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile
echo "EXPOSE 8080" >> tempdir/Dockerfile
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir
docker build -t sampleapp .

docker run -t -d -p 8080:8080 --name samplerunning sampleapp

docker ps -a

```

Figura 23

Explicaremos el contenido del archivo linea por linea:

- Crea un directorio llamado "tempdirçon la instrucción `mkdir tempdir`.
- Dentro de "tempdir", crea dos subdirectorios llamados "templatesz "staticçon las instrucciones `mkdir tempdir/templates` y `mkdir tempdir/static`.
- Copia el archivo "sample_app.py.al directorio "tempdirçon la instrucción `cp sample_app.py tempdir/.`.
- Copia todos los archivos y subdirectorios desde la carpeta "templates.actual al directorio "tempdir/templatesçon la instrucción `cp -r templates/* tempdir/templates/.`.
- Copia todos los archivos y subdirectorios desde la carpeta "static.actual al directorio "tempdir/staticçon la instrucción `cp -r static/* tempdir/static/.`.
- Agrega líneas al archivo "Dockerfile.en "tempdir"para configurar una imagen Docker. Estas líneas incluyen:
  - `FROM python`: Utiliza la imagen base de Python.
  - `RUN pip install flask`: Instala el paquete Flask.
  - Copia los archivos y directorios de "tempdir.al directorio /home/myapp.en la imagen.
  - `EXPOSE 8080`: Expone el puerto 8080.
  - `CMD python3 /home/myapp/sample_app.py`: Establece el comando predeterminado para ejecutar la aplicación cuando se inicie el contenedor.
- Cambia al directorio "tempdirçon `cd tempdir`.
- Construye una imagen de Docker con la instrucción `docker build -t sampleapp .`.
- Ejecuta un contenedor basado en la imagen recién creada con la instrucción `docker run -t -d -p 8080:8080 -name samplerunning sampleapp`. Esto lanza el contenedor con el nombre "samplerunningz lo mapea al puerto 8080.
- Finalmente, muestra una lista de todos los contenedores Docker con `docker ps -a`.

6. Construir, ejecutar y verificar el contenedor Docker

Ejecutar el script bash desde la línea de comandos. Debería verse un resultado similar a lo siguiente. Después de crear los directorios tempdir, el script ejecuta los comandos para crear el contenedor Docker. Observe que el paso 7/7 de la salida ejecuta sample_app.py que crea el servidor web. Además, observe el ID del contenedor. Luego colocamos docker images para listar las imágenes de contenedores Docker que están almacenadas localmente en nuestro sistema. Estas imágenes son la base para crear contenedores.

La salida del comando incluye información sobre las imágenes, como su identificador (ID), repositorio, etiqueta, tamaño y cuando fueron creadas.

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash ./sample-app.sh
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
 ---> 17e65561fd2c
Step 2/7 : RUN pip install flask
 ---> Running in 2a6a702428d8
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c23096649b908c809c69388a805a571a3bea44362fe87e33fc3
afa01f/flask-3.0.0-py3-none-any.whl.metadata
  Downloading flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Obtaining dependency information for Werkzeug>=3.0.0 from https://files.pythonhosted.org/packages/b6/a5/54b01f663d60d5334f6c9c87c26274e94617a4fd463d
812463626423b10d/werkzeug-3.0.0-py3-none-any.whl.metadata
  Downloading werkzeug-3.0.0-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.1/133.1 kB 3.6 MB/s eta 0:00:00
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/00/2e/d53fa4befbf2cfa713304affc7ca780ce4fc1fd87105277
71b58311a3229/click-8.1.7-py3-none-any.whl.metadata
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Obtaining dependency information for blinker>=1.6.2 from https://files.pythonhosted.org/packages/bf/2b/11bcedb7dee4923253a4a21bae3be854bcc4f06295bd8
27756352016d97c/blinker-1.6.3-py3-none-any.whl.metadata
  Downloading blinker-1.6.3-py3-none-any.whl.metadata (1.9 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
```

Figura 24

```
Downloading werkzeug-3.0.0-py3-none-any.whl (226 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.6/226.6 kB 7.8 MB/s eta 0:00:00
Downloading MarkupSafe-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (28 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.0 blinker-1.6.3 click-8.1.7 flask-3.0.0 itsdangerous-2.1.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instea
d: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: pip install --upgrade pip
Removing intermediate container 2a6a702428d8
 ---> c6fbfdb4b534
Step 3/7 : COPY ./static /home/myapp/static/
 ---> 30396d9fea47
Step 4/7 : COPY ./templates /home/myapp/templates/
 ---> af04b792f15b
Step 5/7 : COPY sample_app.py /home/myapp/
 ---> a7b1839341ec
Step 6/7 : EXPOSE 8080
 ---> Running in d34b0e13bf2e
Removing intermediate container d34b0e13bf2e
 ---> 65f00ede8c2f
Step 7/7 : CMD python3 /home/myapp/sample_app.py
 ---> Running in e3fde817a035
Removing intermediate container e3fde817a035
 ---> 59b226d547d9
Successfully built 59b226d547d9
Successfully tagged sampleapp:latest
WARNING: Your kernel does not support swap limit capabilities or the cgroup is not mounted. Memory limited without swap.
26741a05825b1150399785914l6babb86be0140535f56b76428d804d4b7fbf43
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                      PORTS                                       NAMES
26741a05825b   sampleapp      "/bin/sh -c 'python3…"   1 second ago    Up Less than a second       0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   samplerunning
ff4f08f5ee57   17e65561fd2c   "/bin/sh -c 'pip ins…"   26 minutes ago  Exited (2) 26 minutes ago                                               suspicious_blackburn
devasc@labvm:~/labs/devnet-src/sample-app$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
sampleapp    latest    59b226d547d9   20 minutes ago   1.03GB
python       latest    17e65561fd2c   8 days ago       1.02GB
devasc@labvm:~/labs/devnet-src/sample-app$
```

Figura 25

Verificamos la creación de la carpeta **tempdir** con **ls** y posteriormente verdificamos el contenido de la carpeta tempdir.

```
devasc@labvm:~/labs/devnet-src/sample-app$ ls
sample_app.py  sample-app.sh  static  tempdir  templates  user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ls tempdir
Dockerfile  sample_app.py  static  templates
```

Figura 26

Vamos a examinar el contenido del archivo Dockerfile geneardo por el script bash, sin la inclusión de los comandos **echo**, utilizando el comando **cat tempdir/Dockerfile** en la terminal.

```
devasc@labvm:~/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
RUN pip install flask
COPY ./static /home/myapp/static/
COPY ./templates /home/myapp/templates/
COPY sample_app.py /home/myapp/
EXPOSE 8080
CMD python3 /home/myapp/sample_app.py
devasc@labvm:~/labs/devnet-src/sample-app$
```

Figura 27

Usamos el siguiente comando para enviar la salida del comando **docker ps -a** al archivo de texto **running.txt** donde podemos verlo mejor, luego ejecutamos **cat running.txt** para su visualización.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/labs/devnet-src/sample-app$ cat running.txt
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS                     PORTS                                       NAMES
26741a05825b   sampleapp      "/bin/sh -c 'python3…"   About an hour ago   Up About an hour           0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   samplerunning
ff4f08f5ee57   17e65561fd2c   "/bin/sh -c 'pip ins…"   2 hours ago         Exited (2) 2 hours ago                                                 suspicious_blackburn
devasc@labvm:~/labs/devnet-src/sample-app$
```

Figura 28

El contenedor Docker crea su propia dirección IP a partir de un espacio de direcciones de red privada. Verificar que la aplicación web se esté ejecutando e informe de la dirección IP. En un navegador web en **http://localhost:8080**, se debería ver el mensaje **Me estás llamando desde 172.17.0.1** con formato H1 sobre un fondo azul metálico claro.

Figura 29

También, podemos usar el comando curl.

```
devasc@labvm: ~
File  Edit  View  Search  Terminal  Help
devasc@labvm:~$ curl http://172.17.0.1:8080
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>You are calling me from 172.17.0.1</h1>
</body>
</html>devasc@labvm:~$
```

Figura 30

Usamos el comando **ip address** para mostrar todas las direcciones IP utilizadas por la instancia de la VM DEVASC.

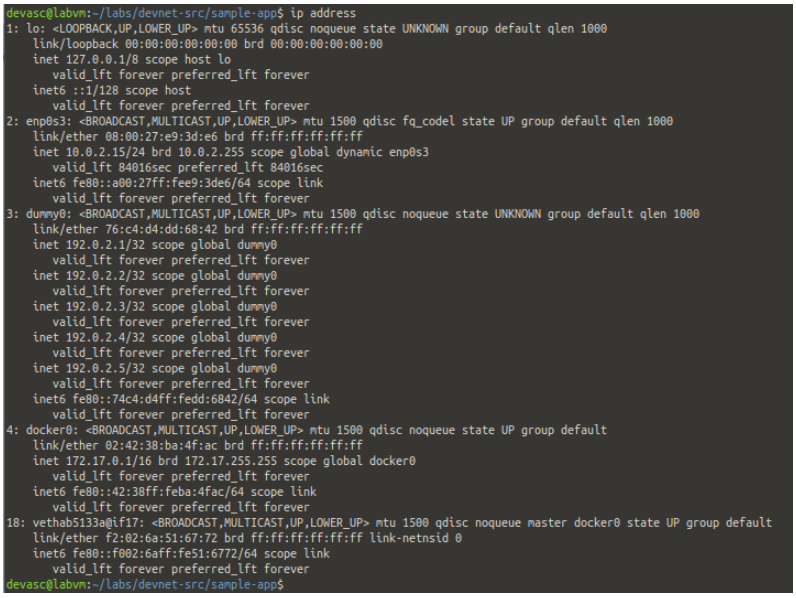

Figura 31

Para ingresar al contenedor en funcionamiento, ejecute el comando **docker exec -it** y especifique el nombre del contenedor en ejecución, en este caso,**samplerunning**, además, indicamos que deseamos abrir un intérprete de comandos bash (/**bin**/**bash**). Luego, podemos explorar el contenido del contenedor Docker ejecutando el comando **ls**.Luego, Introducimos el comando **ls home**/**myapp** con lo cual podmeos ver las carpetas y archivos de nuestra aplicacion web, posteriormente para salir escribimos exit.

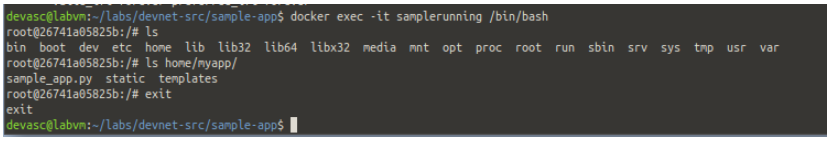

Figura 32

Para detener un contenedor se escribe docker stop (nombre del contenedor)

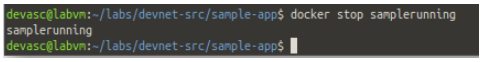

Figura 33

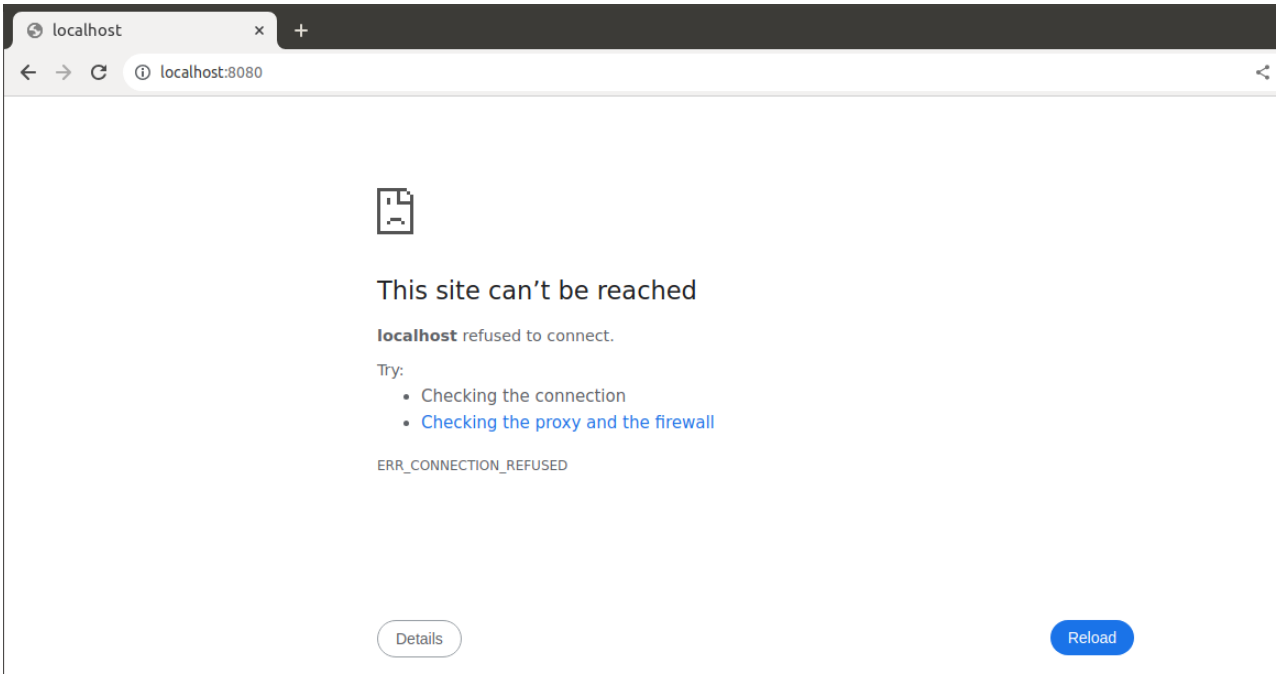

Figura 34

Pero, notamos que aun existe ingresando el comando docker ps -a.

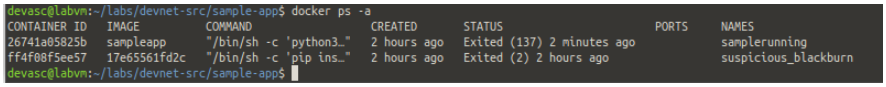

Figura 35

Se puede reiniciar un contenedor detenido con el comando **docker start samplerunning**, donde **samplerunning** es el nombre del contenedor.

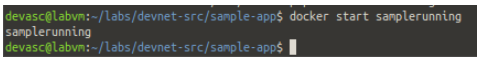

Figura 36

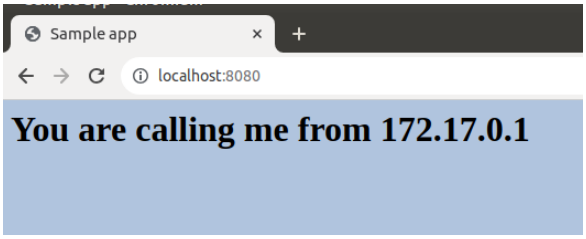

Figura 37

Para eliminar permanentemente el contenedor, primero se debe detener y luego remover con el comando **docker rm samplerunning** y utilizamos el comando **docker ps -a** para verificar que se ha eliminado el contenedor.

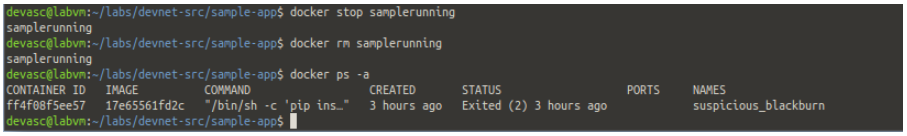

Figura 38

7. CONCLUSIONES

En este laboratorio, se logró con éxito la automatización de la creación de una aplicación web utilizando Flask, un framework web ligero para Python. Para lograr esto, se utilizó un script Bash que facilitó la generación de la aplicación. Flask es especialmente destacado por su capacidad para generar contenido HTML de manera dinámica, permitiendo la construcción de páginas web dinámicas y flexibles.
Flask simplifica el manejo de solicitudes HTTP, lo que resulta fundamental cuando los usuarios visitan la aplicación, y la generación de respuestas, es decir, la información mostrada a los usuarios. Además, se utilizó un contenedor Docker para alojar la aplicación. Este contenedor Docker incorporó otros archivos esenciales necesarios para su funcionamiento.
La ventaja de esta configuración radica en la flexibilidad que ofrece. La aplicación web puede ejecutarse en el interior del contenedor Docker, lo que permite detener y reiniciar el contenedor según sea necesario. Esto brinda una versatilidad fundamental en el desarrollo y las pruebas de aplicaciones web, lo que facilita la iteración y mejora de la aplicación de manera eficiente.