

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS



ÁREA DE CIENCIA DE LA COMPUTACIÓN

6° LABORATORIO - CC312

1 PARTE

- **TÍTULO:** Crear una aplicación web de muestra en un contenedor Docker
- **ALUMNO:**
JHONATAN POMA MARTINEZ 20182729F
- **PROFESORES:** YURI JAVIER.,CCOICCA PACASI

2023

OBJETIVOS:

Parte 1: Iniciar la Máquina Virtual (Virtual Machine) (VM) DEVASC

Parte 2: Crear un script Bash simple

Parte 3: Crear una aplicación web de muestra

Parte 4: Configurar la aplicación web para utilizar archivos de sitio web

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

Parte 6: Construir, ejecutar y verificar el contenedor Docker

Aspectos básicos:

En este laboratorio, revisaremos las técnicas básicas de **scripting bash** ya que este es un requisito previo para el resto del laboratorio. A continuación, compilaremos y modificaremos una secuencia de comandos de Python para una aplicación web simple. Además, se creará un **script bash** para automatizar el proceso de creación de un **Dockerfile**, creación del contenedor Docker y ejecución del contenedor Docker. Finalmente, se utilizarán los comandos de acoplador para investigar las complejidades de la instancia del contenedor Docker.

DEFINICIONES previas:

Bash : "Bourne-Again SHell", es un intérprete de comandos o shell de Unix que se utiliza en sistemas operativos basados en Unix, como Linux y macOS. Es uno de los shells más populares y ampliamente utilizados en el mundo de la informática. Es altamente personalizable y configurable, lo que permite a los usuarios crear sus propios comandos y alias. También proporciona una amplia variedad de herramientas y utilidades de línea de comandos que se pueden utilizar para trabajar con archivos, directorios, procesos y servicios del sistema.

El **scripting en Bash** implica escribir una serie de comandos en un archivo de texto plano, que se ejecutan de forma secuencial cuando se ejecuta el script. Los scripts de Bash pueden ser utilizados para automatizar tareas repetitivas, procesamiento de archivos y para la creación de pequeñas aplicaciones.

Docker: Es una plataforma de virtualización de contenedores que permite a los desarrolladores empaquetar aplicaciones y servicios en un contenedor aislado y portátil que se puede ejecutar en cualquier sistema operativo y en cualquier plataforma de hardware. Docker proporciona una solución de virtualización ligera y eficiente que utiliza los recursos del sistema de forma eficiente y es fácilmente escalable. Docker se utiliza ampliamente en el desarrollo de aplicaciones y en la implementación de aplicaciones en la nube y en entornos de producción. Además, Docker facilita la construcción, distribución y ejecución de aplicaciones y servicios de forma rápida y consistente.

Dockerfile es un archivo de texto plano que contiene instrucciones y comandos utilizados para construir una **imagen de Docker**. Cada instrucción en el Dockerfile crea una nueva capa en la imagen y las capas se pueden volver a utilizar en otras imágenes. El Dockerfile se utiliza para configurar el entorno de la aplicación y establecer las dependencias necesarias para ejecutar el servicio en un contenedor de Docker.

CI/CD: La Integración Continua (CI) es una práctica de desarrollo de software que se enfoca en integrar el código de los desarrolladores en un repositorio común varias veces al día.

Por otro lado, la Implementación Continua (CD) es una práctica que se enfoca en automatizar el proceso de liberación de software, de tal manera que las nuevas características y correcciones de errores puedan ser entregadas rápidamente y de forma segura a los usuarios finales. Con CD, el software se construye, prueba y se despliega automáticamente en diferentes entornos (desarrollo, pruebas, producción, etc.) sin la necesidad de intervención manual.

La combinación de CI/CD se refiere a la implementación de la Integración Continua y la Implementación Continua como un proceso continuo e integrado en el ciclo de vida del desarrollo de software. Esto

permite que los equipos de desarrollo puedan entregar nuevas características y correcciones de errores más rápido y con mayor calidad, al mismo tiempo que se reducen los riesgos y se mejora la eficiencia del proceso de entrega de software.

FLASK: Es un **framework web de Python** que se utiliza para desarrollar aplicaciones web rápidas y escalables. Fue desarrollado por Armin Ronacher en 2010 y se basa en el principio de "micro-frameworks", lo que significa que es minimalista y proporciona solo lo esencial para construir aplicaciones web. Flask se integra fácilmente con otras herramientas y bibliotecas de Python, lo que lo hace flexible y fácil de usar.

Parte 1: Iniciar la Máquina Virtual (Virtual Machine) (VM) DEVASC

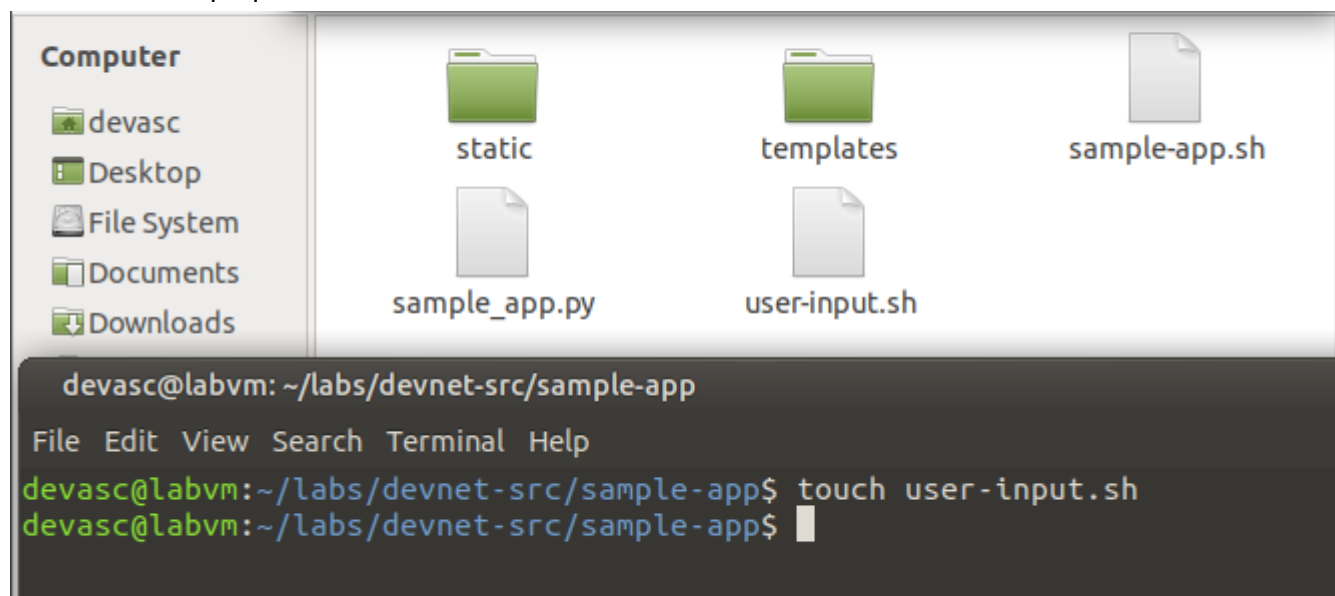
Máquina virtual Iniciada.

Parte 2: Crear un script Bash simple

El conocimiento de Bash es crucial para trabajar con integración continua, implementación continua CI/CD, contenedores y con su entorno de desarrollo. Los scripts de Bash ayudan a los programadores a automatizar una variedad de tareas en un archivo de script. En esta parte, revisará brevemente cómo crear un script bash.

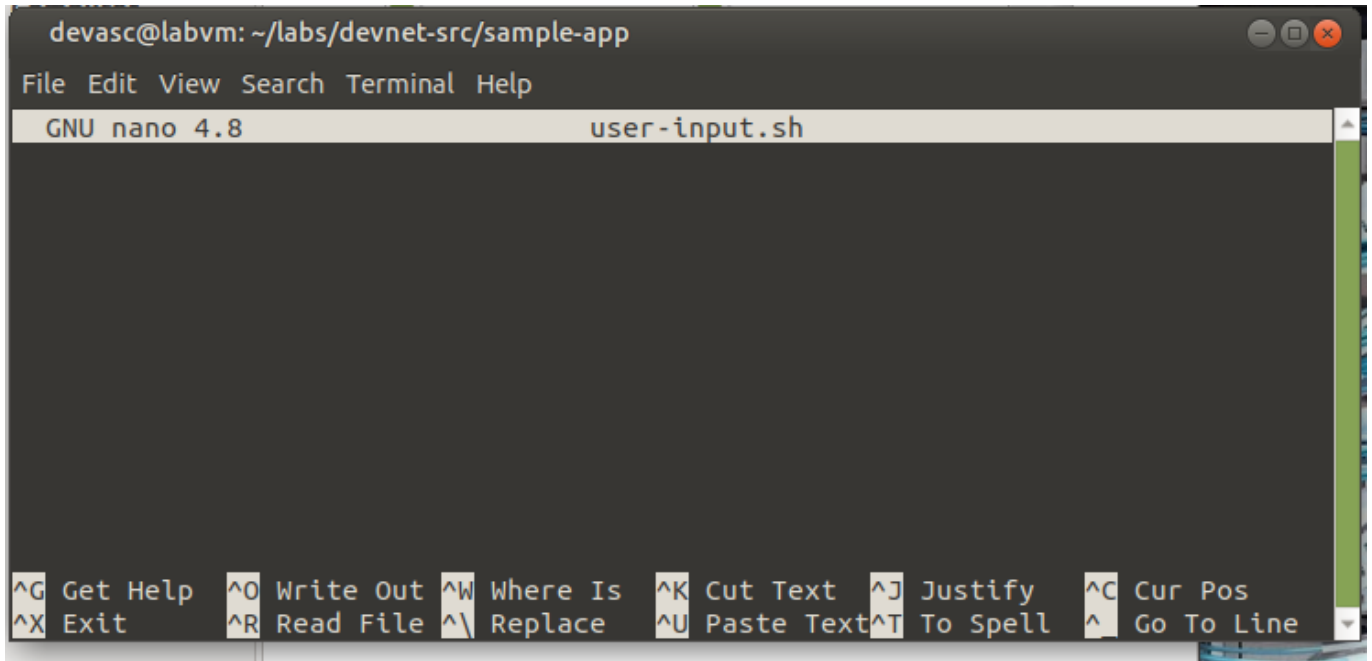
Más adelante, en el laboratorio, usará un script bash para automatizar la creación de una aplicación web dentro de un contenedor Docker.

1. creamos una archivo **user-input.sh**, la extensión **.sh** es una abreviatura de "**shell**", se utiliza para archivos de script que contendrá comandos de shell.



2. Ahora abrimos el archivo en un editor de texto "**nano**" que es un editor de líneas de comandos, que nos permitirá crear y editar archivos de texto en la terminal del sistema operativo.

devasc @labvm: ~/labs/devnet-src/sample-app\$ nano user-input.sh

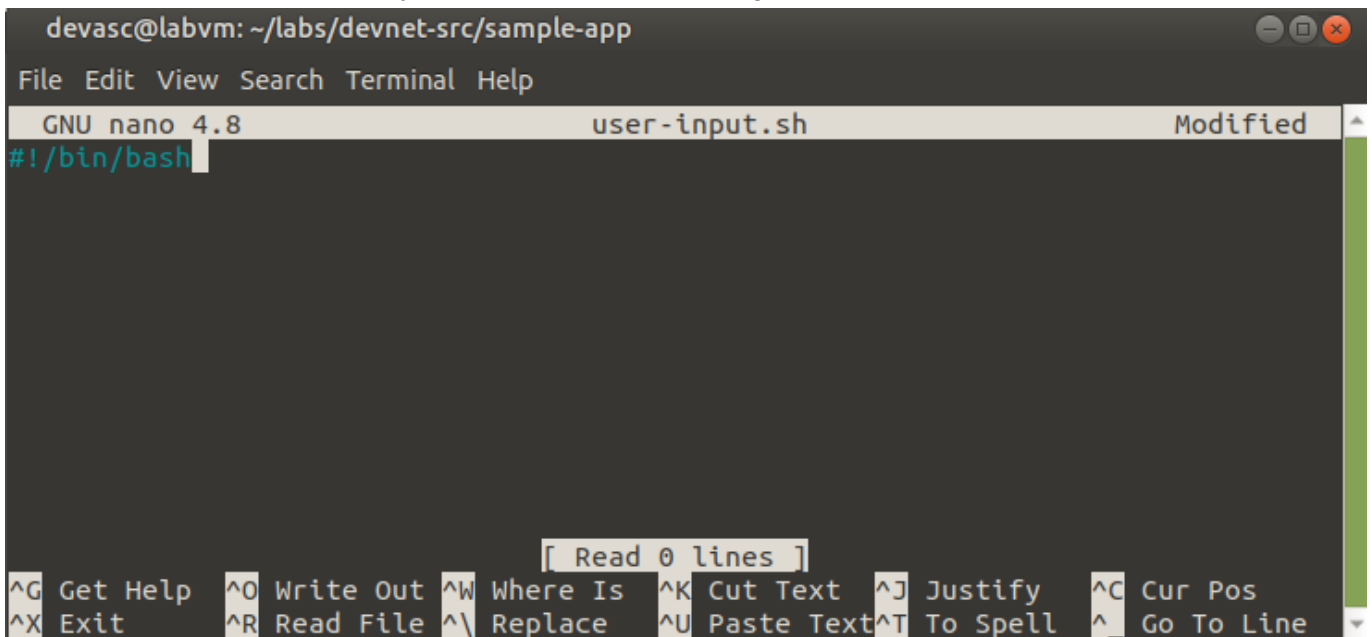


```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 user-input.sh

^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
```

3. Añadimos el “**she-bang**”, que es una forma abreviada de referirse a los caracteres “**#!**”.
la línea “**#!/bin/bash**” indica que el archivo debe ser ejecutado por el intérprete de comandos Bash.
La sintaxis completa de la línea “she-bang” es “**#!<ruta_del_interprete>**”, donde
“**<ruta_del_interprete>**” es la ruta completa del intérprete de comandos que se utilizará para ejecutar el archivo.

Actualizamos con “CTRL +O “ y pulsamos “ENTER” para guardar.



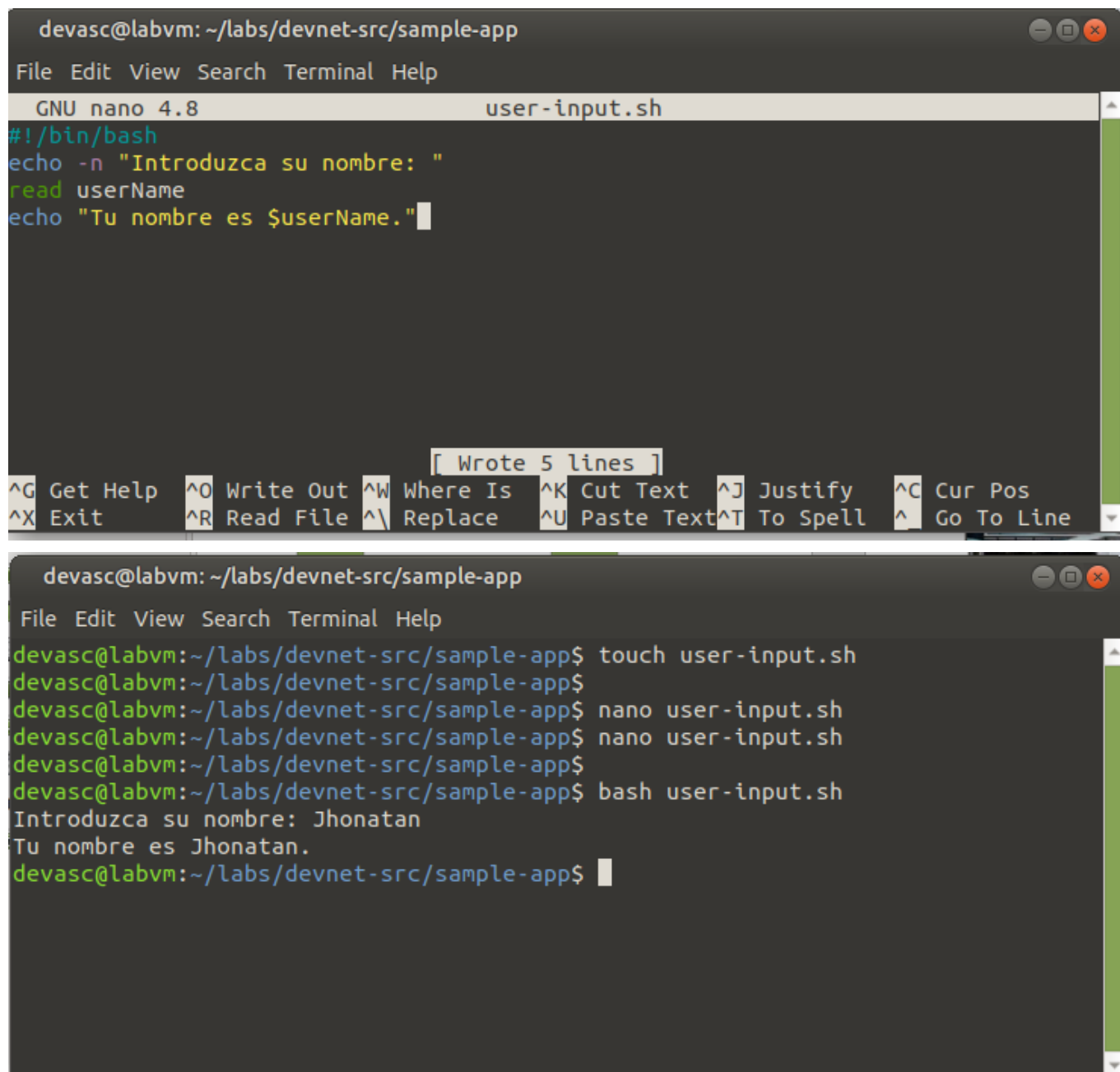
```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 user-input.sh Modified
#!/bin/bash

[ Read 0 lines ]
^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
```

4,5,6. Usamos el comando “**echo**”, para imprimir en la pantalla el “**introduzca su nombre**”, la opción “**-n**”, lo usamos para evitar que haga un salto de línea en el momento que nos pida el nombre. Después usamos el comando “**read**”, para asignar lo que ingresamos en la variable USERNAME, luego se imprime el mensaje más su nombre, el signo de dólar seguido del nombre de la variable “**\$UserName**” se utiliza para hacer referencia al valor almacenado en la variable.

Guardamos y salimos, ejecutamos :

devasc @labvm: ~/labs/devnet-src/sample-app\$ bash user-input.sh



The image consists of two screenshots of a terminal window. The top screenshot shows the nano text editor editing a file named 'user-input.sh'. The content of the file is: `#!/bin/bash`, `echo -n "Introduzca su nombre: "`, `read userName`, and `echo "Tu nombre es $userName."`. The bottom screenshot shows the terminal commands: `touch user-input.sh`, `nano user-input.sh` (run twice), and `bash user-input.sh`. The output of the script execution is: `Introduzca su nombre: Jhonatan` and `Tu nombre es Jhonatan.`

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 user-input.sh
#!/bin/bash
echo -n "Introduzca su nombre: "
read userName
echo "Tu nombre es $userName."
[ Wrote 5 lines ]
^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

devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Introduzca su nombre: Jhonatan
Tu nombre es Jhonatan.
devasc@labvm:~/labs/devnet-src/sample-app$
```

7. Ahora **“Cambiamos el modo del script a un archivo ejecutable para todos los usuarios”** esto significa dar permisos a todos los usuarios del sistema para ejecutar el script de shell.

Para hacer que un script sea ejecutable, debemos cambiar el "modo" a "ejecutable". Esto se logra utilizando el comando **"chmod"** en la terminal, que cambia los permisos de acceso de un archivo o directorio.

Ingresamos, **"ls -l user-input.sh"** esto nos mostrará los detalles del archivo "user-input.sh", permisos y propietario/grupo, número de enlace duros, nombre de usuario, tamaño del archivo, fecha y hora y nombre del archivo.

El símbolo **"+"** se utiliza para agregar permisos, y la letra **"x"** indica que se está otorgando permiso de ejecución. Cuando ejecutamos este comando, se otorga permiso de ejecución para el archivo de shell **"user-input.sh"** para el usuario actual.

La letra **"a"** significa "todos", lo que significa que se están dando permisos de ejecución a todos los usuarios, no solo al usuario actual.

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Introduzca su nombre: Jhonatan
Tu nombre es Jhonatan.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 91 May  5 04:48 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 91 May  5 04:48 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
```

8.9. Podemos cambiar el nombre del archivo para quitar la extensión de modo que los usuarios no tengan que agregar `.sh` al comando para ejecutar la secuencia de comandos. Ahora el script lo podemos ejecutar desde la línea de comandos sin el comando `source` o una extensión. Para ejecutar un script `bash` sin el comando `source`, debe escribir `./` delante del nombre del script.

```
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Introduzca su nombre: Jhonatan
Tu nombre es Jhonatan.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 91 May  5 04:48 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 91 May  5 04:48 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Introduzca su nombre: Jhonatan
Tu nombre es Jhonatan.
devasc@labvm:~/labs/devnet-src/sample-app$
```

10. Algunos script `bash`.

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 user-input.sh
#!/bin/bash
echo -n "Introduzca su nombre: "
read name
echo "su nombre es $name"

for i in {1..10}
do
    echo $i
done

[ Read 11 lines ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell

devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 117 May  5 05:34 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 117 May  5 05:34 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Introduzca su nombre: Jhonatan
su nombre es Jhonatan
1
2
3
4
5
6
7
8
9
10
devasc@labvm:~/labs/devnet-src/sample-app$
```

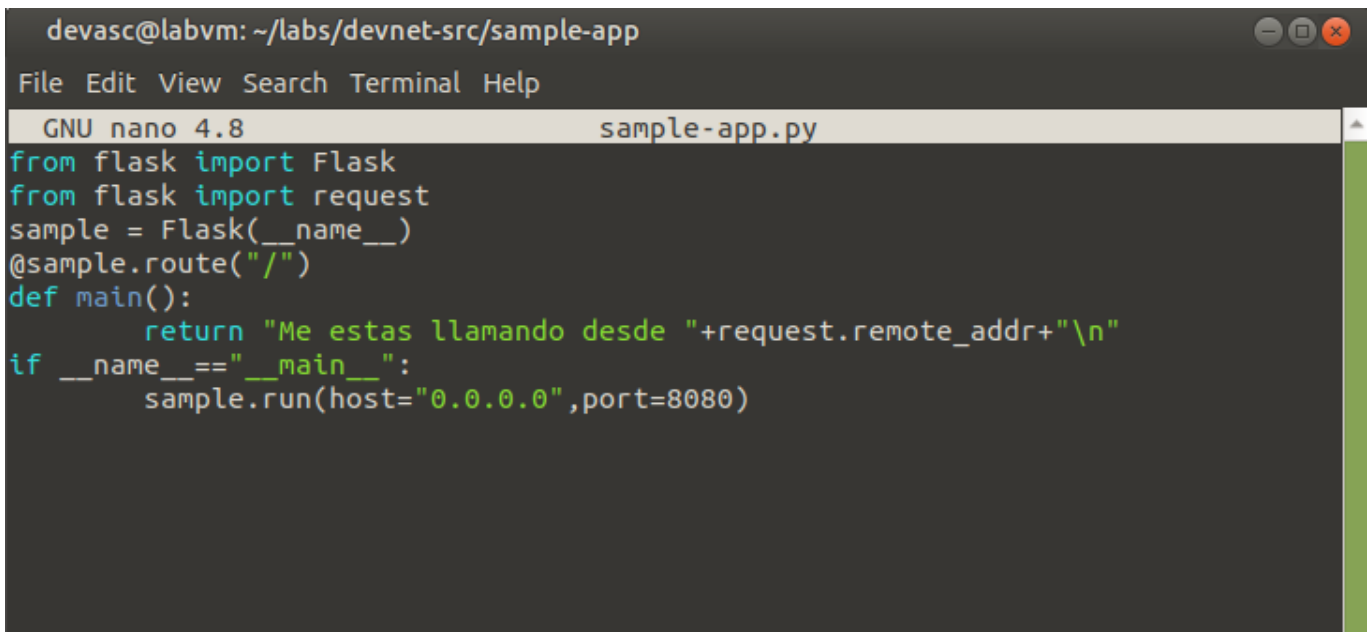
Parte 3: Crear una aplicación web de muestra

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

- Importamos el framework FLASK


```
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: flask in /home/devasc/.local/lib/python3.8/site-packages (1.1.2)
Requirement already satisfied: Werkzeug>=0.15 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (1.0.1)
Requirement already satisfied: Jinja2>=2.10.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
Requirement already satisfied: itsdangerous>=0.24 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (1.1.0)
Requirement already satisfied: click>=5.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /home/devasc/.local/lib/python3.8/site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
devasc@labvm:~/labs/devnet-src/sample-app$
```

- Abrimos el archivo **sample_app.py** y mediante el editor de texto **nano** importamos los métodos necesarios para la biblioteca **flask**.
- Creamos una instancia de la clase Flask, y le asignamos a la variable **sample**, el argumento **__name__** indica el nombre del módulo actual, Flask utiliza este argumento para saber dónde están ubicados los recursos del proyecto, como las plantillas y los archivos estáticos.
- Definimos una ruta y una función de vista, así cuando un usuario visite la página predeterminada nos muestre un mensaje con la dirección IP del cliente.
@sample.route("/") se usará para asociar una URL a una función en Python, en este caso la URL asociada es la ruta principal o raíz ("/"). Cuando un usuario accede a la URL especificada, FLASK ejecutará la función asociada y nos retornará un mensaje escrito en la función main.
- En la función main, **request.remote_addr** es un atributo de la biblioteca Flask que se usará para obtener la dirección IP del cliente que realizó una solicitud a la aplicación. Este atributo nos devuelve la dirección IP del cliente que hizo la solicitud HTTP actual en formato de cadena de texto.
- **sample.run()** es una llamada al método run() de la instancia de la aplicación Flask, que inicia el servidor web y comienza a escuchar las solicitudes entrantes.

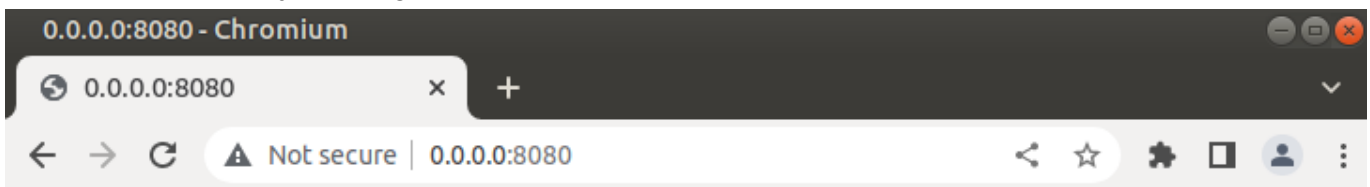


```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 sample-app.py
from flask import Flask
from flask import request
sample = Flask(__name__)
@sample.route("/")
def main():
    return "Me estas llamando desde "+request.remote_addr+"\n"
if __name__=="__main__":
    sample.run(host="0.0.0.0",port=8080)
```

- Guardamos y ejecutamos el script **sample-app.py** veremos que nos aparece SERVICIO de FLASK “sample-app” se está ejecutando. Pero también nos aparece **LAZY LOADING**, eso significa que Flask solo cargará los módulos de la aplicación que son necesarios para manejar la solicitud.


```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ nano sample-app.py
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 deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

- Para comprobar que el servidor se esta ejecutando hay dos maneras:
 1. Mediante el navegador web, introducimos 0.0.0.0:8080 en el campo URL, y nos saldra el mensaje que ingresamos más la dirección IP del cliente que realizo la solicitud.



Me estas llamando desde 127.0.0.1

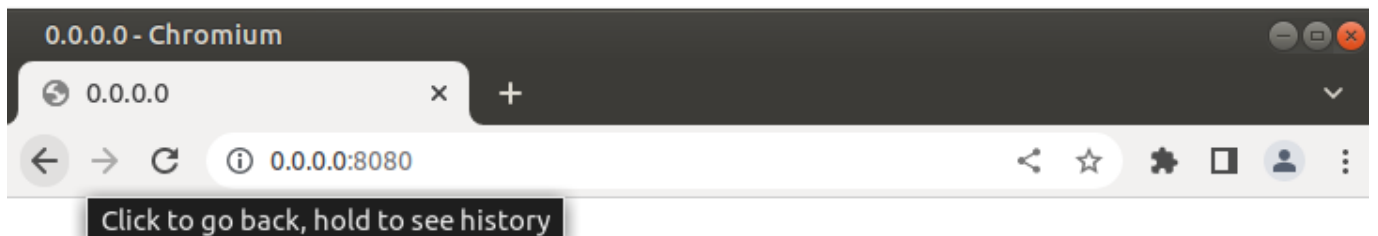
2. Otra forma es usando la TERMINAL, ingresamos cURL que nos servirá para transferir datos entre servidores, en este caso lo usaremos para realizar solicitudes HTTP, **curl http://0.0.0.0:8080**.

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
Me estas llamando desde 127.0.0.1
devasc@labvm:~/labs/devnet-src/sample-app$
```

- Detenemos el servidor, en la terminal donde ejecutamos **sample-app.py**, pulsando **ctrl+c**.

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ nano sample-app.py
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 deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [06/May/2023 00:08:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 00:08:46] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [06/May/2023 00:09:51] "GET / HTTP/1.1" 200 -
```

- Una vez cerrado el servidor, si hacemos otra consulta nos saldrá error.



This site can't be reached

0.0.0.0 refused to connect.

Try:

- Checking the connection
- [Checking the proxy and the firewall](#)

ERR_CONNECTION_REFUSED

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
Me estas llamando desde 127.0.0.1
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
curl: (7) Failed to connect to 0.0.0.0 port 8080: Connection refused
devasc@labvm:~/labs/devnet-src/sample-app$
```

Parte 4: Configurar la aplicación web para utilizar archivos de sitio web

En esta parte, construiremos la aplicación web de ejemplo para incluir una página **index.html** y una especificación **style.css**. El **index.html** es normalmente la primera página cargada en el navegador web de un cliente al visitar su sitio web. El **style.css** es una hoja de estilo utilizada para personalizar el aspecto de la página web.

- Abrimos el **index.html** y **style.css**.
- Notamos que en el **index.html**, mandamos a imprimir en el título dentro de la página **h1**, pero antes de ello en el archivo python importamos **render_template**, esta función proporcionada por Flask nos permite renderizar una plantilla HTML, esta función toma como parametro el nombre del archivo de la plantilla.
- En la función **main** editamos el return, ahí le decimos que nos retorne pasandole como parámetro el **index.html**.
- Ejecutamos el **sample_app.py**, y nos aparecerá igual a lo hecho en la **parte3**, con la diferencia de que esta vez le mandamos con una estructura HTML y un estilo CSS.

```
<> index.html x # style.css
home > devasc > labs > devnet-src > sample-app > templates > <> index.html > html > head > link
1 <html>
2 <head>
3   <title>Sample app</title>
4   <link rel="stylesheet" href="/static/style.css" />
5 </head>
6 <body>
7   <h1>You are calling me from {{request.remote_addr}}</h1>
8 </body>
9 </html>
10
```

```
<> index.html # style.css
home > devasc > labs > devnet-src > sample-app > static > # style.css > ...
1 body {
2   background: lightsteelblue;
3 }
4
```

```
index.html # style.css sample_app.py X
home > devasc > labs > devnet-src > sample-app > sample_app.py
1 # Add to this file for the sample app lab
2 from flask import Flask
3 from flask import request
4 from flask import render_template
5
6 sample = Flask(__name__)
7
8 @sample.route("/")
9 def main():
10     #return "Me estas llamando desde "+request.remote_addr+"\n"
11     return render_template(["index.html"])
12 if __name__=="__main__":
13     sample.run(host="0.0.0.0",port=8080)
14
15
```

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
<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$
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 depl
ent.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [06/May/2023 16:13:42] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 16:13:42] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 16:13:42] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [06/May/2023 16:14:37] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 16:14:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 16:14:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/May/2023 16:16:56] "GET / HTTP/1.1" 200 -
```



```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
<html>
<head>
  <title>Sample app</title>
  <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
  <h1>You are calling me from 127.0.0.1</h1>
</body>
</html>devasc@labvm:~/labs/devnet-src/sample-app$
```

- Detenemos el servidor, en la terminal donde ejecutamos **sample-app.py**, pulsando **ctrl+c**.

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

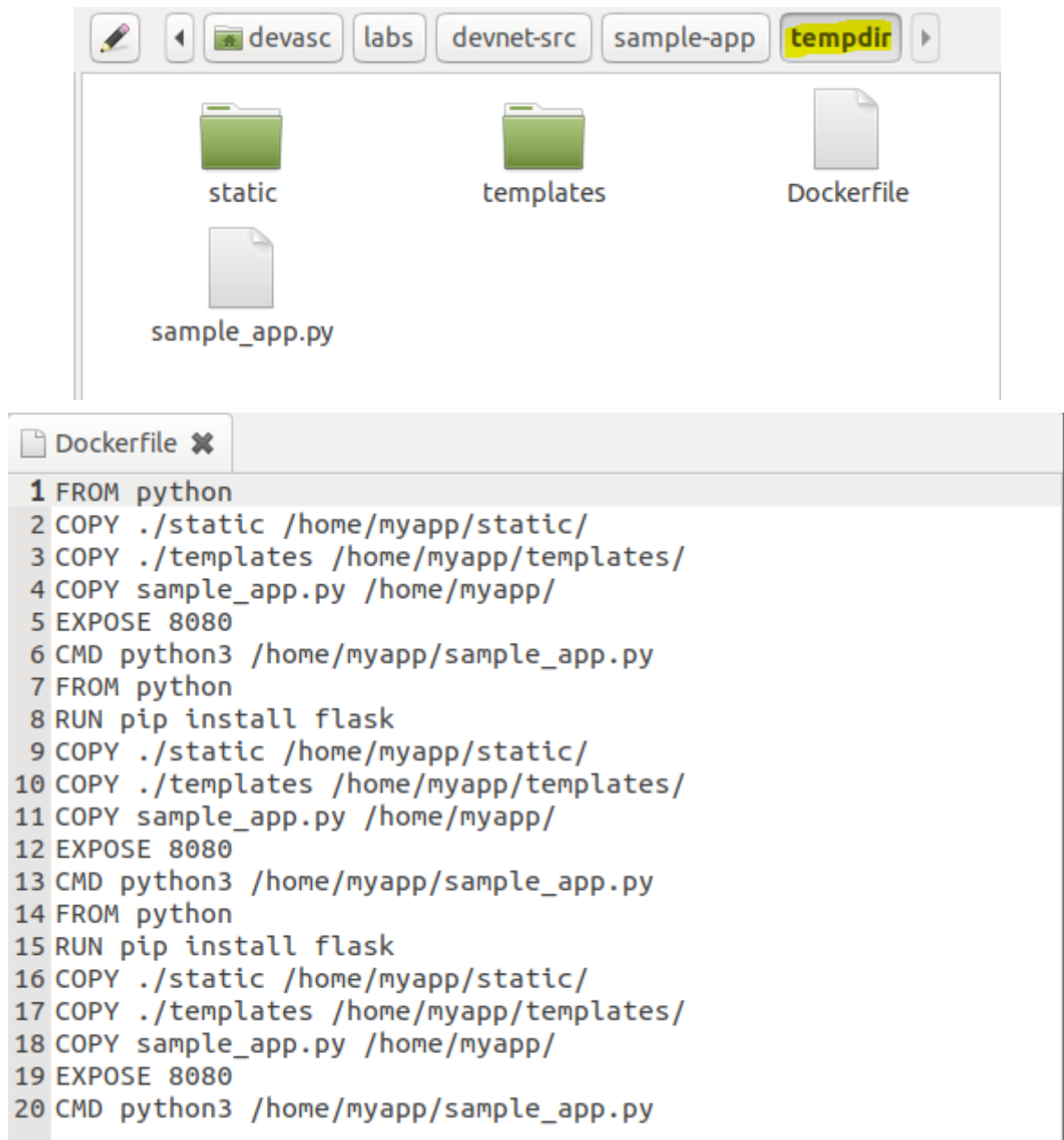
Una aplicación se puede implementar en un servidor bare metal (servidor físico dedicado a un entorno de inquilino único) o en una máquina virtual, como acaba de hacer en la parte anterior. También se puede implementar en una solución contenerizada como Docker.

En esta parte, crearemos un script bash y le agregaremos comandos que completen las siguientes tareas para crear y ejecutar un contenedor Docker.

- Crearemos directorios temporales para almacenar los archivos del sitio web, agregaremos **she-bang**, crearemos un directorio llamado **tempdir** para eso se usa el comando **mkdir**. luego creamos directorios dentro del directorio tempdir, **mkdir tempdir/templates**

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

- Ahora copiaremos los directorios del sitio web y sample_app.py en el directorio temporal que creamos (**tempdir**), mediante el comando **cp**.
- Luego agregamos la cadena de texto "**FROM python**" al archivo DockerFile, que se encuentra en el directorio tempdir, utilizando el operador redireccion ">>".



- la instrucción **cd tempdir** cambia el directorio de trabajo actual a **tempdir**.
- “**docker build -t sampleapp .**”, creamos el contenedor Docker, construye una imagen docker a partir del archivo Dockerfile presente en el directorio actual (tempdir) y le asigna el nombre "sampleapp". El punto "." nos especifica que el contexto de construcción se encuentra en el directorio actual.
- **docker run -t -d -p 8080:8080 --name samplerunning sampleapp**, Este comando ejecuta la imagen sampleapp en un contenedor Docker en segundo plano (**-d**), asignando el puerto 8080 del host al puerto 8080 del contenedor (**-p 8080:8080**) y le da al contenedor un nombre (**--name samplerunning**). **-t** asigna una pseudo terminal al contenedor. Esto permite interactuar con el contenedor a través de la línea de comando.
- Y al final agregamos el comando **docker ps -a** para mostrar todos los contenedores Docker que se están ejecutando actualmente. Este comando será el último ejecutado por el script bash.


```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 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

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

- Guardamos el script bash.

Parte 6: Construir, ejecutar y verificar el contenedor Docker

- Ejecutar el script bash, en el paso 7/7 se ejecuta sample_app.py que crea el servidor web, al final observamos el ID del contenedor, la imagen Docker sampleapp.

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS		PORTS	NAMES
bff5987d8eeb	sampleapp	"/bin/sh -c 'python3..."	23 hours ago
Exited (255) 10 minutes ago		0.0.0.0:8080->8080/tcp	samplerunning

```
devasc@labvm:~/labs/devnet-src/sample-app$
```

- ejecutando.

```

devasc@labvm:~/labs/devnet-src/sample-app$ bash sample-app.sh
mkdir: cannot create directory 'tmpdir': File exists
mkdir: cannot create directory 'tmpdir/templates': File exists
mkdir: cannot create directory 'tmpdir/static': File exists
Sending build context to Docker daemon 6.144kB
Step 1/7 : FROM python
--> 815c8c75dfc0
Step 2/7 : RUN pip install flask
--> Using cache
--> 4b46416de15b
Step 3/7 : COPY ./static /home/myapp/static/
--> Using cache
--> 240bcf74d0bd
Step 4/7 : COPY ./templates /home/myapp/templates/
--> Using cache
--> 23579d62103a
Step 5/7 : COPY sample_app.py /home/myapp/
--> Using cache
--> ff3a43e4367c
Step 6/7 : EXPOSE 8080
--> Using cache
--> cc6e250d8cb0
Step 7/7 : CMD python3 /home/myapp/sample_app.py
--> Using cache
--> 2365ec36406a
Successfully built 2365ec36406a
Successfully tagged sampleapp:latest

```

```

docker: Error response from daemon: Conflict. The container name "/samplerunning" is already in use by container "bff5987d8eeba125e6f884be5c24c5fbe56f40e26f559fc16dc3eb5a6c5402b1". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.

```

CONTAINER ID	IMAGE	COMMAND	CREATED
bff5987d8eeb	sampleapp	"/bin/sh -c 'python3..."	23 hours ago
Exited (255) 10 minutes ago		0.0.0.0:8080->8080/tcp	samplerunning

```

devasc@labvm:~/labs/devnet-src/sample-app$

```

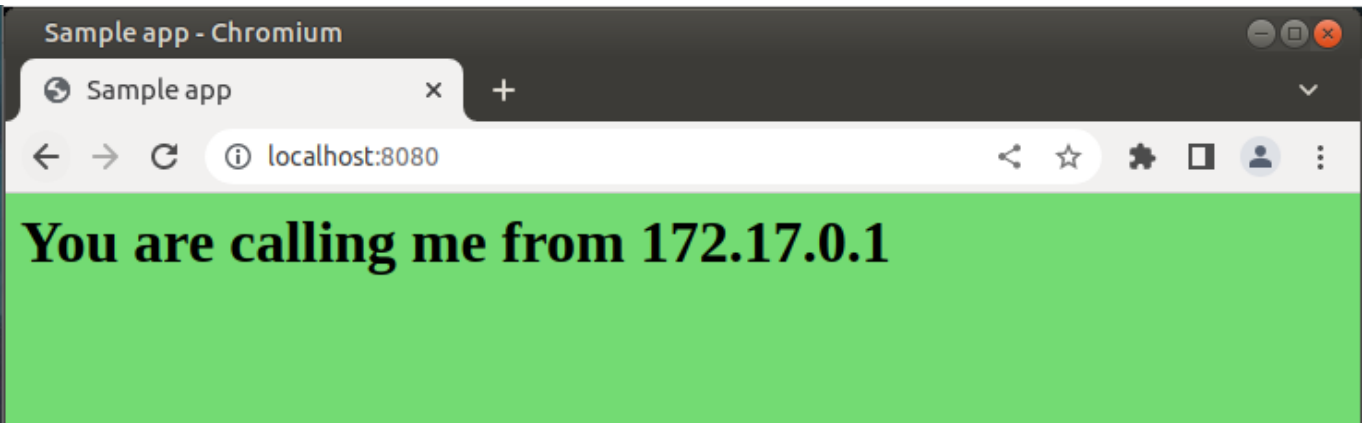
- Verificamos el directorio tmpdir y accedemos al archivo Dockerfile, abrimos y vemos como esta en forma final sin los comandos echo

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

- La salida del comando **docker ps -a** donde vemos el ID, la imagen, el status... lo podemos redirigir a un archivo de texto para mejor visualización usando la siguiente indicación:
docker ps -a >> running.txt

1	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2	bff5987d8eeb	sampleapp	"/bin/sh -c 'python3..."	23 hours ago	Exited (255)	15 minutes ago 0.0.0.0:8080->8080/tcp	samplerunning

- El contenedor Docker creara su propia dirección IP a partir de un espacio de direcciones de red privada. Verificaremos 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. También podemos usar el comando **curl**, si lo desea.



```
devasc@labvm:~/labs/devnet-src/sample-app$ 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:~/labs/devnet-src/sample-app$
```

- De forma predeterminada, Docker utiliza la subred **IPv4 172.17.0.0/16** para redes de contenedores. (Esta dirección se puede cambiar si es necesario.) Introduzca el comando **ip address** para mostrar todas las direcciones IP utilizadas por la instancia de la VM DEVASC. Debería ver la dirección de bucle invertido (loopback) 127.0.0.1 que usó la aplicación web anteriormente en el laboratorio y la nueva interfaz Docker con la dirección IP 172.17.0.1.

```
devasc@labvm:~/labs/devnet-src/sample-app$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:e9:3d:e6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 83410sec preferred_lft 83410sec
    inet6 fe80::a00:27ff:fee9:3de6/64 scope link
        valid_lft forever preferred_lft forever
3: dummy0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether fe:b6:50:40:1e:f8 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/32 scope global dummy0
        valid_lft forever preferred_lft forever
    inet 192.0.2.2/32 scope global dummy0
        valid_lft forever preferred_lft forever
    inet 192.0.2.3/32 scope global dummy0
        valid_lft forever preferred_lft forever
    inet 192.0.2.4/32 scope global dummy0
        valid_lft forever preferred_lft forever
    inet 192.0.2.5/32 scope global dummy0
        valid_lft forever preferred_lft forever
    inet6 fe80::fcb6:50ff:fe40:1ef8/64 scope link
        valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:35:32:8b:4c brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
devasc@labvm:~/labs/devnet-src/sample-app$
```

- Para acceder al **contenedor en ejecución**, introduzca el comando **docker exec -it** especificando el nombre del contenedor en ejecución (**samplerunning**) y que desea un shell bash (**/bin/bash**). La opción **-i** especifica que desea que sea interactivo y la opción **-t** especifica que desea acceder a la terminal. El prompt cambiara a **root @containerID**. Observamos que el ID del contenedor coincide con el ID mostrado en la salida de **docker ps -a**.
- Ahora estaremos en la raíz para el contenedor DOCKER de samplerunning, desde aqui podemos usar comandos linux.
- Como copiamos los directorios en dockfile, entonces podemos verlos en **home/myapp** , y veremos lo que hemos copiado, salimos escribiendo **exit**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@bff5987d8eeb:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@bff5987d8eeb:/# ls home/myapp/
sample_app.py  static  templates
root@bff5987d8eeb:/# exit
exit
devasc@labvm:~/labs/devnet-src/sample-app$
```

- Detenemos y retiramos el contenedor DOCKER.
- para eliminar permanentemente el contenedor usamos docker rm samplerunning, pero podemos reconstruir de nuevo ejecutando el programa sample-app.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
9672ddaa46bd       sampleapp          "/bin/sh -c 'python3..." 30 minutes ago
Created                                     tender_keldysh
03f6c04a2c27       sampleapp          "/bin/sh -c 'python3..." 43 minutes ago
Exited (255) 16 minutes ago 0.0.0.0:8080->8080/tcp    dazzling_lehmann
bff5987d8eeb       sampleapp          "/bin/sh -c 'python3..." 24 hours ago
Exited (137) 3 seconds ago    samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker rm samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
9672ddaa46bd       sampleapp          "/bin/sh -c 'python3..." 30 minutes ago
Created                                     tender_keldysh
03f6c04a2c27       sampleapp          "/bin/sh -c 'python3..." 43 minutes ago
Exited (255) 16 minutes ago 0.0.0.0:8080->8080/tcp    dazzling_lehmann
devasc@labvm:~/labs/devnet-src/sample-app$
```

Conclusiones: (en cada paso escribia un resumen de lo que hacia y algunos conceptos pequeños)

En la parte 2, aprendimos a crear un script bash **.sh**, eso para realizar tareas como el ejemplo de mandar un mensaje como tambien manipular archivos y directorios que lo veremos mas adelante.

En la parte 3, creamos un script en python que nos muestre la direccion IP del cliente cuando visitamos una pagina web, para ello importamos request del framework flask, esto nos ayudara a hacer un solicitud http, y corroboramos en la web chromium par obtener la salida de donde lo estamos llamando.

En la parte 4, Aqui le dimos un poco de estilo a la pag web, pero python puede ser un poco tedioso , para ello lo renderizamos usando render_template.

En la parte 5, aqui aplicamos lo visto en la parte2 para compilar y ejecutar un contenedor DOCKER, por eso aqui crearemos un script bash para poder crear un directorio temporal usando el comando mkdir al directorio le pusimo tempdir, depues copiamos todo el directorio del sitio web a tempdir, creamos tambien un archivo dockerfile en este construiremos el contenedor, usamos el comando EXPOSE para exponer el puerto 8080 para poder usarlo. con el comando docker build especificamos el nombre del contenedor, y lo guardamos.

para ver todos los contenedores docker que se ejecutan usamos docker ps -a.
y para iniciar el docker usamos docker start samplerunning.

En la parte 6, vi como podemos ejecutar y verificar el contenedor docker, una vez ejecutado cuando hago una consulta en chromium me aparece la direccion ip del contenedor docker. Tambien cuando hago una consulta con ip address veremos una nueva interfaz DOCKER con la direccion de este.Podemos acceder tambien shell del contenedor y ahi adentro podremos ver todo lo copiado, al final tenemos que cerrar con `docker stop samplerunning` y si queremos eliminar temporalmente lo haremos con `docker rm samplerunning`, pero podemos reconstruirlo ejecutando de nuevo el `sample-app`.