



Lab - Build a Sample Web App in a Docker Container

Objectives

- Part 1: Launch the DEVASC VM
- Part 2: Create a Simple Bash Script
- Part 3: Create a Sample Web App
- Part 4: Configure the Web App to Use Website Files
- Part 5: Create a Bash Script to Build and Run a Docker Container
- Part 6: Build, Run, and Verify the Docker Container

Background / Scenario

In this lab, you will review basic bash scripting techniques because bash scripting is a prerequisite for the rest of the lab. You will then build and modify a Python script for a simple web application. Next, you will create a bash script to automate the process for creating a Dockerfile, building the Docker container, and running the Docker container. Finally, you will use `docker` commands to investigate the intricacies of the Docker container instance.

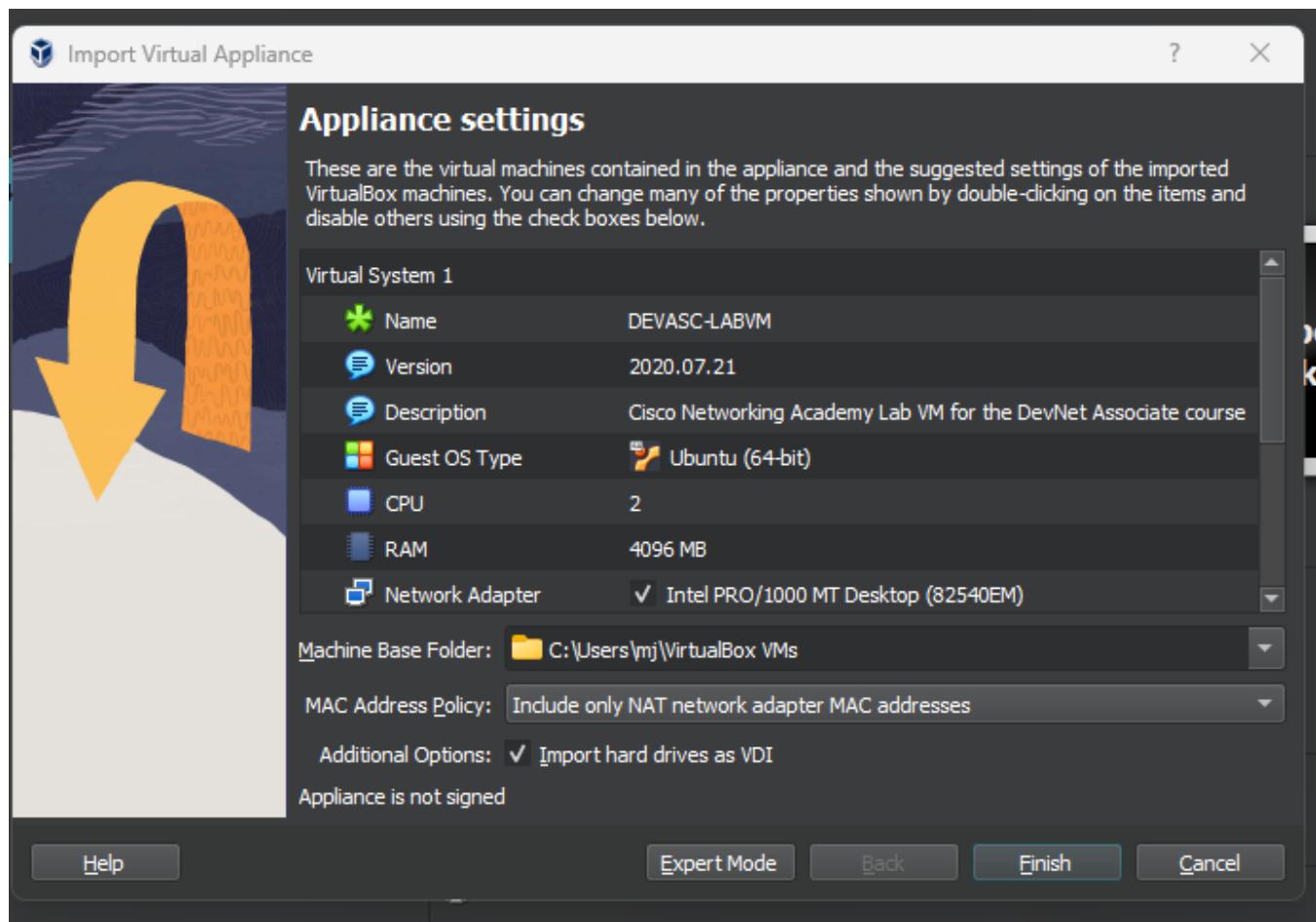
Required Resources

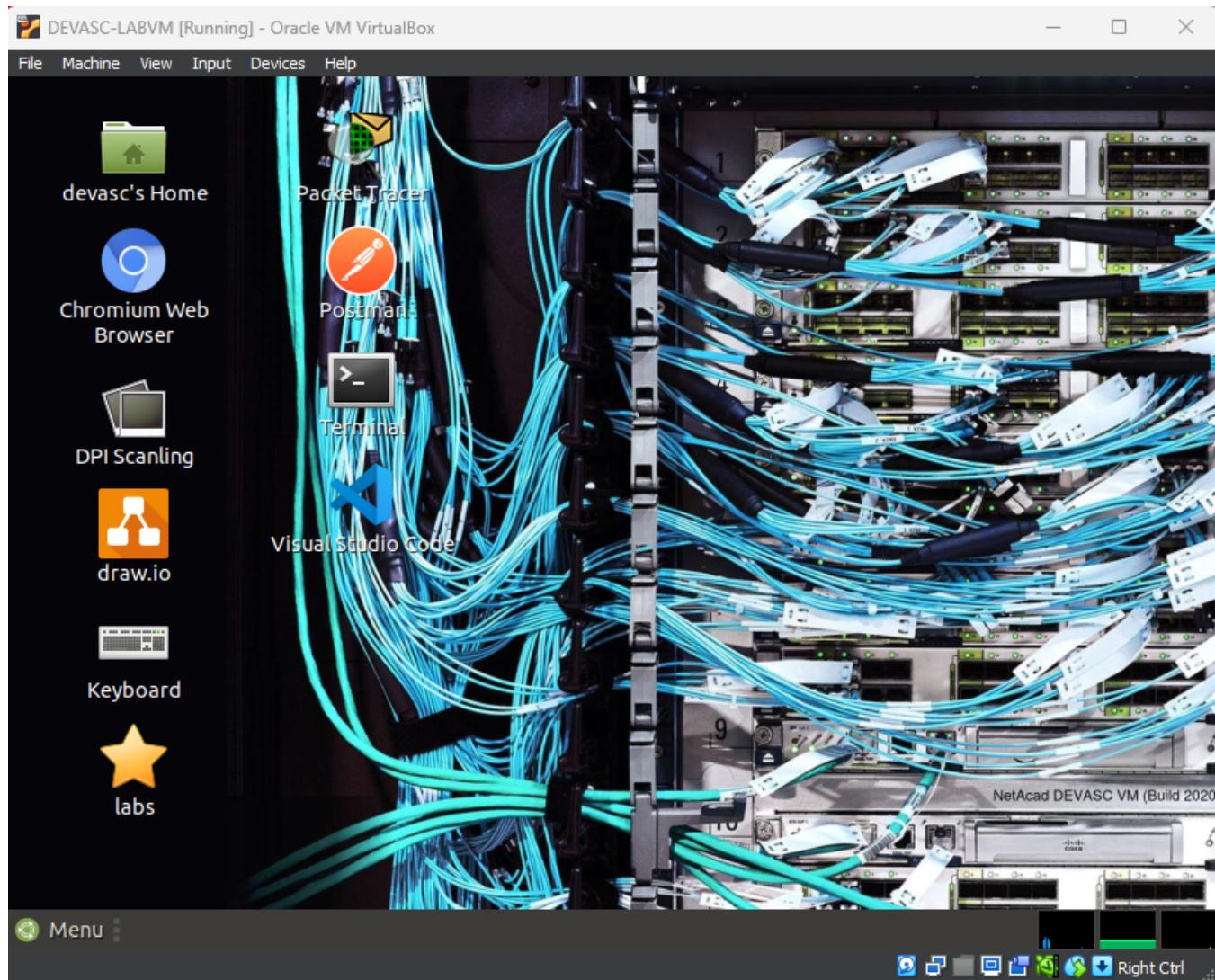
- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the DEVASC VM

If you have not already completed the [Lab - Install the Virtual Machine Lab Environment](#), do so now. If you have already completed that lab, launch the DEVASC VM now.





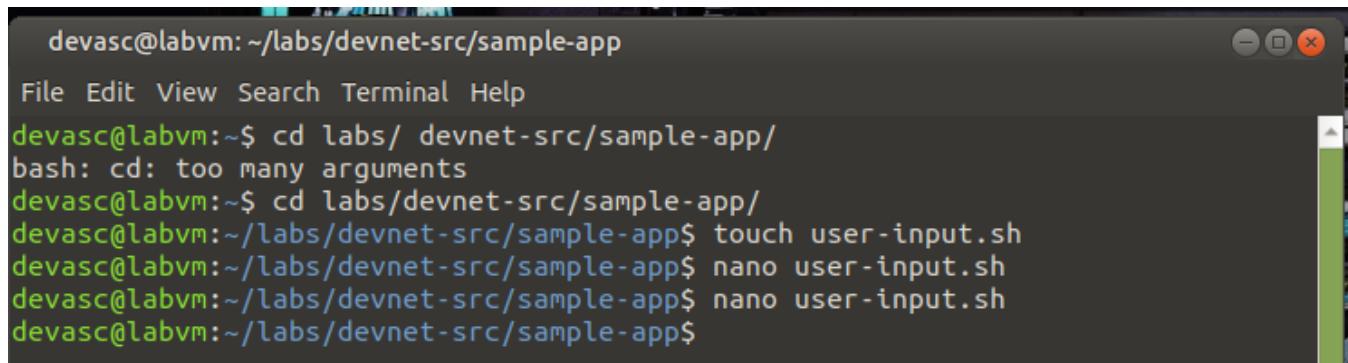
Part 2: Create a Simple Bash Script

Bash knowledge is crucial for working with continuous integration, continuous deployment, containers, and with your development environment. Bash scripts help programmers automate a variety of tasks in one script file. In this part, you will briefly review how to create a bash script. Later in the lab, you will use a bash script to automate the creation of a web app inside of a Docker container.

Step 1: Create an empty bash script file.

Change your working directory to `~/labs/devnet-src/sample-app` and add a new file called `user-input.sh`.

```
devasc@labvm:~$ cd labs/devnet-src/sample-app/  
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
```

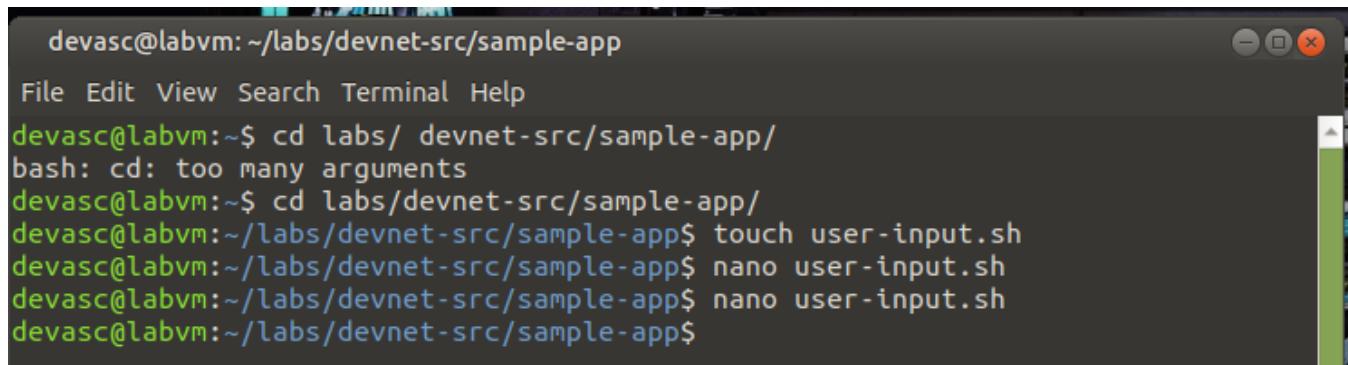


```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd labs/ devnet-src/sample-app/
bash: cd: too many arguments
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$
```

Step 2: Open the file in the nano text editor.

Use the **nano** command to open the nano text editor.

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



```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd labs/ devnet-src/sample-app/
bash: cd: too many arguments
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
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$
```

Step 3: Add the ‘she-bang’ to the top of the script.

From here you can enter commands for your bash script. Use the arrow keys to navigate in **nano**. Notice the commands at the bottom (not shown here) for managing the file. The caret symbol (^) indicates that you use the CTRL or Command key on your keyboard. For example, to exit **nano**, type CTRL+X.

Add the ‘she-bang’ which tells the system that this file includes commands that need to be run in the bash shell.

```
#!/bin/bash
```

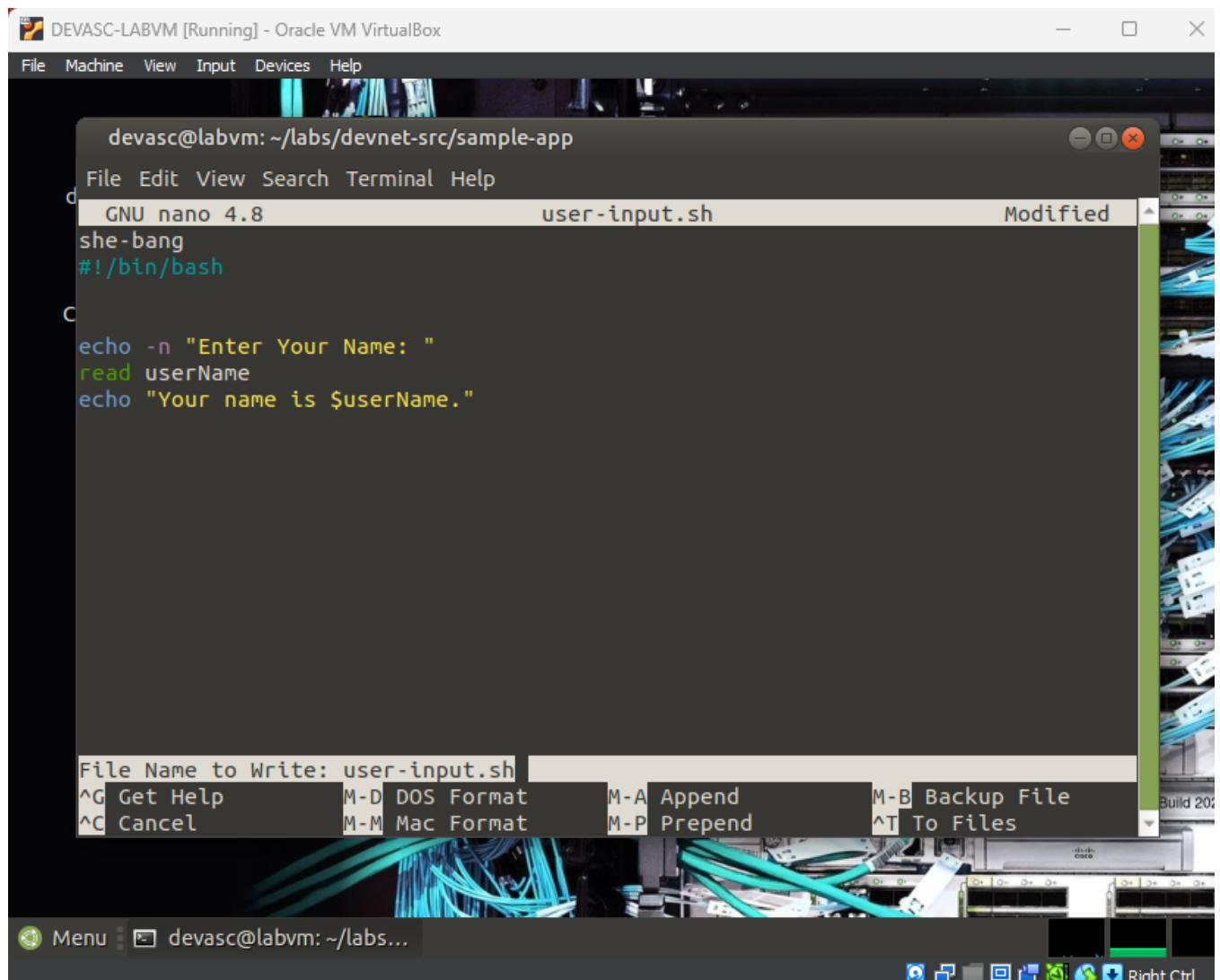
Note: You can use a graphical text editor or open the file with VS Code. However, you should be familiar with command-line text editors like **nano** and **vim**. Search the internet for tutorials to refresh your skill or learn more about them.

Step 4: Add simple bash commands to the script.

Enter some simple bash commands for your script. The following commands will ask the user for a name, set the name to a variable called **userName**, and display a string of text with the user’s name.

```
echo -n "Enter Your Name: "
read userName
echo "Your name is $userName."
```

Lab - Build a Sample Web App in a Docker Container



Step 5: Exit nano and save your script.

Press **CTRL+X**, then **Y**, then **ENTER** to exit **nano** and save your script.

Step 6: Run your script from the command line.

You can run it directly from the command line using the following command.

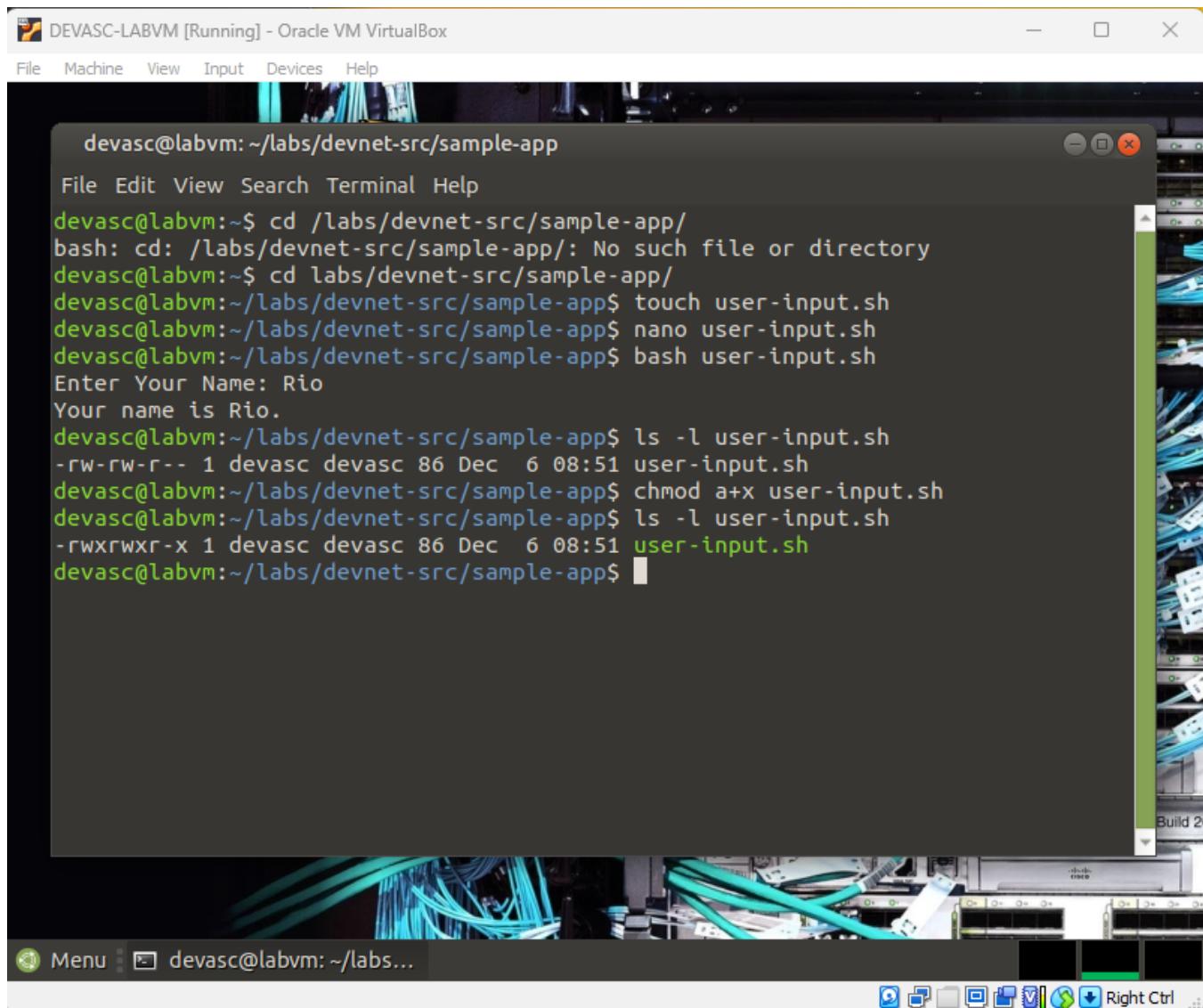
```
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Enter Your Name: Bob
Your name is Bob.
devasc@labvm:~/labs/devnet-src/sample-app$
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
user-input.sh: line 1: she-bang: command not found
Enter Your Name: Rio
Your name is Rio.
devasc@labvm:~/labs/devnet-src/sample-app$
```

Step 7: Change the mode of the script to an executable file for all users.

Change the mode of the script to an executable using the **chmod** command. Set the options to **a+x** to make the script executable (x) by all users (a). After using **chmod**, notice permissions have been modified for users, groups, and others to include the "x" (executable).

```
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh  
-rw-rw-r-- 1 devasc devasc 84 Jun  7 16:43 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 84 Jun  7 16:43 user-input.sh
```

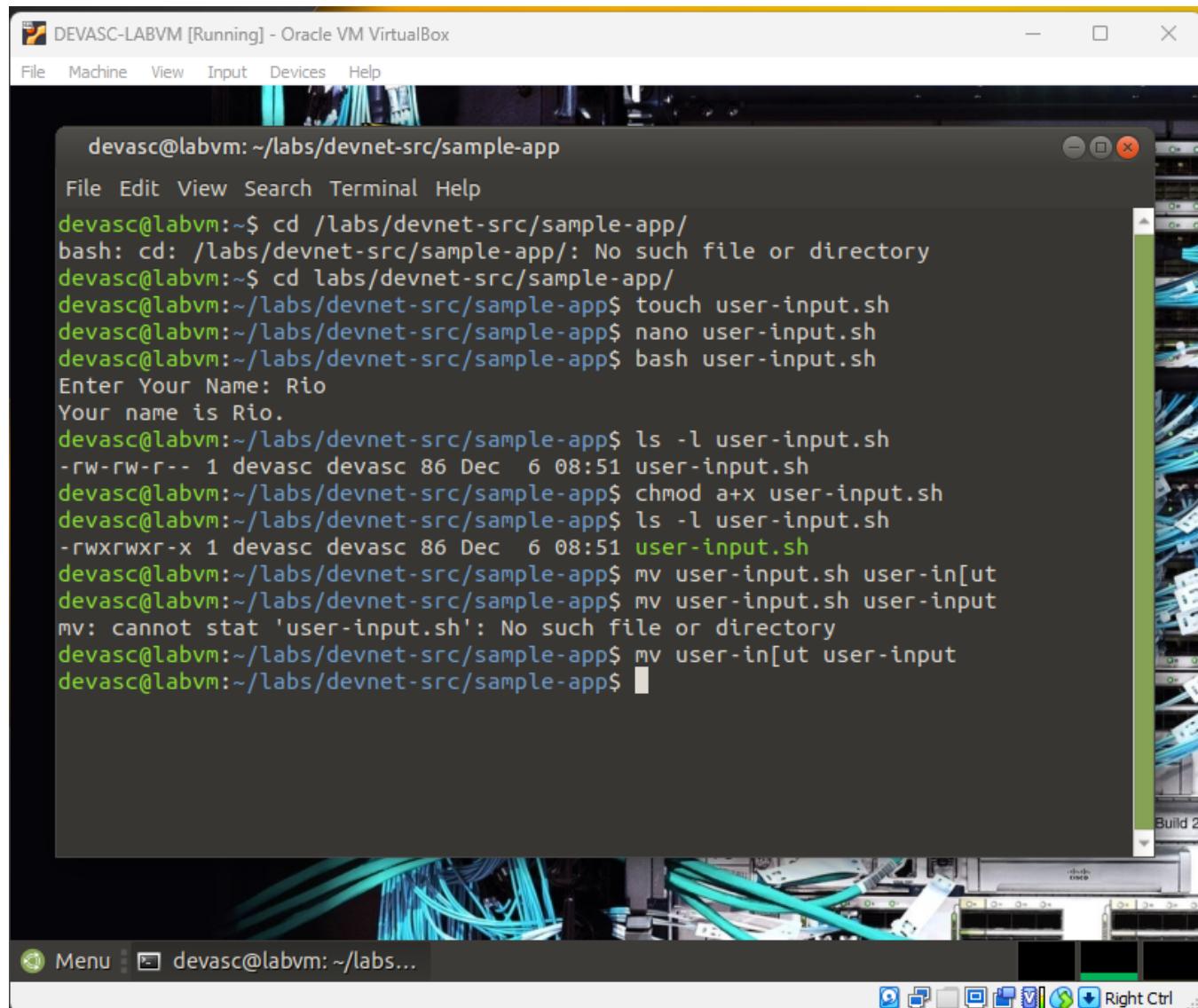


Step 8: Rename the file to remove the .sh extension.

You can rename the file to remove the extension so that users do not have to add .sh to the command to execute the script.

Lab - Build a Sample Web App in a Docker Container

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



The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal is running a bash shell on a host system named "devasc@labvm". The user has navigated to the directory "/labs/devnet-src/sample-app". They run several commands to create a file named "user-input.sh", edit it with nano, and then run it with bash. The script prompts for a name ("Enter Your Name: Rio") and prints it back ("Your name is Rio"). It then lists the file with ls, changes its permissions with chmod, lists it again, and attempts to move it to "user-in[ut" (likely a typo for "user-input"), which fails because the file does not exist. Finally, they try to move it to "user-input" again, which also fails due to a permission error.

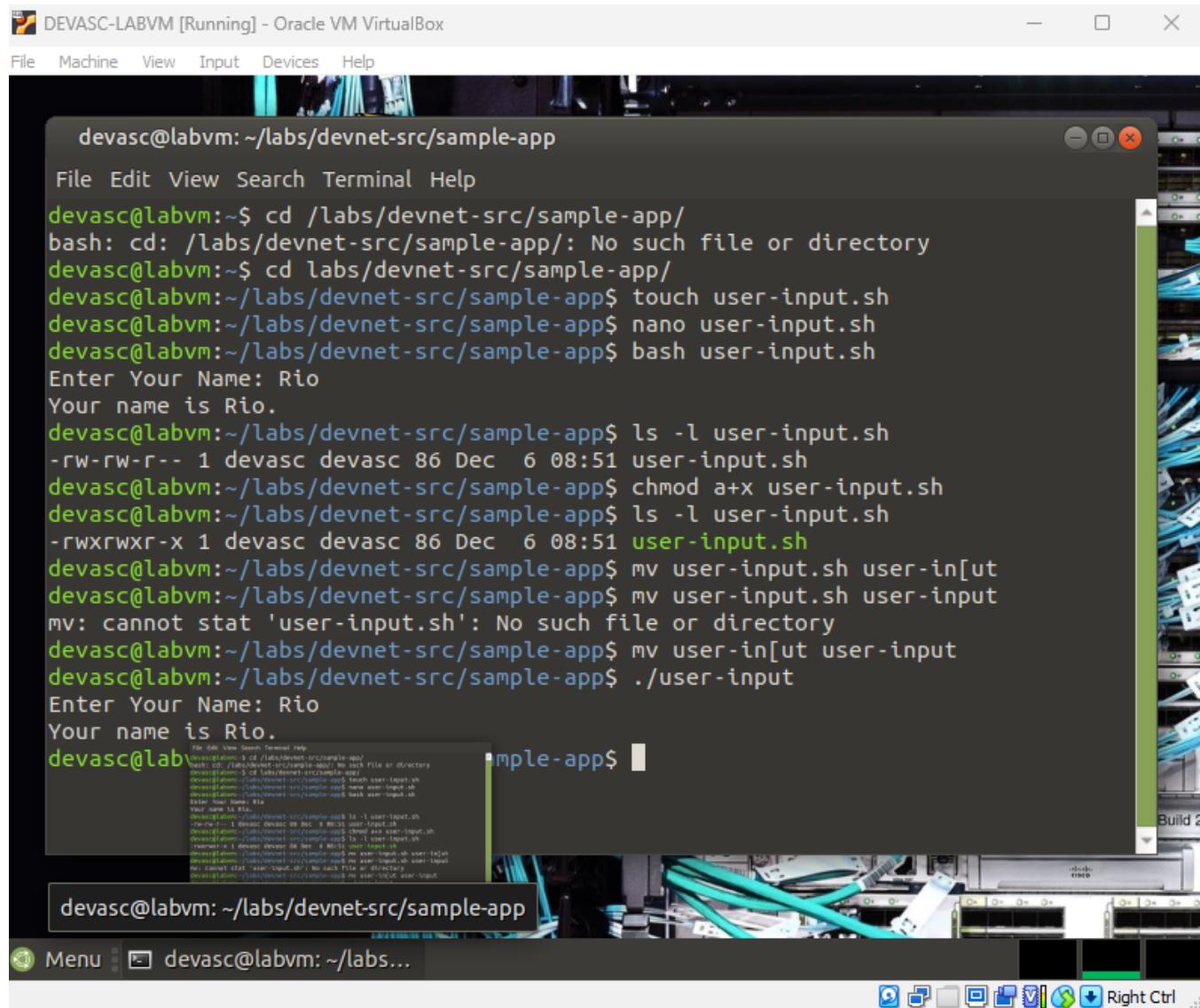
```
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
devasc@labvm:~$ cd /labs/devnet-src/sample-app/
bash: cd: /labs/devnet-src/sample-app/: No such file or directory
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Enter Your Name: Rio
Your name is Rio.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 86 Dec  6 08:51 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 86 Dec  6 08:51 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-in[ut
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
mv: cannot stat 'user-input.sh': No such file or directory
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-in[ut user-input
devasc@labvm:~/labs/devnet-src/sample-app$
```

Step 9: Execute the script from the command line.

Now the script can be run from the command line without the **source** command or an extension. To run a bash script without the source command, you must preface the script with "./".

```
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Enter Your Name: Bob
Your name is Bob.
devasc@labvm:~/labs/devnet-src/sample-app$
```

Lab - Build a Sample Web App in a Docker Container



```
DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd /labs/devnet-src/sample-app/
bash: cd: /labs/devnet-src/sample-app/: No such file or directory
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch 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
Enter Your Name: Rio
Your name is Rio.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 86 Dec 6 08:51 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 86 Dec 6 08:51 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-in[ut
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
mv: cannot stat 'user-input.sh': No such file or directory
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-in[ut user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Enter Your Name: Rio
Your name is Rio.
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd /labs/devnet-src/sample-app/
bash: cd: /labs/devnet-src/sample-app/: No such file or directory
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch 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
Enter Your Name: Rio
Your name is Rio.
devasc@labvm: ~/labs/devnet-src/sample-app
```

Step 10: Investigate other bash scripts.

If you have little or no experience creating bash scripts, take some time to search the internet for bash tutorials, bash examples, and bash games.

Part 3: Create a Sample Web App

Before we can launch an application in a Docker container, we first need to have the app. In this part, you will create a very simple Python script that will display the IP address of the client when the client visits the web page.

Step 1: Install Flask and open a port on the DEVASC VM firewall.

Web application developers using Python typically leverage a framework. A framework is a code library to make it easier for developers to create reliable, scalable and maintainable web applications. Flask is a web application framework written in Python. Other frameworks include Tornado and Pyramid.

Lab - Build a Sample Web App in a Docker Container

You will use this framework to create the sample web app. Flask receives requests and then provides a response to the user in the web app. This is useful for dynamic web applications because it allows user interaction and dynamic content. What makes your sample web app dynamic is that it will be displaying the IP address of the client.

Note: Understanding Flask functions, methods, and libraries are beyond the scope of this course. It is used in this lab to show how quickly you can get a web application up and running. If you want to learn more, search the internet for more information and tutorials on the Flask framework.

Open a terminal window and import **flask**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
Requirement already satisfied: flask in /home/devasc/.local/lib/python3.8/site-packages (1.1.2)
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: Werkzeug>=0.15 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (1.0.1)
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$
```

Step 2: Open the **sample_app.py** file.

Open the **sample_app.py** file located in the **/sample-app** directory. You can do this inside VS Code or you can use a command-line text editor like **nano** or **vim**.

Lab - Build a Sample Web App in a Docker Container

The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal session is as follows:

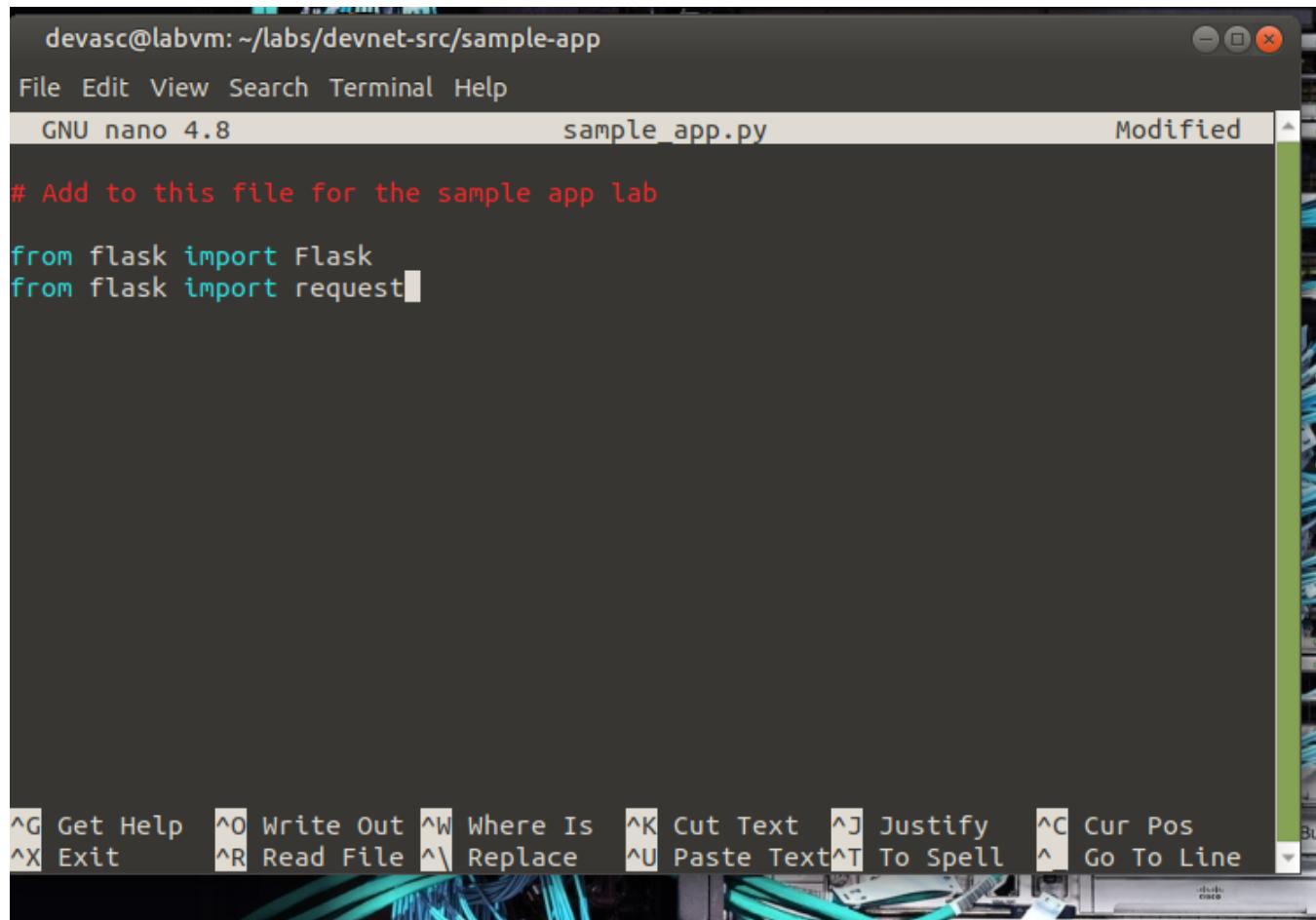
```
devasc@labvm:~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 86 Dec  6 08:51 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-in[ut
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
mv: cannot stat 'user-input.sh': No such file or directory
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-in[ut user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Enter Your Name: Rio
Your name is Rio.
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
Requirement already satisfied: flask in /home/devasc/.local/lib/python3.8/site-packages (1.1.2)
Requirement already satisfied: click>=5.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (7.1.2)
Requirement already satisfied: Werkzeug>=0.15 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (1.0.1)
Requirement already satisfied: itsdangerous>=0.24 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (1.1.0)
Requirement already satisfied: Jinja2>=2.10.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
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$ nano sample_app.py
```

The terminal window is running on an Ubuntu 20.04 LTS host. The desktop environment visible in the background is Oracle VM VirtualBox.

Step 3: Add the commands to import methods from flask.

Add the following commands to import the required methods from the flask library.

```
from flask import Flask
from flask import request
```



The screenshot shows a terminal window titled "sample_app.py" in the nano 4.8 editor. The file contains the following Python code:

```
# Add to this file for the sample app lab
from flask import Flask
from flask import request
```

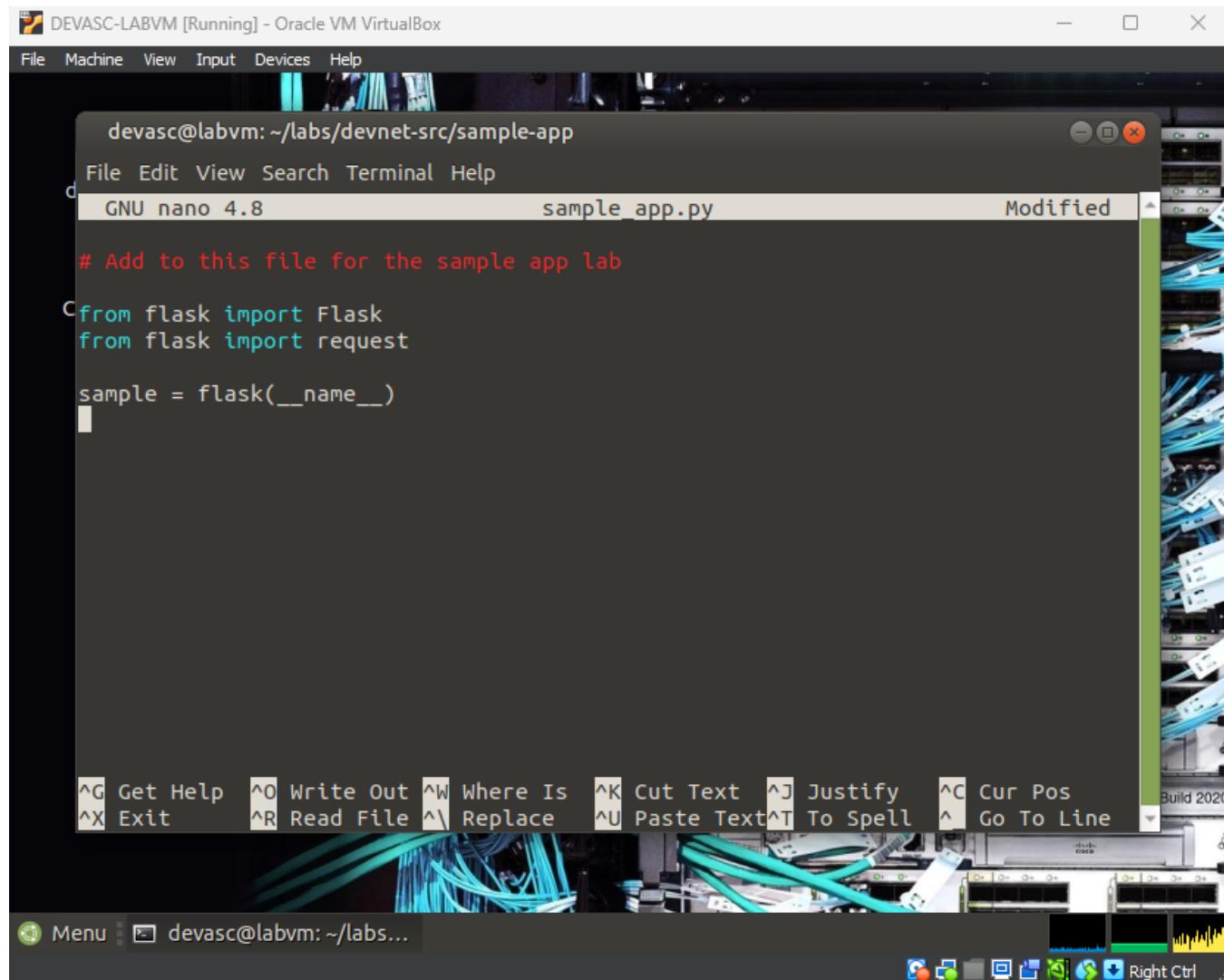
The terminal window has a dark background with light-colored text. The bottom of the window displays a menu bar with options like File, Edit, View, Search, Terminal, Help, and a modified status indicator. A status bar at the bottom shows keyboard shortcuts for various functions.

Step 4: Create an instance of the Flask class.

Create an instance of the Flask class and name it **sample**. Be sure to use two underscores before and after the "name".

```
sample = Flask(__name__)
```

Lab - Build a Sample Web App in a Docker Container



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

sample = flask(__name__)

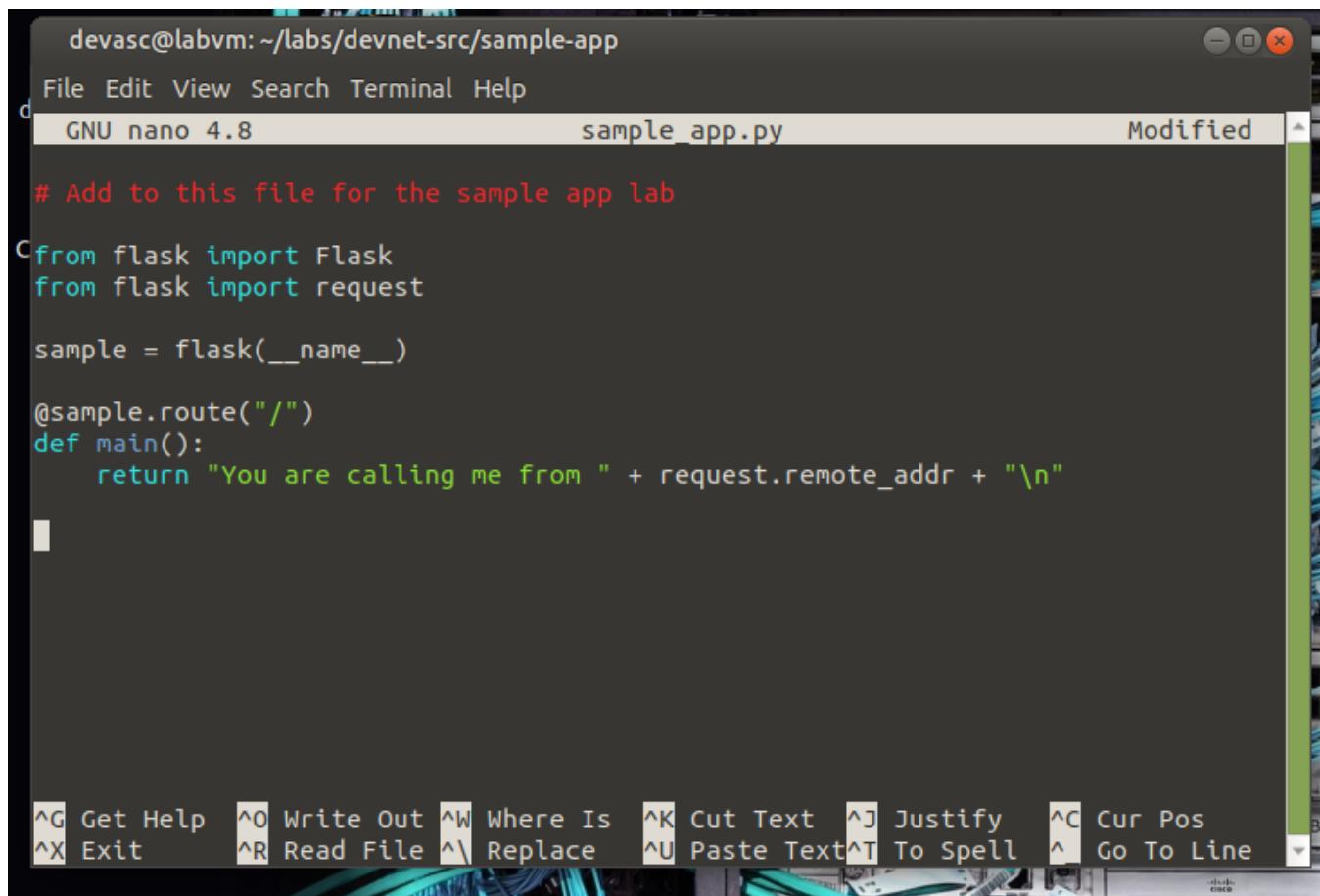
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
Build 2020

devasc@labvm: ~/labs...
```

Step 5: Define a method to display the client IP address.

Next, configure Flask so that when a user visits the default page (root directory), it displays a message with the IP address of the client.

```
@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"
```



The screenshot shows a terminal window titled "devasc@labvm: ~/labs/devnet-src/sample-app". The window contains a file named "sample_app.py" which is being edited with the nano 4.8 text editor. The code in the file is:

```
# Add to this file for the sample app lab
from flask import Flask
from flask import request

sample = flask(__name__)

@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"
```

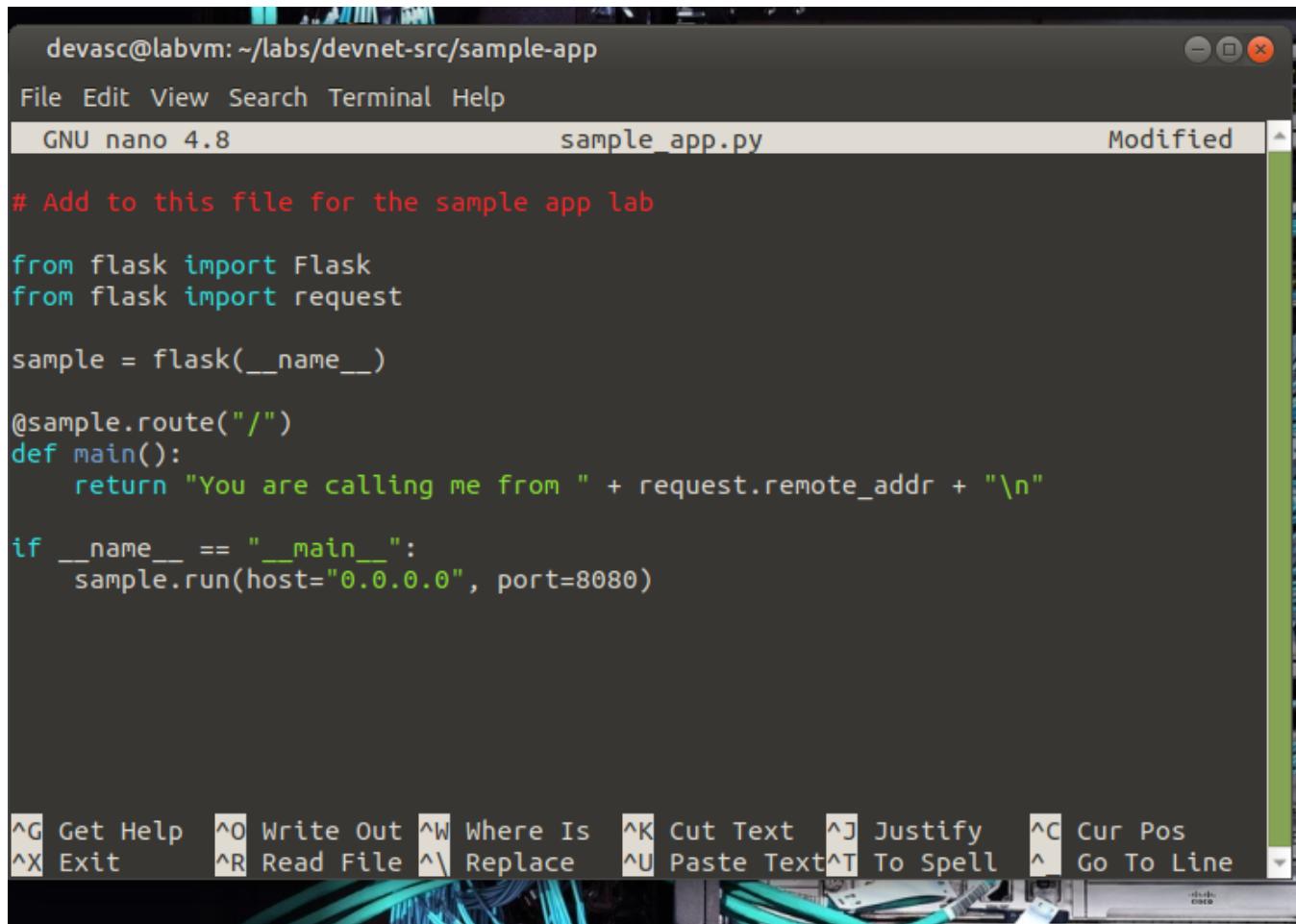
The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The status bar at the bottom shows various keyboard shortcuts for navigating and editing the text.

Notice the `@sample.route("/")` Flask statement. Frameworks such as Flask use a routing technique (.route) to refer to an application URL (this not to be confused with network routing). Here the "/" (root directory) is bound to the `main()` function. So, when the user goes to `http://localhost:8080/` (root directory) URL, the output of the return statement will be displayed in the browser.

Step 6: Configure the app to run locally.

Finally, configure Flask to run the app locally at `http://0.0.0.0:8080`, which is also `http://localhost:8080`. Be sure to use two underscores before and after "name", and before and after "main".

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



The screenshot shows a terminal window titled "sample_app.py" which is "Modified". The file contains the following Python code:

```
# Add to this file for the sample app lab

from flask import Flask
from flask import request

sample = flask(__name__)

@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"

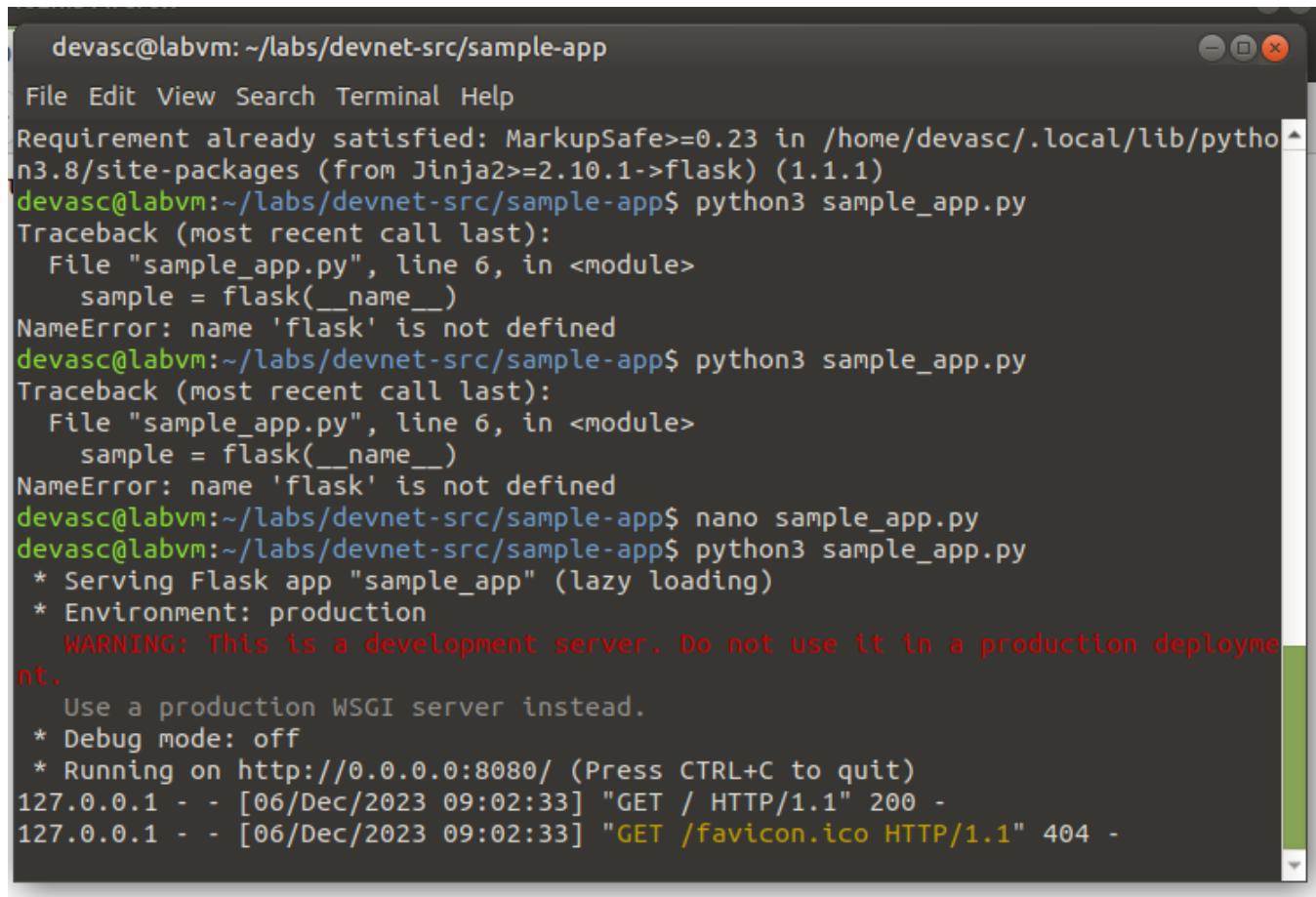
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". At the bottom, there is a toolbar with various keyboard shortcuts for file operations like "Get Help", "Write Out", "Where Is", "Cut Text", "Justify", "Cur Pos", "Exit", "Read File", "Replace", "Paste Text", "To Spell", and "Go To Line".

Step 7: Save and run your sample web app.

Save your script and run it from the command line. You should see the following output which indicates that your "sample-app" server is running. If you do not see the following output or if you receive an error message, check your sample_app.py script carefully.

```
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)
```



The screenshot shows a terminal window titled "Terminal". The command "python3 sample_app.py" is run, resulting in the following output:

```
devasc@labvm:~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
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$ python3 sample_app.py
Traceback (most recent call last):
  File "sample_app.py", line 6, in <module>
    sample = flask(__name__)
NameError: name 'flask' is not defined
devasc@labvm:~/labs/devnet-src/sample-app$ python3 sample_app.py
Traceback (most recent call last):
  File "sample_app.py", line 6, in <module>
    sample = flask(__name__)
NameError: name 'flask' is not defined
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/Dec/2023 09:02:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2023 09:02:33] "GET /favicon.ico HTTP/1.1" 404 -
```

Step 8: Verify the server is running.

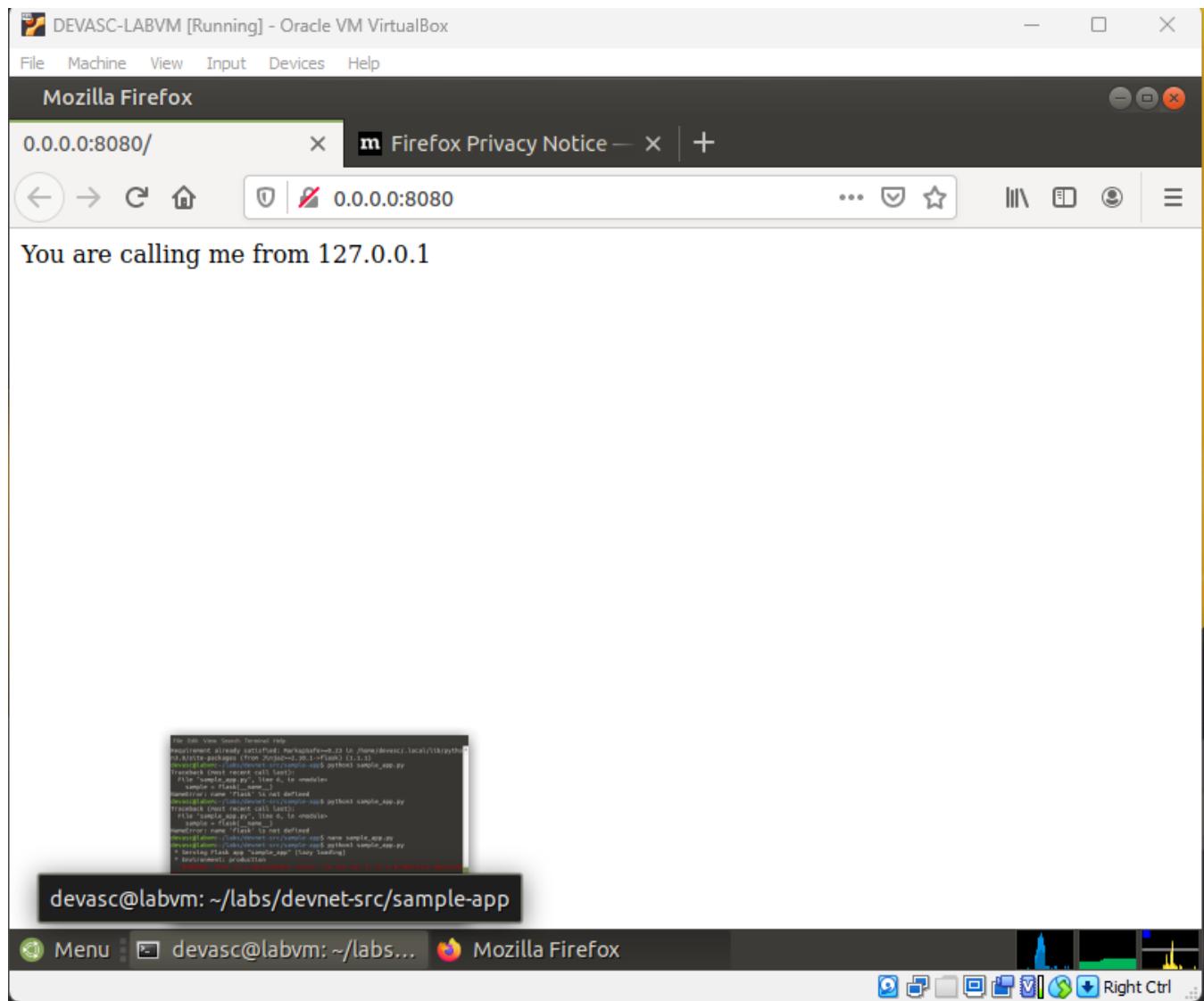
You can verify the server is running in one of two ways.

- Open the Chromium web browser and enter 0.0.0.0:8080 in the URL field. You should get the following output:

You are calling me from 127.0.0.1

If you receive an "HTTP 400 Bad Request" response, check your sample_app.py script carefully.

Lab - Build a Sample Web App in a Docker Container



- b. Open another terminal window and use the command-line URL tool (cURL) to verify the server's response.

```
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
```

You are calling me from 127.0.0.1

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

```
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
```

You are calling me from 127.0.0.1

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

Step 9: Stop the server.

Return to the terminal window where the server is running and press CTRL+C to stop the server.

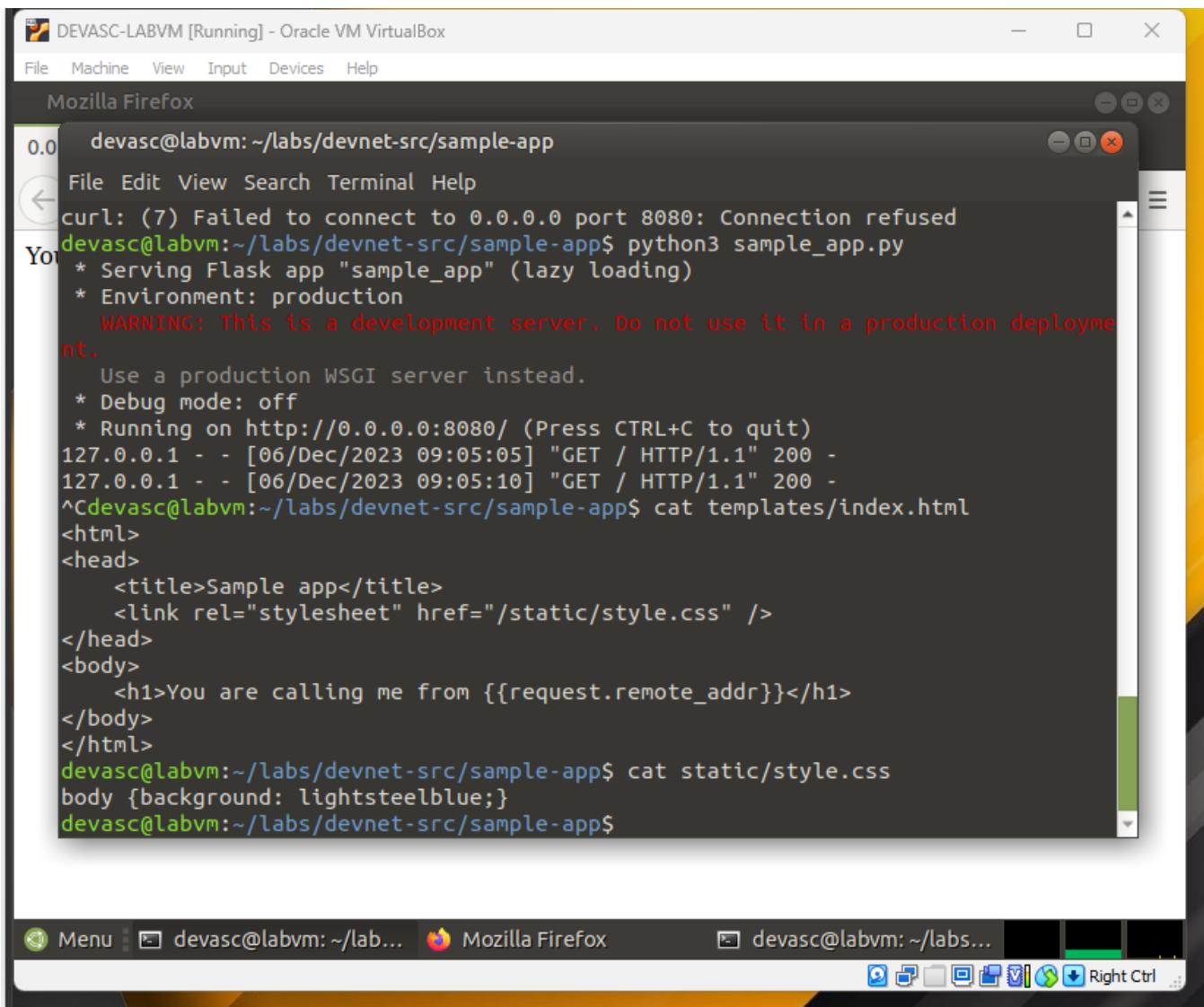
Part 4: Configure the Web App to use Website Files

In this part, build out the sample web app to include an **index.html** page and **style.css** specification. The **index.html** is typically the first page loaded in a client's web browser when visiting your website. The **style.css** is a style sheet used to customize the look of the web page.

Step 1: Explore the directories that will be used by the web app.

The directories **templates** and **static** are already in the **sample-app** directory. Open the **index.html** and **style.css** to view their contents. If you are familiar with HTML and CSS, feel free to customize these directories and files as much as you like. However, be sure you keep the embedded **{{request.remote_addr}}** Python code in the **index.html** file as this is the dynamic aspect of the sample web app.

```
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 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Mozilla Firefox
0.0  devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
curl: (7) Failed to connect to 0.0.0.0 port 8080: Connection refused
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/Dec/2023 09:05:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2023 09:05:10] "GET / HTTP/1.1" 200 -
^Cdevasc@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$
```

Step 2: Update the Python code for the sample web app.

Now that you have explored the basic website files, you need to update the **sample_app.py** file so that it renders the **index.html** file instead of just returning data. Generating HTML content using Python code can be cumbersome, especially when using conditional statements or repeating structures. The HTML file can be rendered in Flask automatically using the `render_template` function. This requires importing the `render_template` method from the `flask` library and editing to the `return` function. Make the highlighted edits to your script.

```
from flask import Flask
from flask import request
from flask import render_template

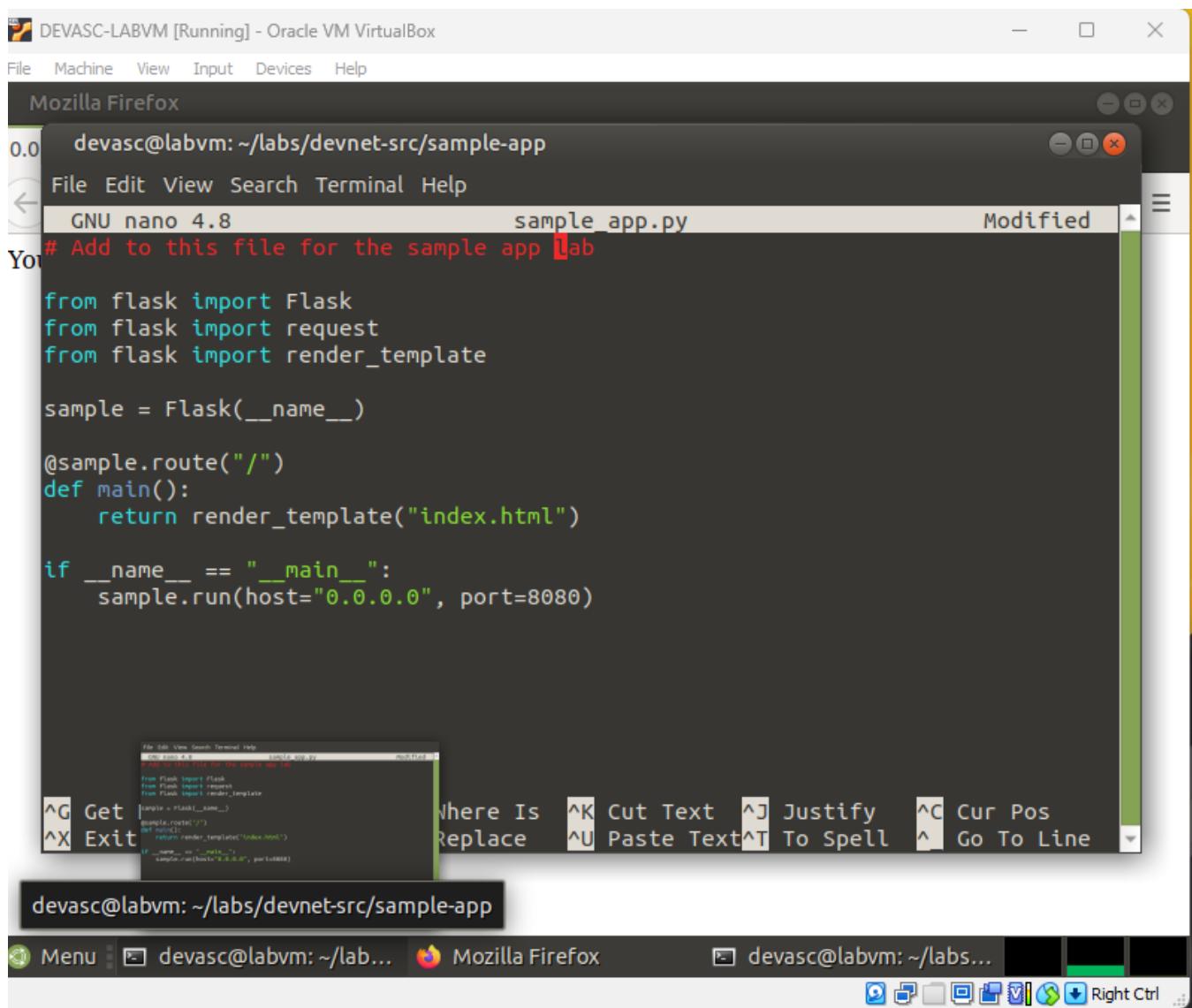
sample = Flask(__name__)

@sample.route("/")
def main():
```

Lab - Build a Sample Web App in a Docker Container

```
return render_template("index.html")

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



Step 3: Save and run your script.

Save and run your **sample-app.py** script. You should get output like the following:

```
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)
```

The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal session is as follows:

```
0.0 devasc@labvm:~/labs/devnet-src/sample-app
File Edit View Terminal Help
127.0.0.1 - - [06/Dec/2023 09:05:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Dec/2023 09:05:10] "GET / HTTP/1.1" 200 -
You are calling me from 127.0.0.1
^cdevasc@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$ 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 C
[1]
[2]
```

A modal window titled "File Edit View Details Terminal Help" is displayed, showing the command "curl -s http://127.0.0.1:8080" and its output:

```
curl: (7) Failed to connect to 127.0.0.1 port 8080: connection refused
curl: (7) Failed to connect to 127.0.0.1 port 8080: connection refused
You are calling me from 127.0.0.1
devasc@labvm:~/labs/devnet-src/sample-app$ curl -s http://127.0.0.1:8080
[1]
[2]
```

The terminal prompt "devasc@labvm: ~/labs/devnet-src/sample-app" is highlighted.

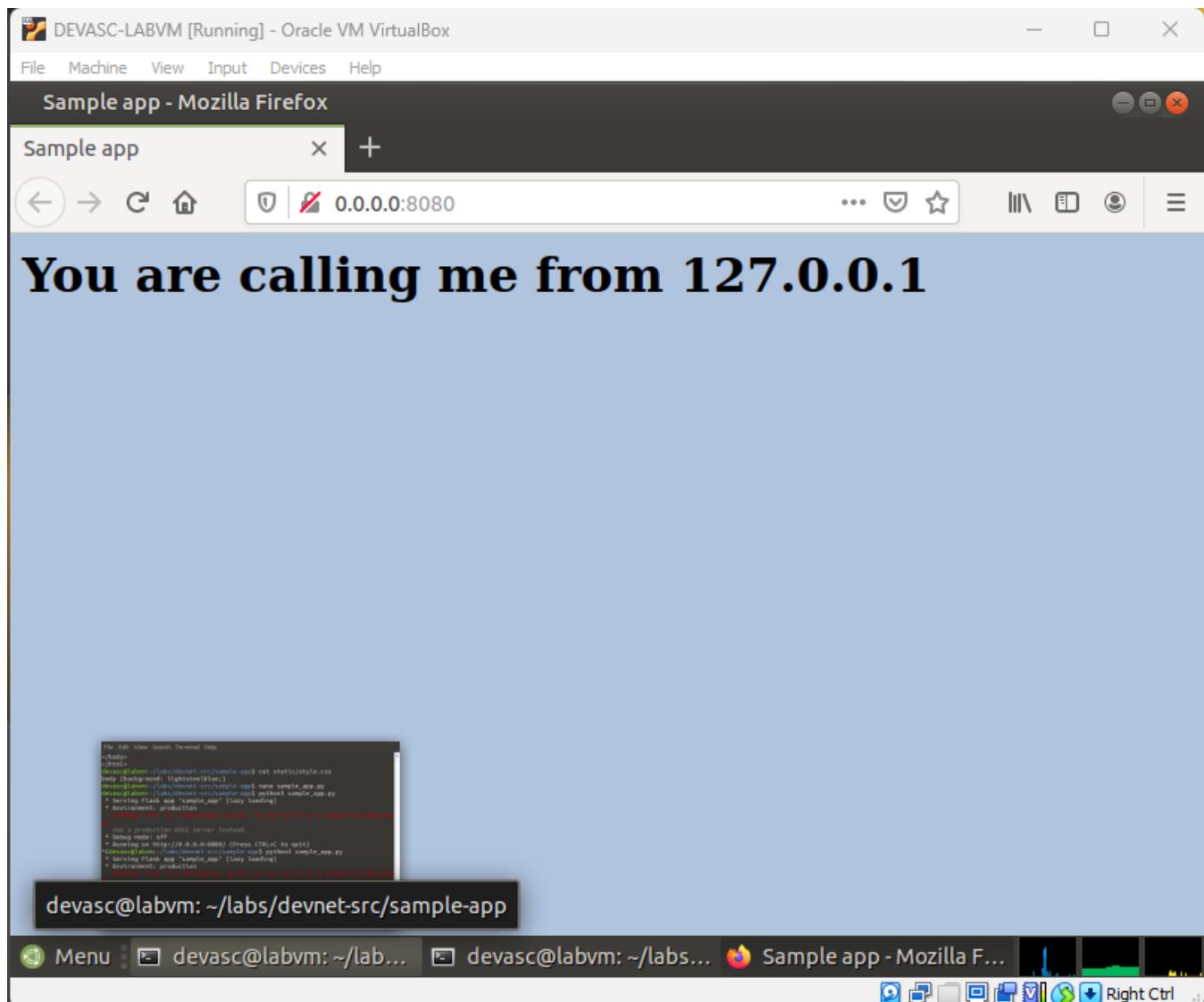
Note: If you got Traceback output and an error with the message with something like **OSSError: [Errno 98] Address already in use**, then you did not shutdown your previous server. Return to the terminal window where that server is running and press CTRL+C to end the server process. Re-run your script.

Step 4: Verify your program is running.

Again, you can verify your program is running in one of two ways.

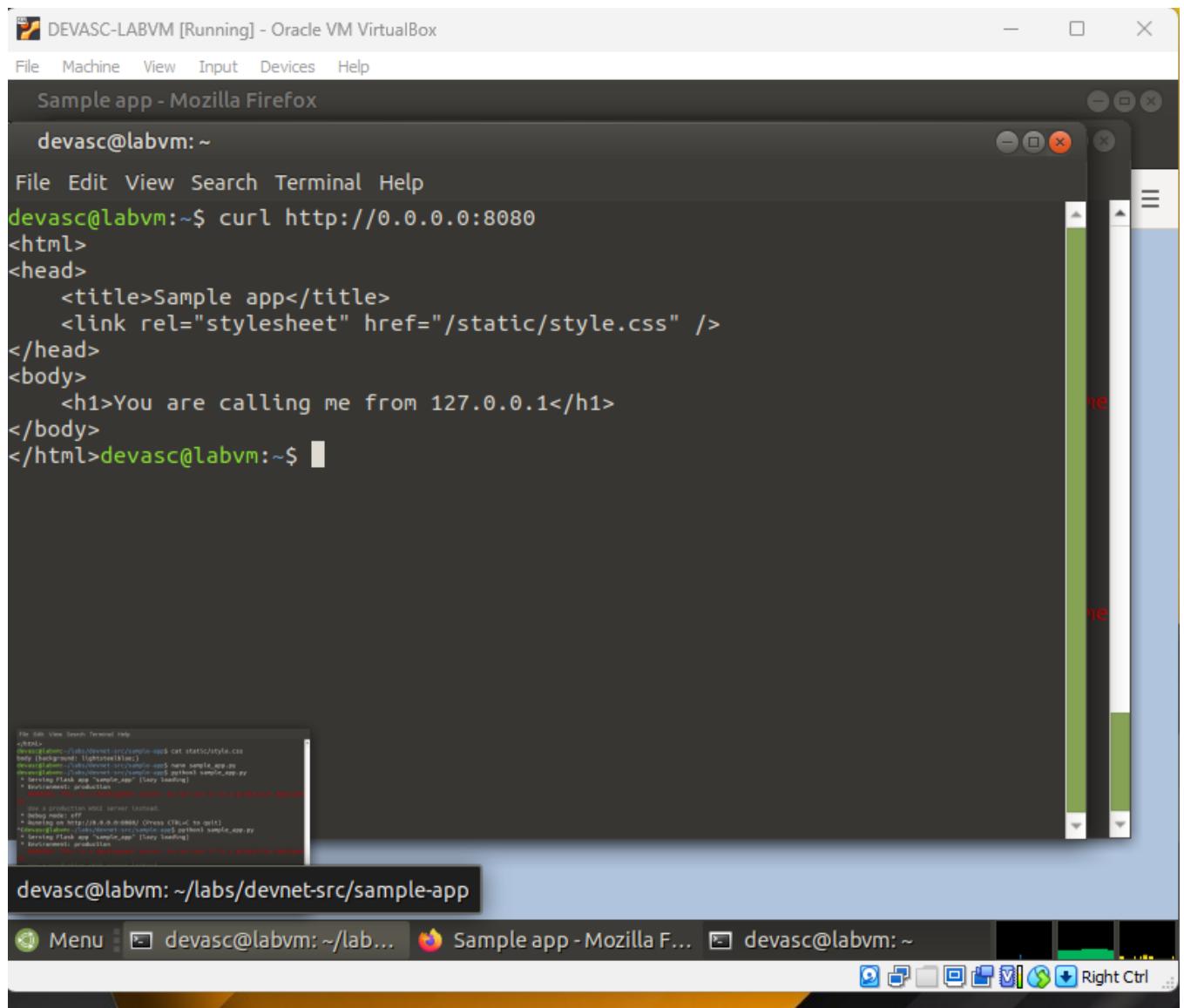
- Open the Chromium web browser and enter 0.0.0.0:8080 in the URL field. You should get the same output as before. However, your background will be light steel blue and the text will be formatted as H1.

You are calling me from 127.0.0.1



- b. Open another terminal window and use the `curl` command to verify the server's response. This is where you will see the result of the HTML code rendered automatically using the `render_template` function. In this case, you will get all the HTML content. However, the dynamic Python code will be replaced with the value for `{{request.remote_addr}}`. Also, notice your prompt will be on the same line as the last line of HMTL output. Press ENTER to get a new line.

```
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$
```



Step 5: Stop the server.

Return to the terminal window where the server is running and press **CTRL+C** to stop the server.

Part 5: Create a Bash Script to Build and Run a Docker Container

An application can be deployed on a bare metal server (physical server dedicated to a single-tenant environment) or in a virtual machine, like you just did in the previous Part. It can also be deployed in a containerized solution like Docker. In this part, you will create a bash script and add commands to it that complete the following tasks to build and run a Docker container:

- Create temporary directories to store the website files.
- Copy the website directories and `sample_app.py` to the temporary directory.
- Build a Dockerfile.
- Build the Docker container.

Lab - Build a Sample Web App in a Docker Container

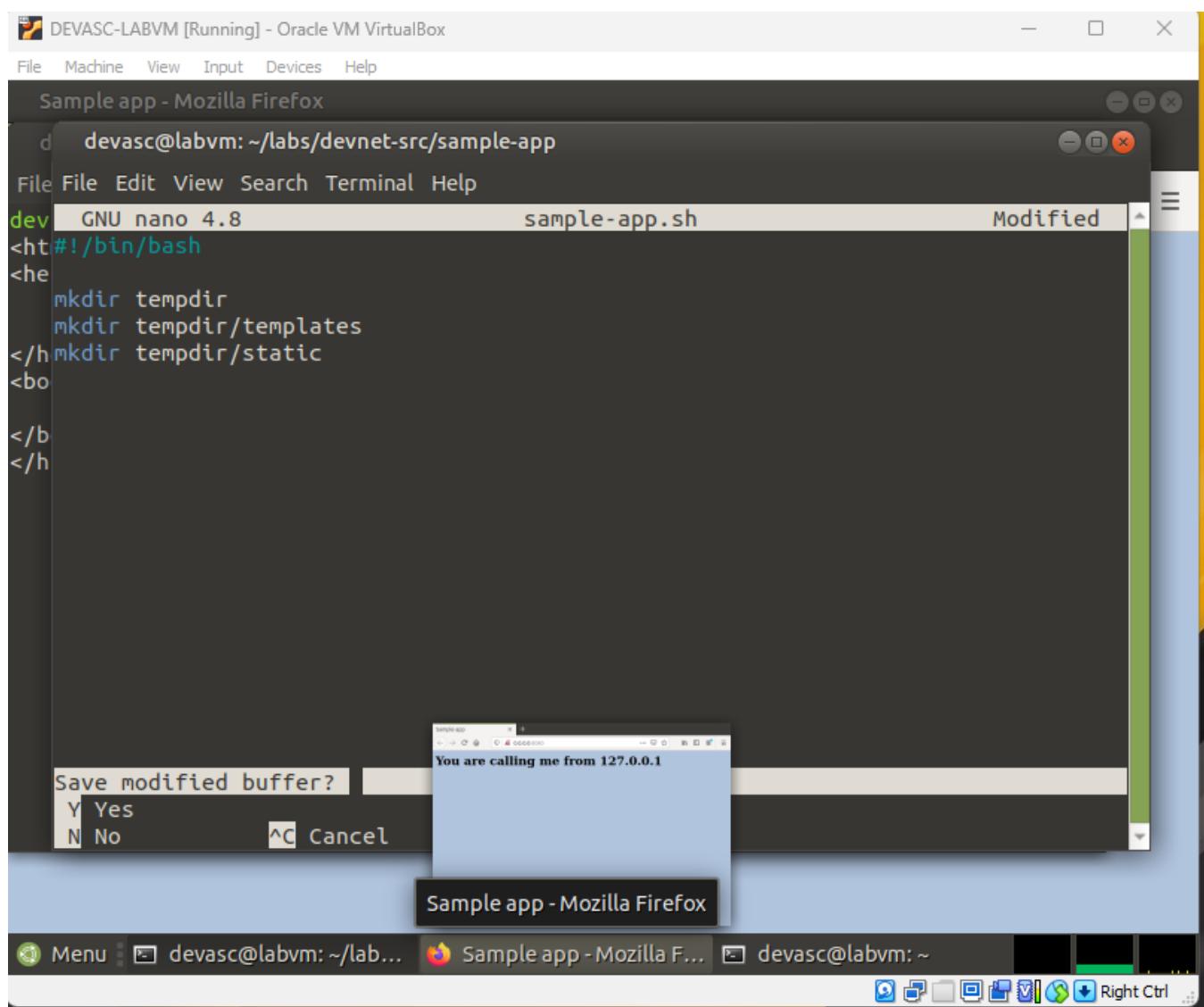
- Start the container and verify it is running.

Step 1: Create temporary directories to store the website files.

Open the **sample-app.sh** bash script file in the `~/labs/devnet-src/sample-app` directory. Add the 'she-bang' and the commands to create a directory structure with **tempdir** as the parent folder.

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static
```



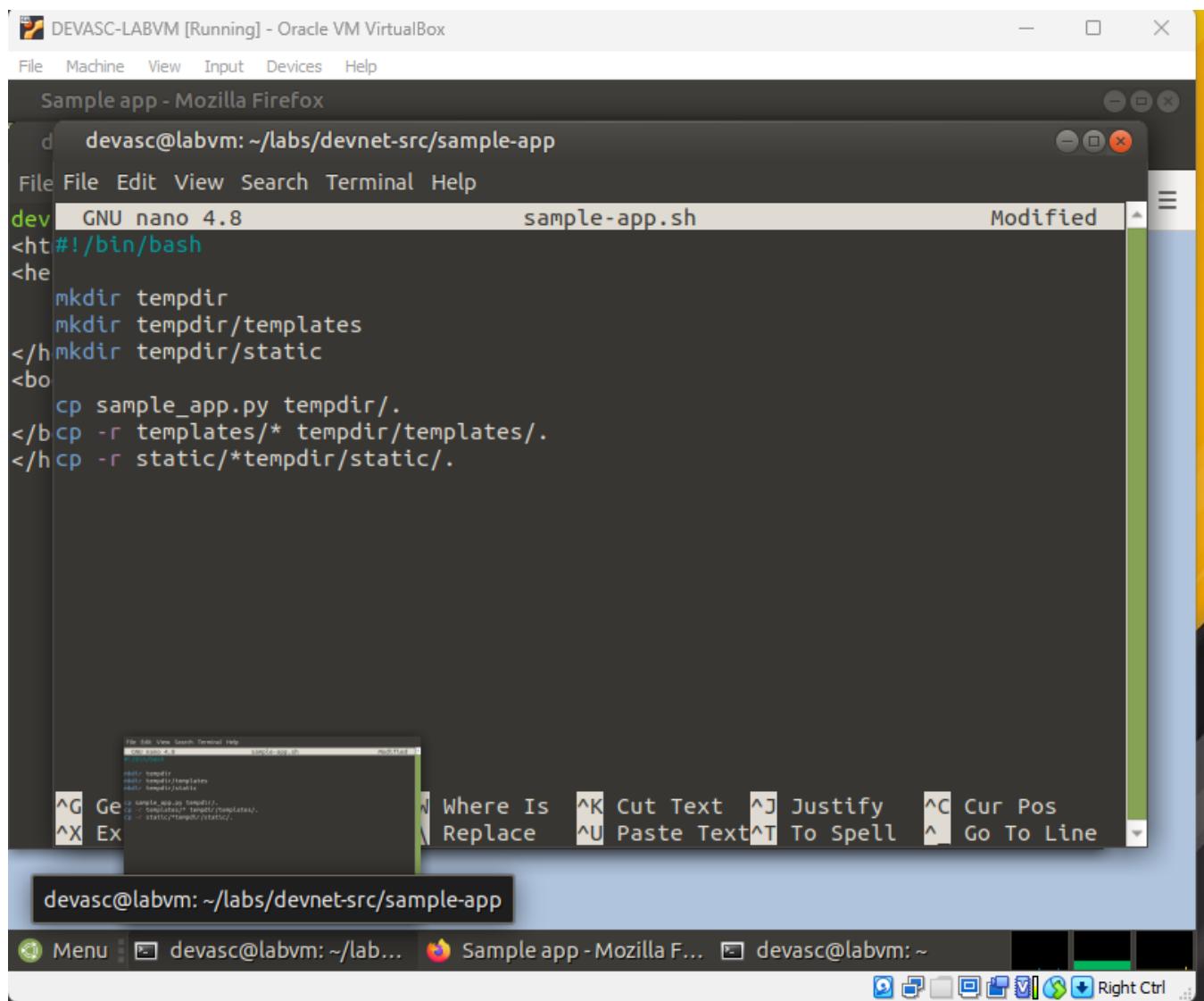
Step 2: Copy the website directories and sample_app.py to the temporary directory.

in the **sample-app.sh** file, add the commands to copy the website directory and script to **tempdir**.

```
cp sample_app.py tempdir/.
```

Lab - Build a Sample Web App in a Docker Container

```
cp -r templates/* tempdir/templates/.  
cp -r static/* tempdir/static/.
```



Step 3: Create a Dockerfile.

In this step, you enter the necessary bash **echo** commands to the **sample-app.sh** file to create a Dockerfile in the **tempdir**. This Dockerfile will be used to build the container.

- You need Python running in the container, so add the Docker **FROM** command to install Python in the container.

```
echo "FROM python" >> tempdir/Dockerfile
```

- Your **sample_app.py** script needs Flask, so add the Docker **RUN** command to install Flask in the container.

```
echo "RUN pip install flask" >> tempdir/Dockerfile
```

- Your container will need the website folders and the **sample_app.py** script to run the app, so add the Docker **COPY** commands to add them to a directory in the Docker container. In this example, you will

create **/home/myapp** as the parent directory inside the Docker container. Besides copying the `sample_app.py` file to the `Dockerfile`, you will also be copying the `index.html` file from the `templates` directory and the `style.css` file from the `static` directory.

```
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
```

- d. Use the Docker **EXPOSE** command to expose port 8080 for use by the webserver.

```
echo "EXPOSE 8080" >> tempdir/Dockerfile
```

- e. Finally, add the Docker **CMD** command to execute the Python script.

```
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile
```

Step 4: Build the Docker container.

Add the commands to the **sample-app.sh** file to switch to the **tempdir** directory and build the Docker container. The **docker build** command **-t** option allows you to specify the name of the container and the trailing period (.) indicates that you want the container built in the current directory.

```
cd tempdir
docker build -t sampleapp .
```

Step 5: Start the container and verify it is running.

- a. Add the **docker run** command to the **sample-app.sh** file to start the container.

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

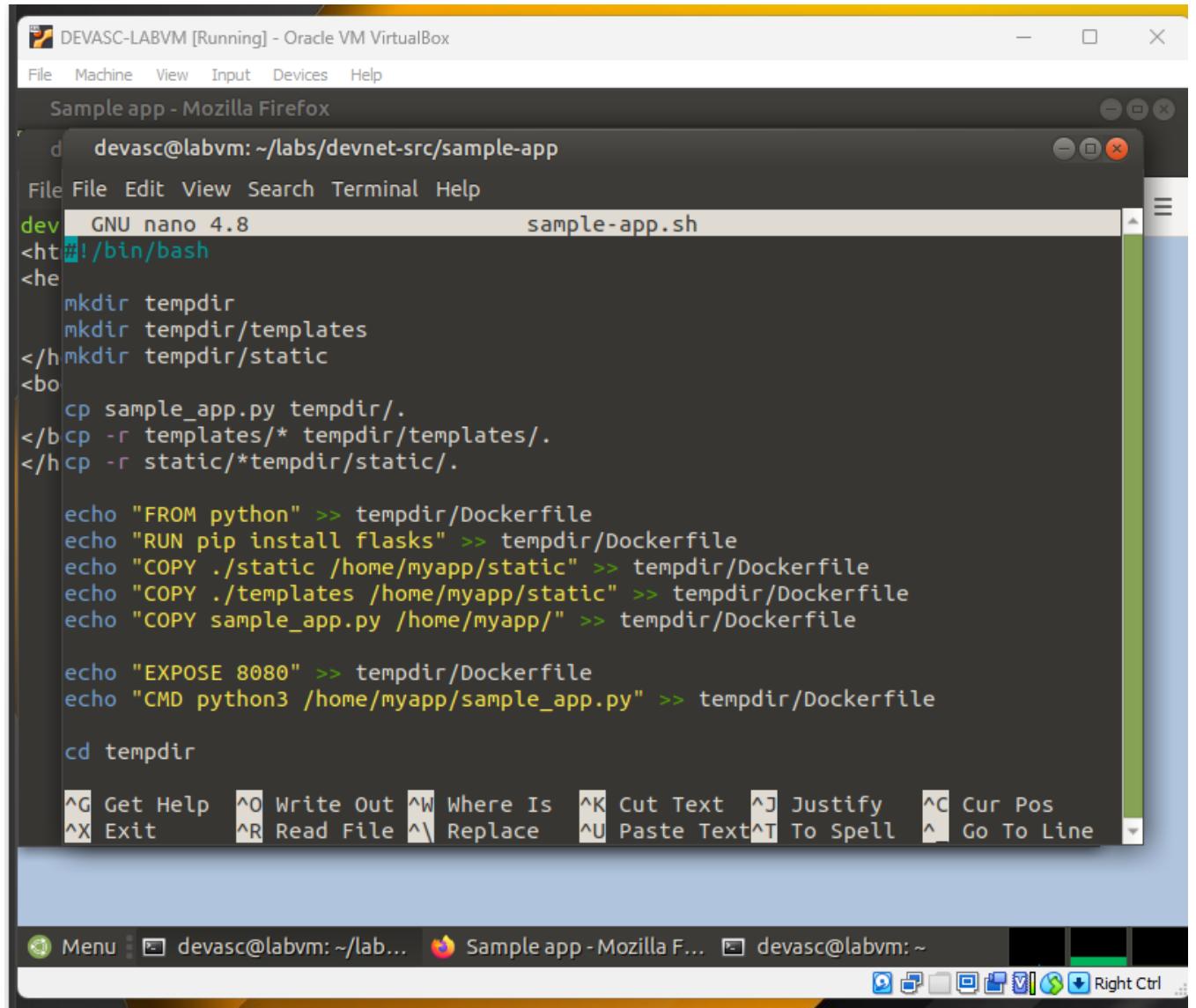
The **docker run** options indicate the following:

- **-t** specifies that you want a terminal created for the container so you can access it at the command line.
- **-d** indicates that you want the container to run in the background and print the container ID when executing the **docker ps -a** command.
- **-p** specifies that you want to publish the container's internal port to the host. The first "8080" references the port for the app running in the docker container (our `sampleapp`). The second "8080" tells docker to use this port on the host. These values do not have to be the same. For example, an internal port 80 to external 800 (**80:800**).
- **--name** specifies first what you want to call the instance of the container (**samplerunning**) and then the container image that the instance will be based on (**sampleapp**). The instance name can be anything you want. However, the image name needs to match the container name you specified in the `docker build` command (**sampleapp**).

- b. Add the **docker ps -a** command to display all currently running Docker containers. This command will be the last one executed by the bash script.

```
docker ps -a
```

Lab - Build a Sample Web App in a Docker Container



The screenshot shows a Linux desktop environment with a terminal window running inside a Firefox browser window. The terminal window is titled "Sample app - Mozilla Firefox" and displays a nano editor session for a file named "sample-app.sh". The script content is a Dockerfile for a Python application. The terminal window has a dark background with light-colored text. The Firefox window title bar also shows "Sample app - Mozilla F...". The desktop taskbar at the bottom includes icons for the terminal, file manager, and other applications.

```
d  devasc@labvm: ~/labs/devnet-src/sample-app
File File Edit View Search Terminal Help
dev GNU nano 4.8           sample-app.sh
<ht#!/bin/bash
<he
</hmkdir tempdir
<bo
<htcp sample_app.py tempdir/.
<htcp -r templates/* tempdir/templates/.
<htcp -r static/*tempdir/static/.

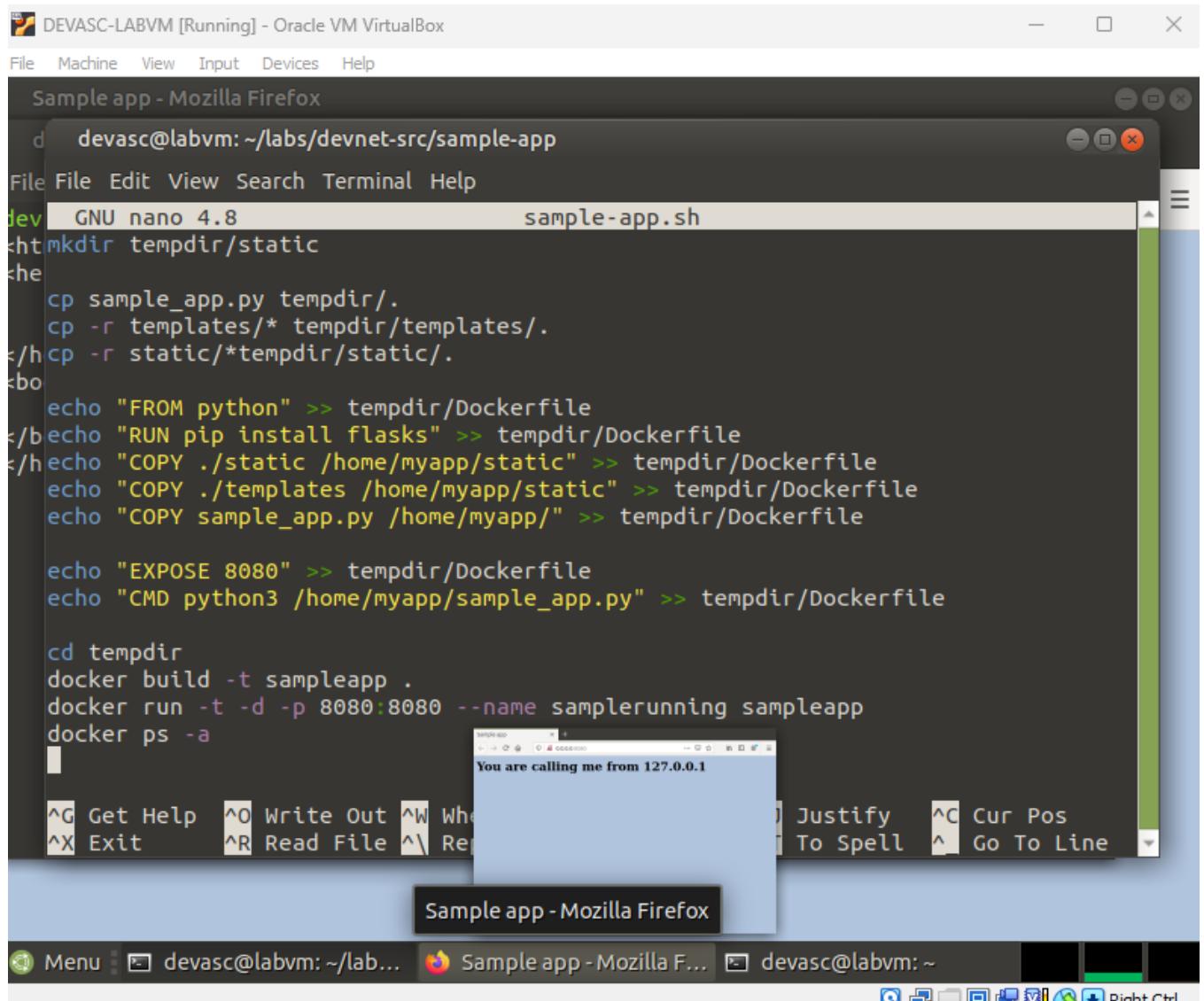
echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flasks" >> tempdir/Dockerfile
echo "COPY ./static /home/myapp/static" >> tempdir/Dockerfile
echo "COPY ./templates /home/myapp/static" >> tempdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile

echo "EXPOSE 8080" >> tempdir/Dockerfile
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir

^C 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
```

Lab - Build a Sample Web App in a Docker Container



```
DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Mozilla Firefox
d  devasc@labvm: ~/labs/devnet-src/sample-app
File File Edit View Search Terminal Help
dev  GNU nano 4.8           sample-app.sh
ht  mkdir tempdir/static
he
cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/*tempdir/static/.
bo
echo "FROM python" >> tempdir/Dockerfile
becho "RUN pip install flask" >> tempdir/Dockerfile
hecho "COPY ./static /home/myapp/static" >> tempdir/Dockerfile
echo "COPY ./templates /home/myapp/static" >> 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 Who
^X Exit      ^R Read File  ^\ Re...
Justify  ^C Cur Pos
To Spell  ^ Go To Line
Sample app - Mozilla Firefox
You are calling me from 127.0.0.1
Sample app - Mozilla Firefox
Menu  devasc@labvm: ~/lab...  Sample app - Mozilla F...  devasc@labvm: ~  Right Ctrl
```

Step 6: Save your bash script.

Part 6: Build, Run, and Verify the Docker Container

In this part, you will execute bash script which will make the directories, copy over the files, create a Dockerfile, build the Docker container, run an instance of the Docker container, and display output from the `docker ps -a` command showing details of the container currently running. Then you will investigate the Docker container, stop the container from running, and remove the container.

Note: Be sure you stopped any other web server processes you may still have running from the previous parts of this lab.

Step 1: Execute the bash script.

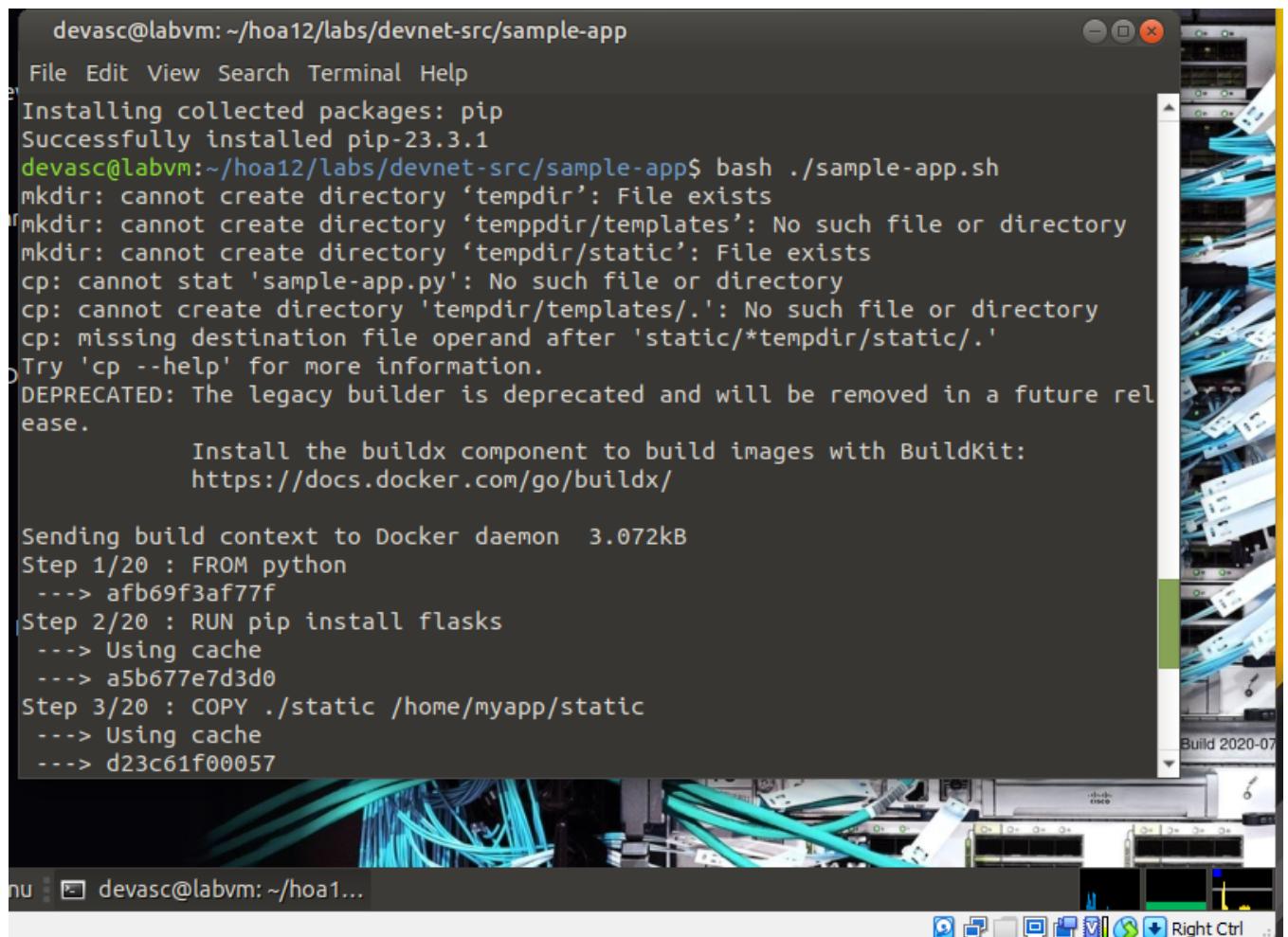
Execute the bash script from the command line. You should see output similar to the following. After creating the `tempdir` directories, the script executes the commands to build the Docker container. Notice that Step 7/7 in the output executes the `sample_app.py` that creates the web server. Also, notice the container ID. You will see this in the Docker command prompt later in the lab.

Lab - Build a Sample Web App in a Docker Container

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash ./sample-app.sh
Sending build context to Docker daemon 6.144kB
Step 1/7 : FROM python
latest: Pulling from library/python
90fe46dd8199: Pulling fs layer
35a4f1977689: Pulling fs layer
bbc37f14aded: Pull complete
74e27dc593d4: Pull complete
4352dcff7819: Pull complete
deb569b08de6: Pull complete
98fd06fa8c53: Pull complete
7b9cc4fdefe6: Pull complete
512732f32795: Pull complete
Digest: sha256:ad7fb5bb4770e08bf10a895ef64a300b288696a1557a6d02c8b6fba98984b86a
Status: Downloaded newer image for python:latest
--> 4f7cd4269fa9
Step 2/7 : RUN pip install flask
--> Running in 32d28026afea
Collecting flask
    Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
Collecting click>=5.1
    Downloading click-7.1.2-py2.py3-none-any.whl (82 kB)
Collecting Jinja2>=2.10.1
    Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)
Collecting Werkzeug>=0.15
    Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
Collecting itsdangerous>=0.24
    Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=0.23
    Downloading MarkupSafe-1.1.1-cp38-cp38-manylinux1_x86_64.whl (32 kB)
Installing collected packages: click, MarkupSafe, Jinja2, Werkzeug, itsdangerous, flask
Successfully installed Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2 flask-1.1.2 itsdangerous-1.1.0
Removing intermediate container 32d28026afea
--> 619aee23fd2a
Step 3/7 : COPY ./static /home/myapp/static/
--> 15fac1237eec
Step 4/7 : COPY ./templates /home/myapp/templates/
--> dc807b5cf615
Step 5/7 : COPY sample_app.py /home/myapp/
--> d4035a63ae14
Step 6/7 : EXPOSE 8080
--> Running in 40c2d35aa29a
Removing intermediate container 40c2d35aa29a
--> eb789099a678
Step 7/7 : CMD python3 /home/myapp/sample_app.py
--> Running in 41982e2c6209
Removing intermediate container 41982e2c6209
```

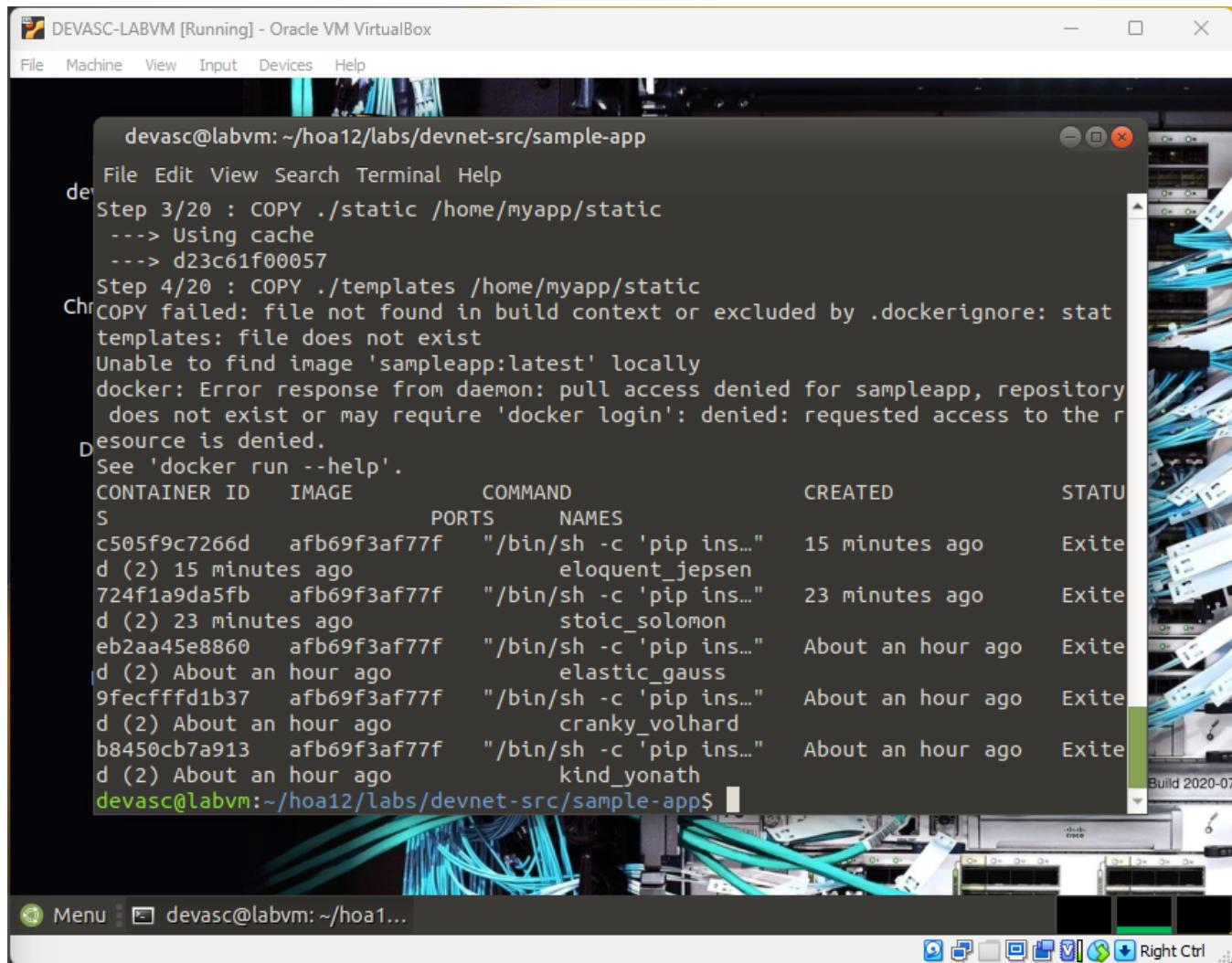
Lab - Build a Sample Web App in a Docker Container

```
--> a2588e9b0593
Successfully built a2588e9b0593
Successfully tagged sampleapp:latest
8953a95374ff8ebc203059897774465312acc8f0ed6abd98c4c2b04448a56ba5
CONTAINER ID        IMAGE               COMMAND             CREATED            NAMES
STATUS              PORTS              NAMES
8953a95374ff      sampleapp          "/bin/sh -c 'python ..."   1 second ago     samplerunning
Up Less than a second   0.0.0.0:8080->8080/tcp    samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$
```



```
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app
File Edit View Search Terminal Help
Installing collected packages: pip
Successfully installed pip-23.3.1
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$ bash ./sample-app.sh
mkdir: cannot create directory 'tempdir': File exists
mkdir: cannot create directory 'tempdir/templates': No such file or directory
mkdir: cannot create directory 'tempdir/static': File exists
cp: cannot stat 'sample-app.py': No such file or directory
cp: cannot create directory 'tempdir/templates/.': No such file or directory
cp: missing destination file operand after 'static/*tempdir/static/.'
Try 'cp --help' for more information.
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 3.072kB
Step 1/20 : FROM python
--> afb69f3af77f
Step 2/20 : RUN pip install flasks
--> Using cache
--> a5b677e7d3d0
Step 3/20 : COPY ./static /home/myapp/static
--> Using cache
--> d23c61f00057
```



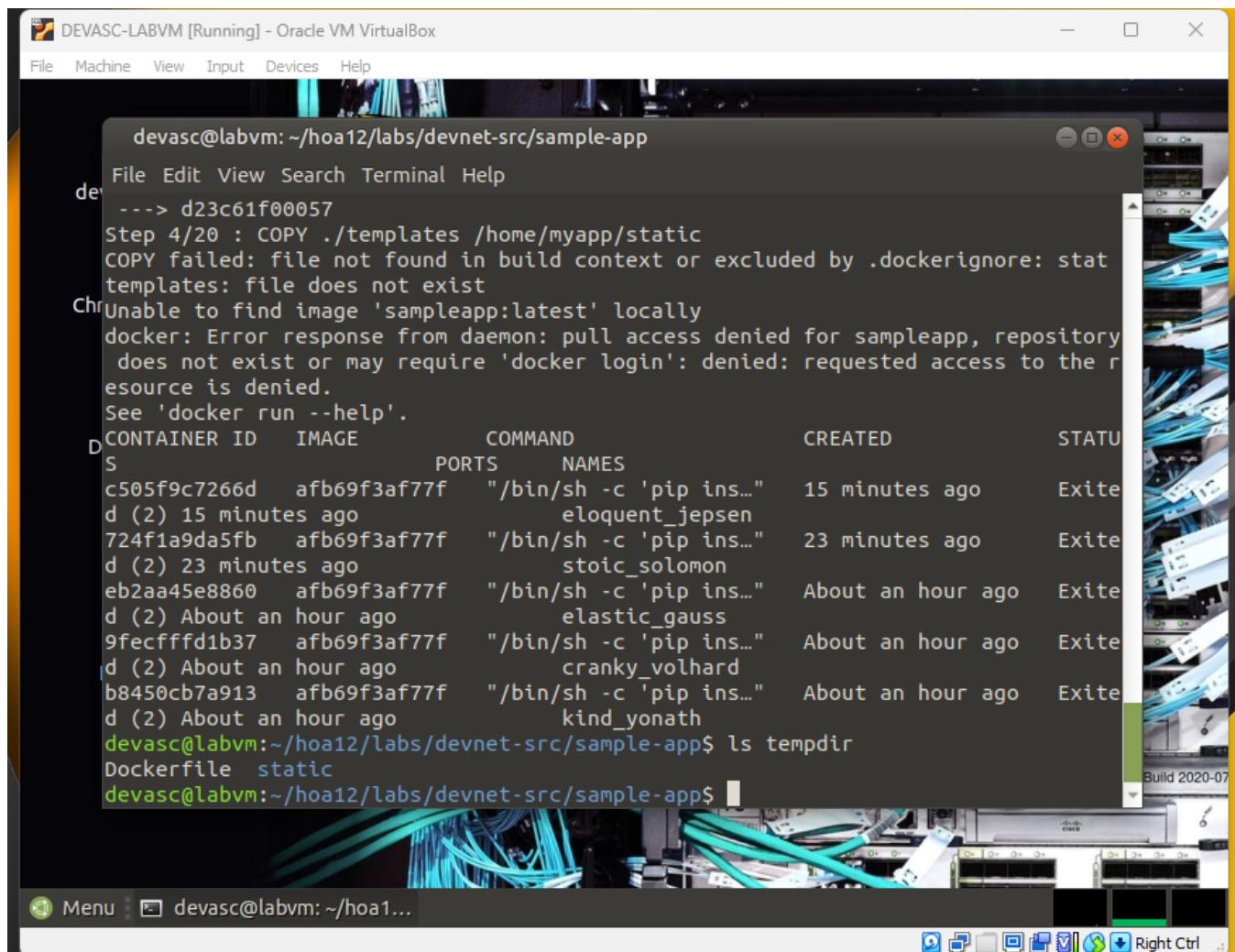
```
DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app
File Edit View Search Terminal Help
Step 3/20 : COPY ./static /home/myapp/static
--> Using cache
--> d23c61f00057
Step 4/20 : COPY ./templates /home/myapp/static
COPY failed: file not found in build context or excluded by .dockerignore: stat
templates: file does not exist
Unable to find image 'sampleapp:latest' locally
docker: Error response from daemon: pull access denied for sampleapp, repository
does not exist or may require 'docker login': denied: requested access to the r
esource is denied.
See 'docker run --help'.
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
S                   PORTS              NAMES
c505f9c7266d      afb69f3af77f    "/bin/sh -c 'pip ins..."   15 minutes ago    Exited (0)
d (2) 15 minutes ago          eloquent_jepsen
724f1a9da5fb      afb69f3af77f    "/bin/sh -c 'pip ins..."   23 minutes ago    Exited (0)
d (2) 23 minutes ago          stoic_solomon
eb2aa45e8860      afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (0)
d (2) About an hour ago      elastic_gauss
9fecffffd1b37     afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (0)
d (2) About an hour ago      cranky_volhard
b8450cb7a913      afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (0)
d (2) About an hour ago      kind_yonath
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$
```

Step 2: Investigate the running Docker container and the web app.

- The creation of the **tempdir** directories is not shown in the output for the script. You could add **echo** commands to print out messages when they are successfully created. You can also verify they are there with the **ls** command. Remember, this directory has the files and folders used to build the container and launch the web app. It is not the container that was built.

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

Lab - Build a Sample Web App in a Docker Container



The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal output is as follows:

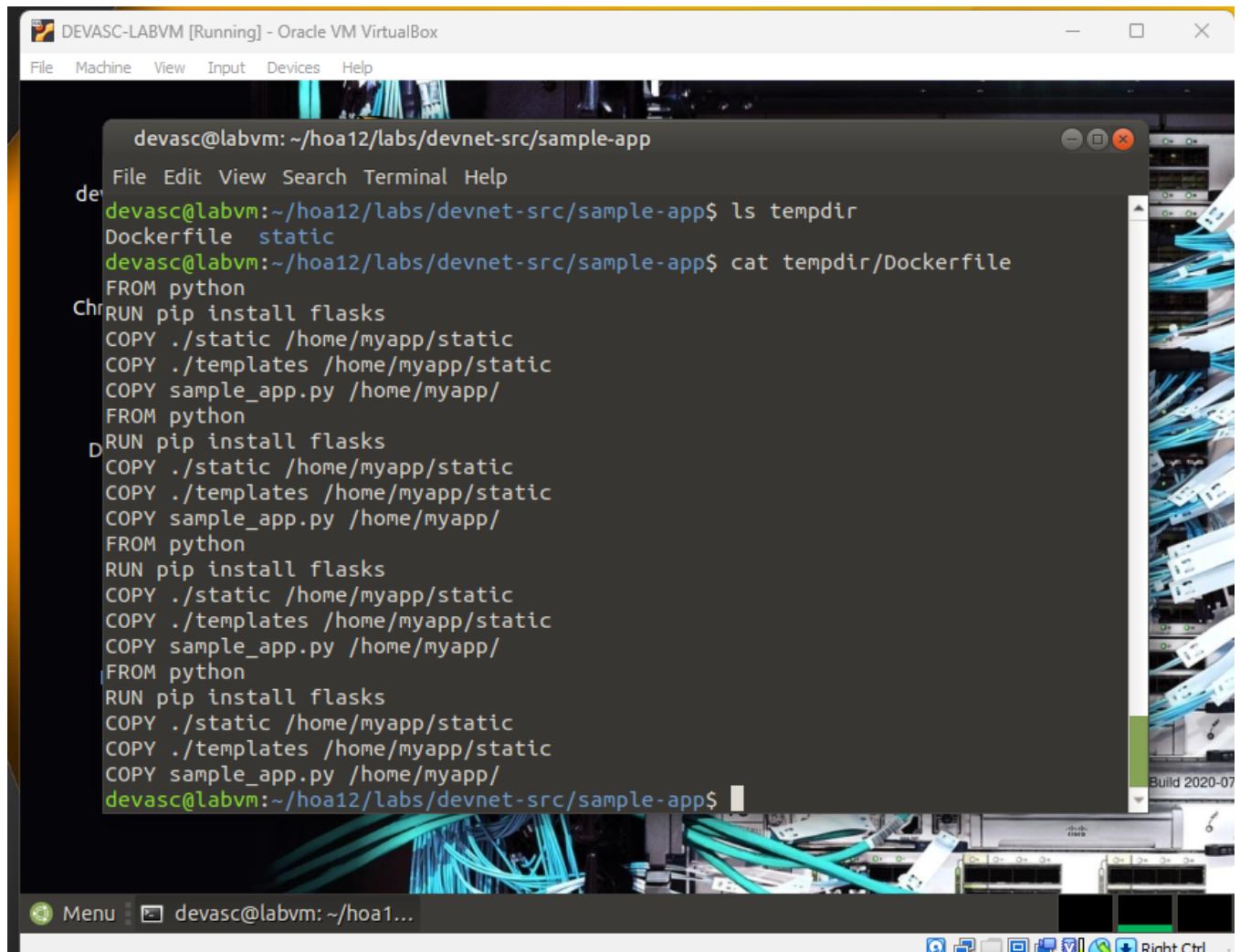
```
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app
File Edit View Search Terminal Help
de ---> d23c61f00057
Step 4/20 : COPY ./templates /home/myapp/static
COPY failed: file not found in build context or excluded by .dockerignore: stat
templates: file does not exist
ChrUnable to find image 'sampleapp:latest' locally
docker: Error response from daemon: pull access denied for sampleapp, repository
does not exist or may require 'docker login': denied: requested access to the r
esource is denied.
See 'docker run --help'.
D CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS     NAMES
S c505f9c7266d      afb69f3af77f    "/bin/sh -c 'pip ins..."   15 minutes ago    Exited (1)
d (2) 15 minutes ago          eloquent_jepsen
724f1a9da5fb      afb69f3af77f    "/bin/sh -c 'pip ins..."   23 minutes ago    Exited (1)
d (2) 23 minutes ago          stoic_solomon
eb2aa45e8860      afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (1)
d (2) About an hour ago      elastic_gauss
9fecfffd1b37      afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (1)
d (2) About an hour ago      cranky_volhard
b8450cb7a913      afb69f3af77f    "/bin/sh -c 'pip ins..."   About an hour ago  Exited (1)
d (2) About an hour ago      kind_yonath
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$ ls tempdir
Dockerfile static
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$
```

The terminal prompt shows the user is in the directory `~/hoa12/labs/devnet-src/sample-app`. The command `ls tempdir` was run, showing the files `Dockerfile` and `static`.

- b. Notice the Dockerfile created by your bash script. Open this file to see how it looks in its final form without the `echo` commands.

```
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
```

Lab - Build a Sample Web App in a Docker Container



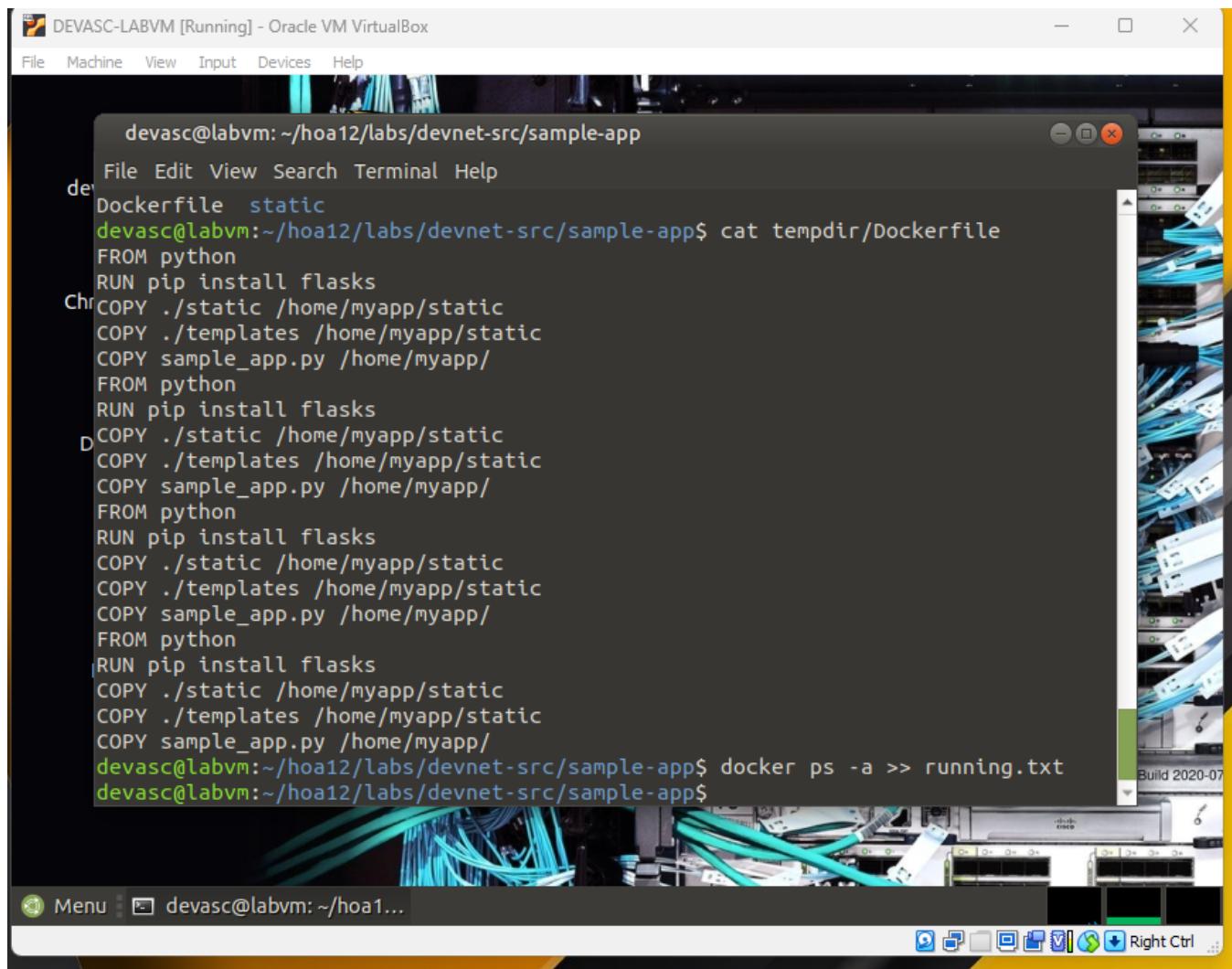
The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal is running on a Linux system with a root prompt. The user has typed the command "cat tempdir/Dockerfile" to view the contents of a Dockerfile. The Dockerfile defines a Python application structure and dependencies. The terminal window is set against a background image of server racks and network cables.

```
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
RUN pip install flasks
COPY ./static /home/myapp/static
COPY ./templates /home/myapp/static
COPY sample_app.py /home/myapp/
FROM python
RUN pip install flasks
COPY ./static /home/myapp/static
COPY ./templates /home/myapp/static
COPY sample_app.py /home/myapp/
FROM python
RUN pip install flasks
COPY ./static /home/myapp/static
COPY ./templates /home/myapp/static
COPY sample_app.py /home/myapp/
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$
```

- c. The output for the **docker ps -a** command may be hard to read depending on the width of your terminal display. You can redirect it to a text file where you can view it better without word wrapping.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/labs/devnet-src/sample-app$
```

Lab - Build a Sample Web App in a Docker Container

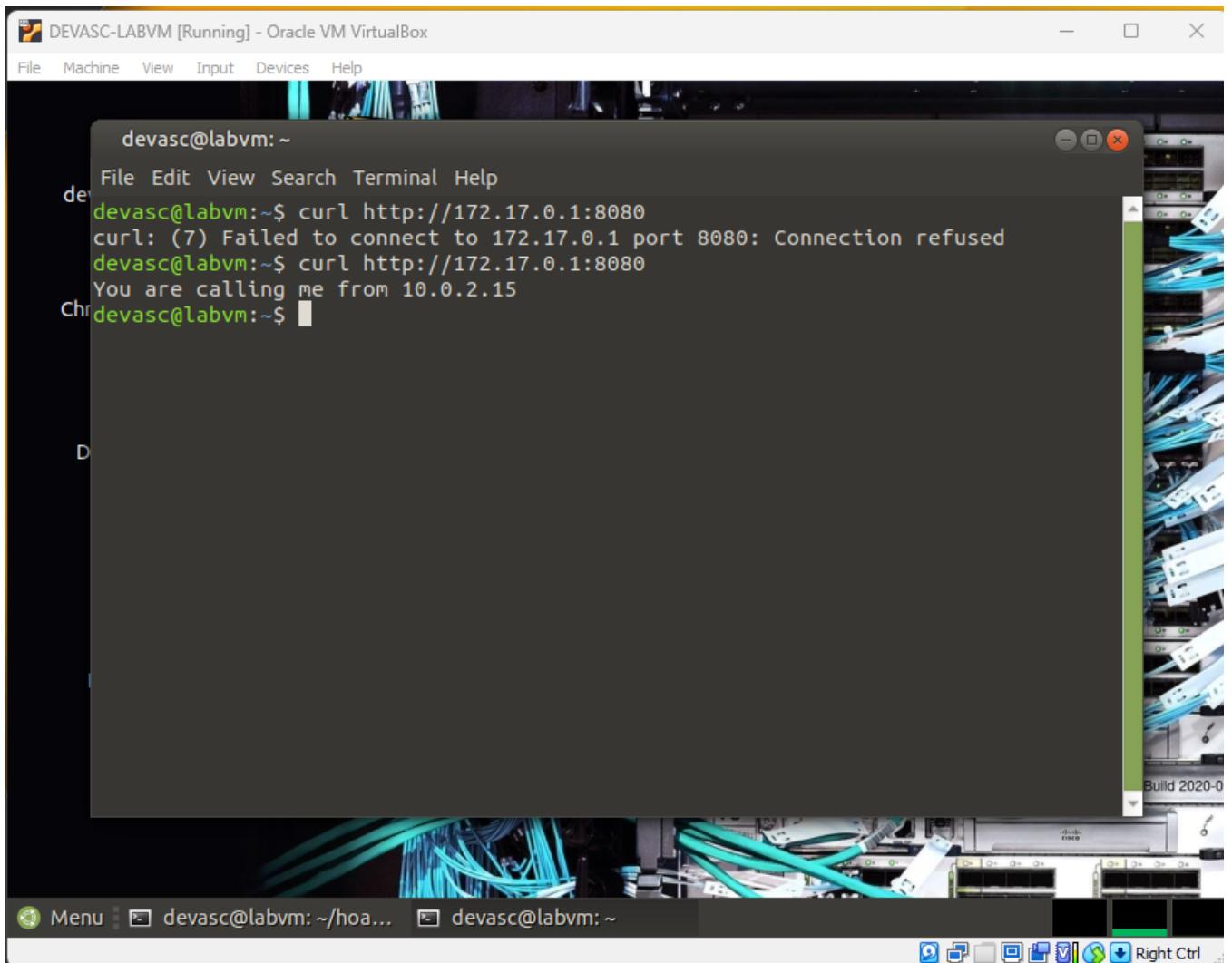


```
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
Dockerfile static
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$
```

- d. The Docker container creates its own IP address from a private network address space. Verify the web app is running and reporting the IP address. In a web browser at <http://localhost:8080>, you should see the message **You are calling me from 172.17.0.1** formatted as H1 on a light steel blue background. You can also use the **curl** command, if you like.

```
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$
```

Lab - Build a Sample Web App in a Docker Container

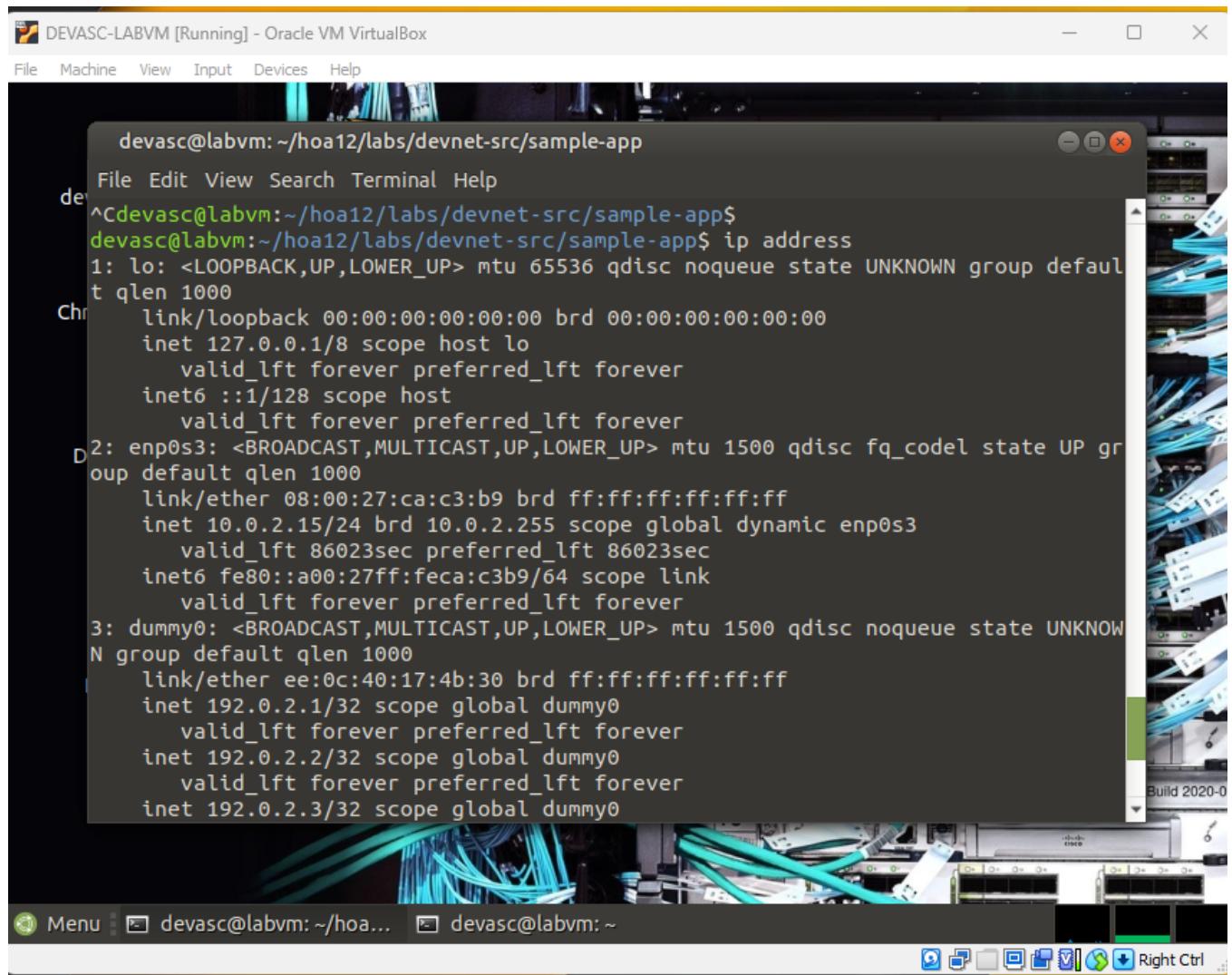


- e. By default, Docker uses the IPv4 172.17.0.0/16 subnet for container networking. (This address can be changed if necessary.) Enter the command **ip address** to display all the IP addresses used by your instance of the DEVASC VM. You should see the loopback address 127.0.0.1 that the web app used earlier in the lab and the new Docker interface with the IP address 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
<output omitted>
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c2:d1:8a:2d 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
```

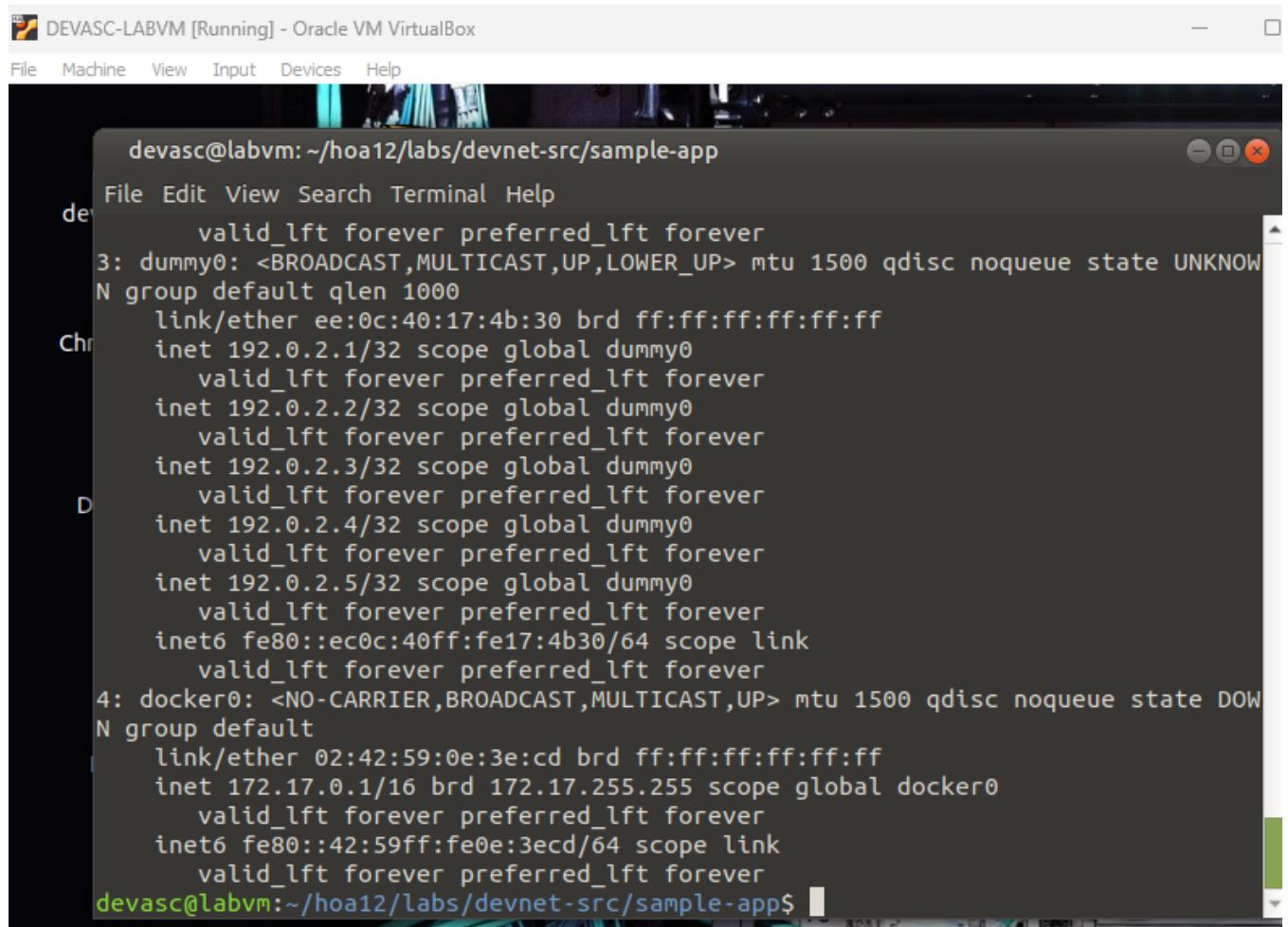
Lab - Build a Sample Web App in a Docker Container

```
inet6 fe80::42:c2ff:fed1:8a2d/64 scope link  
      valid_lft forever preferred_lft forever  
<output omitted>
```



The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The window contains a command-line interface where the user is viewing network interface details. The output shows four interfaces: "lo", "enp0s3", and two "dummy" interfaces ("dummy0" and "dummy1"). Each interface has its MTU, queueing discipline (qdisc), state, group, and queue length (qlen) displayed. IP addresses and their subnet masks are also listed, along with their lifetimes (valid_lft and preferred_lft).

```
devasc@labvm:~/hoa12/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:ca:c3:b9 brd ff:ff:ff:ff:ff:ff  
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3  
        valid_lft 86023sec preferred_lft 86023sec  
    inet6 fe80::a00:27ff:fea:c3b9/64 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 ee:0c:40:17:4b:30 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
```



The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The window contains the output of the "ip link" command, which lists network interfaces and their configurations. The output includes details for interfaces like dummy0, docker0, and several ethernet interfaces (eth0, eth1, etc.) with their MTU, queueing discipline (qdisc), and link layer information. The prompt at the bottom indicates the user is in a Docker container.

```
devasc@labvm: ~/hoa12/labs/devnet-src/sample-app$ ip link
1: dummy0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 00:0c:40:17:4b:30 brd ff:ff:ff:ff:ff:ff
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether ee:0c:40:17:4b:30 brd ff:ff:ff:ff:ff:ff
3: eth1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 192.0.2.1/32 brd ff:ff:ff:ff:ff:ff
4: eth2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 192.0.2.2/32 brd ff:ff:ff:ff:ff:ff
5: eth3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 192.0.2.3/32 brd ff:ff:ff:ff:ff:ff
6: eth4: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 192.0.2.4/32 brd ff:ff:ff:ff:ff:ff
7: eth5: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 192.0.2.5/32 brd ff:ff:ff:ff:ff:ff
8: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:59:0e:3e:cd brd ff:ff:ff:ff:ff:ff
9: eth6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fe80::ec0c:40ff:fe17:4b30/64 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
10: eth7: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fe80::42:59ff:fe0e:3ecd/64 brd ff:ff:ff:ff:ff:ff
        valid_lft forever preferred_lft forever
devasc@labvm:~/hoa12/labs/devnet-src/sample-app$
```

Step 3: Access and explore the running container.

Remember that a Docker container is a way of encapsulating everything you need to run your application so that it can easily be deployed in a variety of environments--not just in your DEVASC VM.

- To access the running container, enter the **docker exec -it** command specifying the name of the running container (**samplerunning**) and that you want a bash shell (**/bin/bash**). The **-i** option specifies that you want it to be interactive and the **-t** option specifies that you want terminal access. The prompt changes to **root@containerID**. Your container ID will be different than the one shown below. Notice the container ID matches the ID shown in the output from **docker ps -a**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker exec -it samplerunning
/bin/bash
root@8953a95374ff:/#
```

- You are now in root access for the **samplerunning** Docker container. From here, you can use Linux commands to explore the Docker container. Enter **ls** to see the directory structure at the root level.

```
root@8953a95374ff:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@8953a95374ff:/#
```

- c. Recall that in your bash script, you added commands in the Dockerfile that copied your app directories and files to the **home/myapp** directory. Enter the **ls** command again for that folder to see your **sample_app.py** script and directories. To get a better understanding of what is included in your Docker container, you may wish to use the **ls** command to examine other directories such as /etc and /bin.

```
root@8953a95374ff:/# ls home/myapp/  
sample_app.py  static  templates  
root@8953a95374ff:/#
```

- d. Exit the Docker container to return to the DEVASC VM command line.

```
root@8953a95374ff:/# exit  
exit  
devasc@labvm:~/labs/devnet-src/sample-app$
```

Step 4: Stop and remove the Docker container.

- a. You can stop the Docker container with the **docker stop** command specifying the name of the running container. It will take a few seconds to clean up and cache the container. You can see that it still exists by entering the **docker ps -a** command. However, if you refresh the web page for **http://localhost:8080**, you will see the web app is no longer running.

```
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            NAMES  
STATUS              PORTS              NAMES  
df034cb53e72        sampleapp          "/bin/sh -c 'python ...'"   49 minutes ago   samplerunning  
Exited (137) 20 seconds ago  
devasc@labvm:~/labs/devnet-src/sample-app$
```

- b. You can restart a stopped container with the **docker start** command. The container will immediately spin up.

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

- c. To permanently remove the container, first stop it and then remove it with the **docker rm** command. You can always rebuild it again executing the **sample-app** program. Use the **docker ps -a** command to verify the container has been removed.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning  
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            NAMES  
STATUS              PORTS              NAMES  
devasc@labvm:~/labs/devnet-src/sample-app$
```

Lab - Build a Sample Web App in a Docker Container

```
devasc@labvm:~/labs/devnet-src/sample-app$ nano sample-app.sh
devasc@labvm:~/labs/devnet-src/sample-app$ bash ./sample-app.sh
[suppressed] The legacy buildx is deprecated and will be removed in a future release.
[suppressed] Instead, use buildx component to build images with buildkit:
[suppressed] https://docs.docker.com/go/buildx/
Sending build context to Docker daemon: 4.144kB
Step 1/7 : FROM python:3.7-slim
--> 5f7f14a0e05d
Step 2/7 : RUN pip install flask
--> Using cache
--> 83024ed440e6
Step 3/7 : COPY ./static /home/myapp/static/
--> Using cache
--> 205a0a000000
Step 4/7 : COPY ./templates /home/myapp/templates/
--> Using cache
--> 5a0e3a9d4
Step 5/7 : COPY ./sample_app.py /home/myapp/
--> Using cache
--> 2047150712
Step 6/7 : EXPOSE 8000
--> Using cache
--> 81a3a1e34c7
Step 7/7 : CMD ["python3 /home/myapp/sample_app.py"]
--> Using cache
--> cabb03ebea9
Successfully built cabb03ebea9
Successfully tagged sampleapp:latest
devasc@labvm:~/labs/devnet-src/sample-app$ docker build -t sampleapp .
devasc@labvm:~/labs/devnet-src/sample-app$ docker run -it sampleapp
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://172.17.0.1:8000
<!DOCTYPE html>
<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$ 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 00:00:27:b9:e8:0c brd ff:ff:ff:ff:ff:ff
  inet 10.0.2.1/24 brd 10.0.2.255 scope global dynamic enp0s3
    valid_lft 8633sec preferred_lft 8633sec
  inet6 fe80::200:27ff:feb9:e80c/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 00:00:c7:0d:c8:0c 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 fe00::0:200:c7ff:fe0d:c80c/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 02:42:53:a0:de:00 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
  inet6 fe00::42:53ff:fe0d:a0de/64 scope link
    valid_lft forever preferred_lft forever
124: vethde0d981f23: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state DOWN group default
  link/ether 74:16:bf:40 brd ff:ff:ff:ff:ff:ff link-mtu 1500
  inet6 fe00::74:16ff:fe16:b740/64 scope link
    valid_lft forever preferred_lft forever
devasc@labvm:~/labs/devnet-src/sample-app$ docker exec -it sampleapp /bin/bash
root@5a0e3a9d4:~# ls
bin  dev  home  lib32  lib32  lib  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  shbin  sys  usr
root@5a0e3a9d4:~# ls /home/myapp/
sample_app.py  static  templates
root@5a0e3a9d4:~# exit
exit
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop sampleapprunning
sampleapprunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID  IMAGE      COMMAND   CREATED          STATUS          PORTS     NAMES
5a0e3a9d4:~  sampleapp  "/bin/sh -c 'python3 ./sample_app.py'"  2 minutes ago  Exited (127) 0 seconds ago  sampleapprunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker start sampleapprunning
sampleapprunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker rm sampleapprunning
sampleapprunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID  IMAGE      COMMAND   CREATED          STATUS          PORTS     NAMES
5a0e3a9d4:~  sampleapp  "/bin/sh -c 'python3 ./sample_app.py'"  2 minutes ago  Up 38 seconds  0.0.0.0:8000->8000/tcp, 0.0.0.0:8000->8000/tcp  sampleapprunning
devasc@labvm:~/labs/devnet-src/sample-app$
```

Conclusion

To conclude, this activity helps the student to utilize and explore the DEVASC VM to organize the creation, execution, and verification of a Docker container. It served as a tangible exploration of development and deployment methodologies, commencing with the establishment of the DEVASC VM as the foundation for subsequent tasks. Engaging in the scripting domain showcased a proficiency in automation through crafting a Bash script. The transition into application development marked the birth of a sample web app, and the introduction of complexity involved configuring the web app for seamless integration of website files. The pinnacle of the journey unfolded in Parts 5 and 6, where a Bash script was devised to construct and launch a Docker container encapsulating the web app. Achieving these objectives deepened the understanding of development and scripting, emphasizing the efficacy of containerization in modern application deployment. This hands-on experience significantly enriched the skill set, offering practical insights into the evolving realms of software development and containerized deployment.

Github link: <https://github.com/RioMariee/hoa12.git>