Lab - Build a Sample Web App in a Docker Container

by Seruelas, Ronn Kristoper H. of CPE31S4
in CPE232, submitted to Dr. Jonathan V. Taylar at 11-21-2023

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.

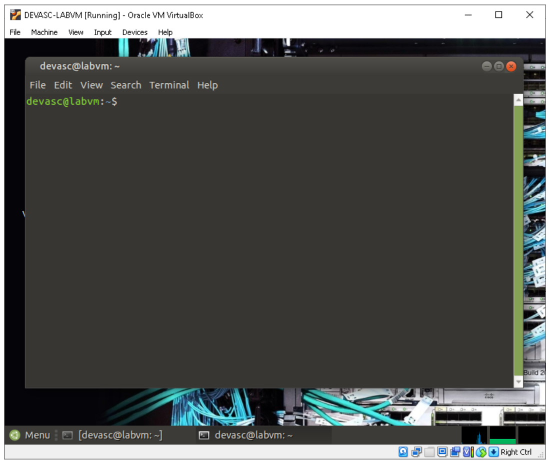
Required Resources

- 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.



```
devasc@labvm:~$
```

Figure 1.1 - DEVASC VM Running successfully.

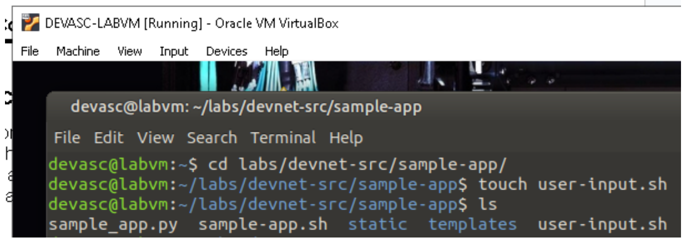
Part 2: Create a Simple Bash Script

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
```



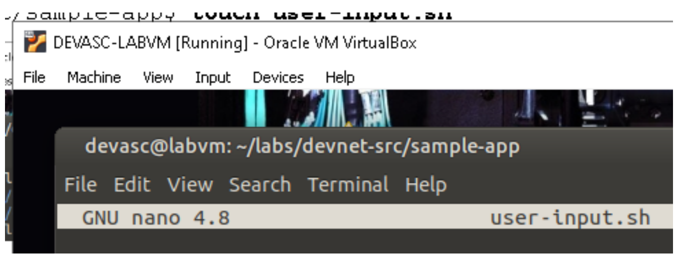
```
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls
sample_app.py  sample-app.sh  static  templates  user-input.sh
```

Figure 2.1.1 - Creation of **user-input.sh** in **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
```



```
  GNU nano 4.8                              user-input.sh
```

Figure 2.2.1 - Using nano to open the editor for **user-input.sh**.

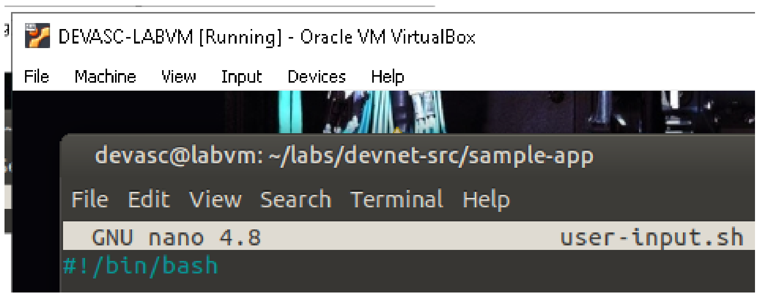
Step 3: Add the 'she-bang' to the top of the script.

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 carat 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.


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

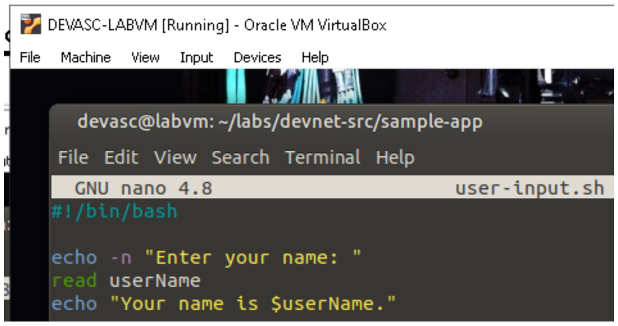
```
  GNU nano 4.8                    user-input.sh
#!/bin/bash
```

Figure 2.3.1 - Adding she-bang on top of the **user-input.sh** script.

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."
```


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

```
  GNU nano 4.8                    user-input.sh
#!/bin/bash

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

Figure 2.4.1 - Adding a simple bash command unto the script.

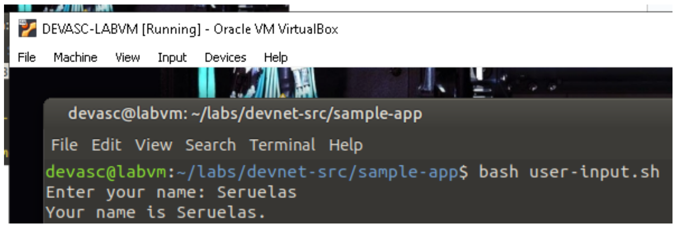
Step 5: Exit nano and save your script.

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 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

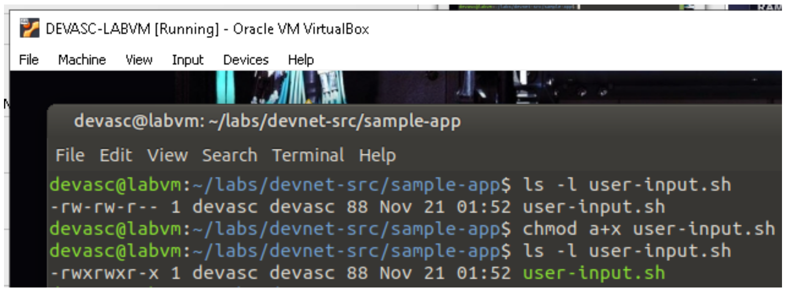
```
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Enter your name: Seruelas
Your name is Seruelas.
```

Figure 2.6.1 - Running the bash script from the terminal.

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
```


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

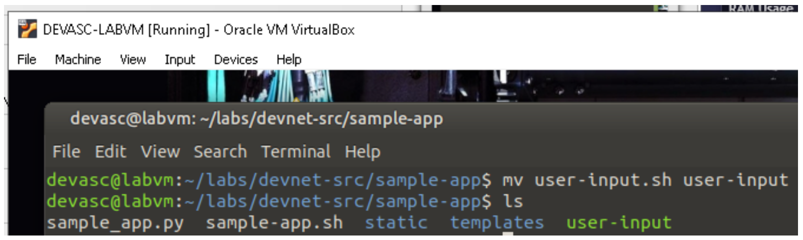
```
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 88 Nov 21 01:52 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 88 Nov 21 01:52 user-input.sh
```

Figure 2.7.1 - Changing mode or permission to an executable file for all users.

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.

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


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

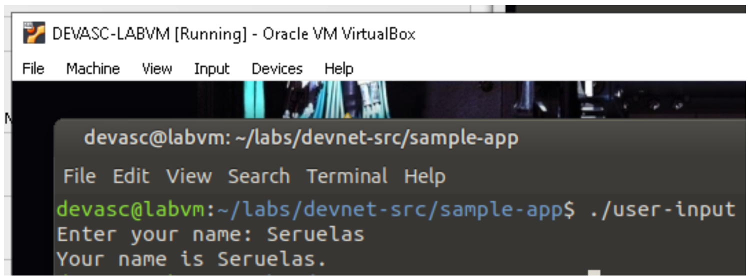
```
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
devasc@labvm:~/labs/devnet-src/sample-app$ ls
sample_app.py  sample-app.sh  static  templates  user-input
```

Figure 2.8.1 - Renaming the script file to remove the extension of the script.

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$
```


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:~/labs/devnet-src/sample-app$ ./user-input
Enter your name: Seruelas
Your name is Seruelas.
```

Figure 2.9.1 - Execution of renamed script from the terminal.

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.

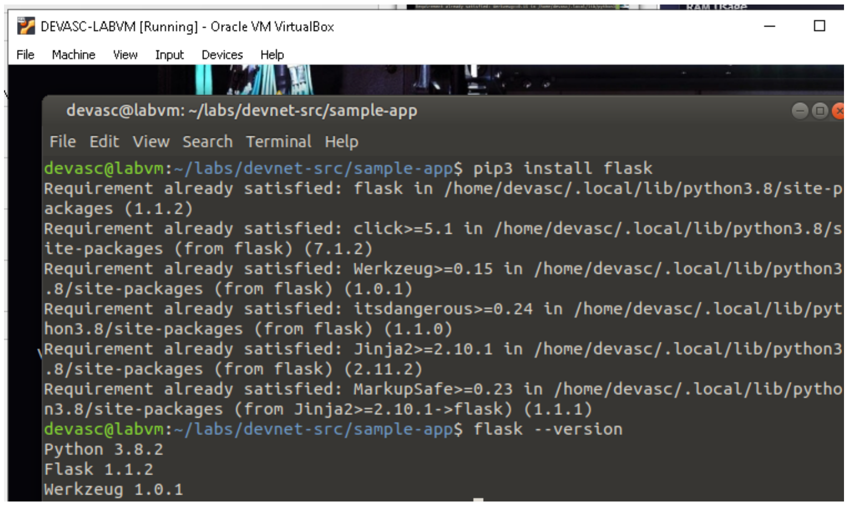
Web application developers using Python typically leverage a framework. A framework is a code library to make it easier for developers to create reliable, scalable and maintainable web applications. Flask is a web application framework written in Python. Other frameworks include Tornado and Pyramid.

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 [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:~/labs/devnet-src/sample-app$ pip3 install flask
Requirement already satisfied: flask in /home/devasc/.local/lib/python3.8/site-p
ackages (1.1.2)
Requirement already satisfied: click>=5.1 in /home/devasc/.local/lib/python3.8/s
ite-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/pyt
hon3.8/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/pytho
n3.8/site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
devasc@labvm:~/labs/devnet-src/sample-app$ flask --version
Python 3.8.2
Flask 1.1.2
Werkzeug 1.0.1
```

Figure 3.1.1 - Importing or installation of **flask** via pip3.

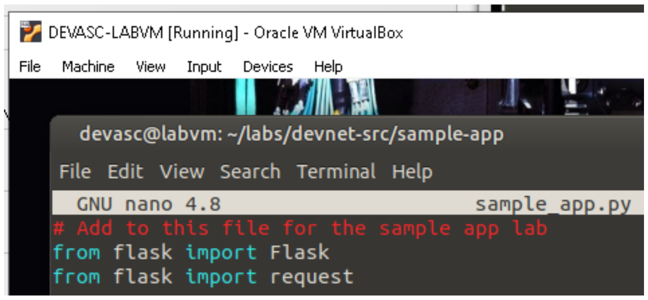
Step 2: **Open the sample_app.py file.**

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**.

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
```



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
GNU nano 4.8
sample_app.py

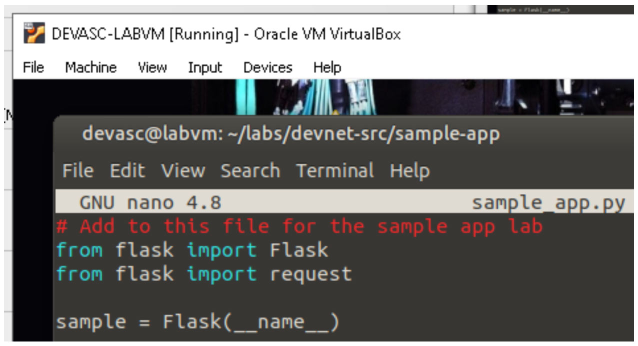
```
# Add to this file for the sample app lab
from flask import Flask
from flask import request
```

Figure 3.3.1 - Importing methods from Flask.

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__)
```


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

```
GNU nano 4.8                          sample_app.py
# Add to this file for the sample app lab
from flask import Flask
from flask import request

sample = Flask(__name__)
```

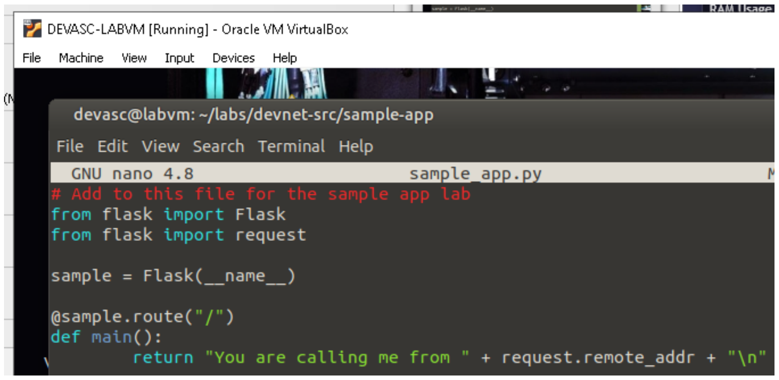
Figure 3.4.1 - Creation of the Flask class named sample.

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"
```

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.


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

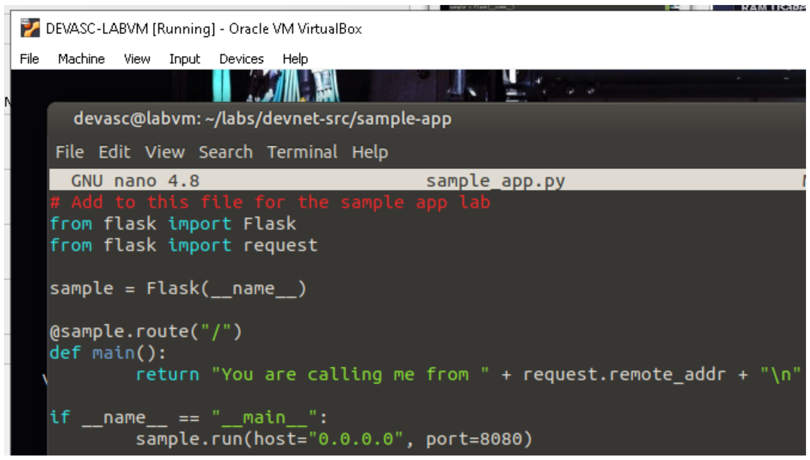
```
GNU nano 4.8                    sample_app.py
# Add to this file for the sample app lab
from flask import Flask
from flask import request

sample = Flask(__name__)

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

Figure 3.5.1 - Defining a method to display the client IP address.

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)
```


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
GNU nano 4.8 sample_app.py

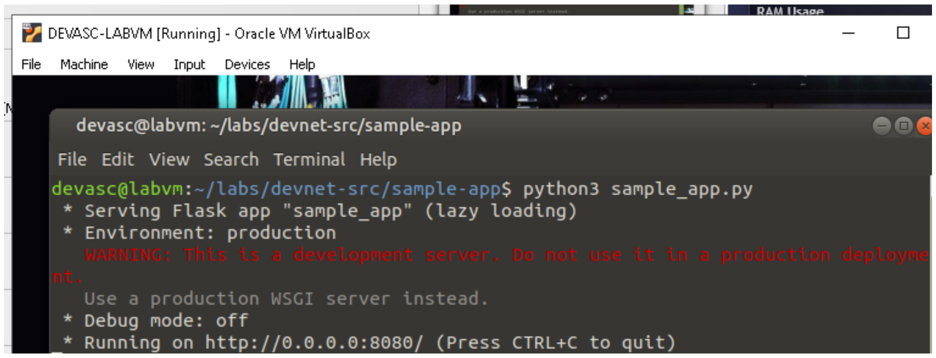
```
# Add to this file for the sample app lab
from flask import Flask
from flask import request

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)
```

Figure 3.6.1 - Configuration of the app to run locally.

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)
```


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

Figure 3.7.1 - Running the sample_app.py on a web app locally.

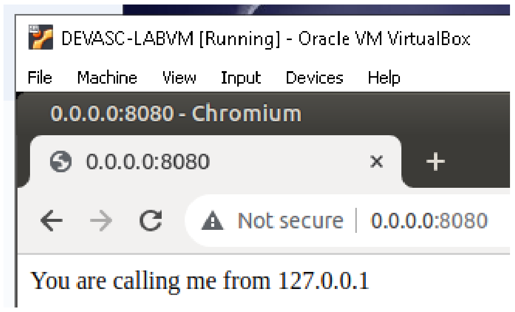
Step 8: Verify the server is running.

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

a. 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.

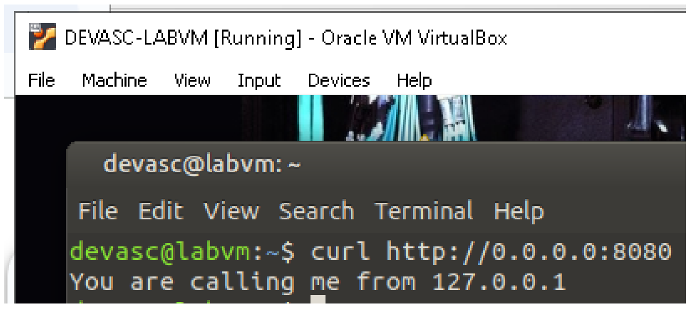

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
0.0.0.0:8080 - Chromium
0.0.0.0:8080
Not secure | 0.0.0.0:8080

You are calling me from 127.0.0.1

Figure 3.8.1 - Verification of web server running on 0.0.0.0:8080 via Chromium Browser.

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 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help

```
devasc@labvm:~$ curl http://0.0.0.0:8080
You are calling me from 127.0.0.1
```

Figure 3.8.2 - Verification of web server running locally through cURL.

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

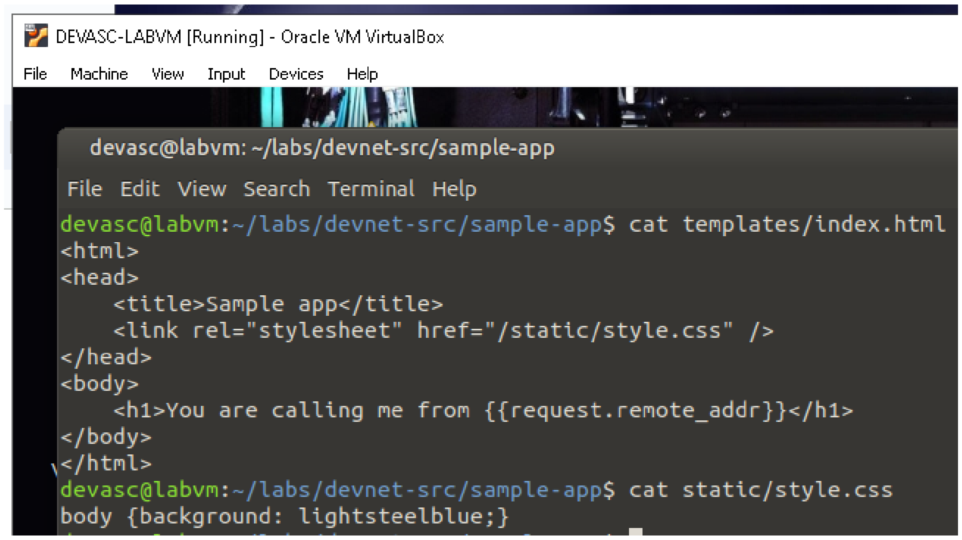
In this part, build out the sample web app to include an **index.html** page and **style.css** specification. The **index.html** is typically the first page loaded in a client's web browser when visiting your website. The **style.css** is a style sheet used to customize the look of the web page.

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:~/labs/devnet-src/sample-app$
```


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

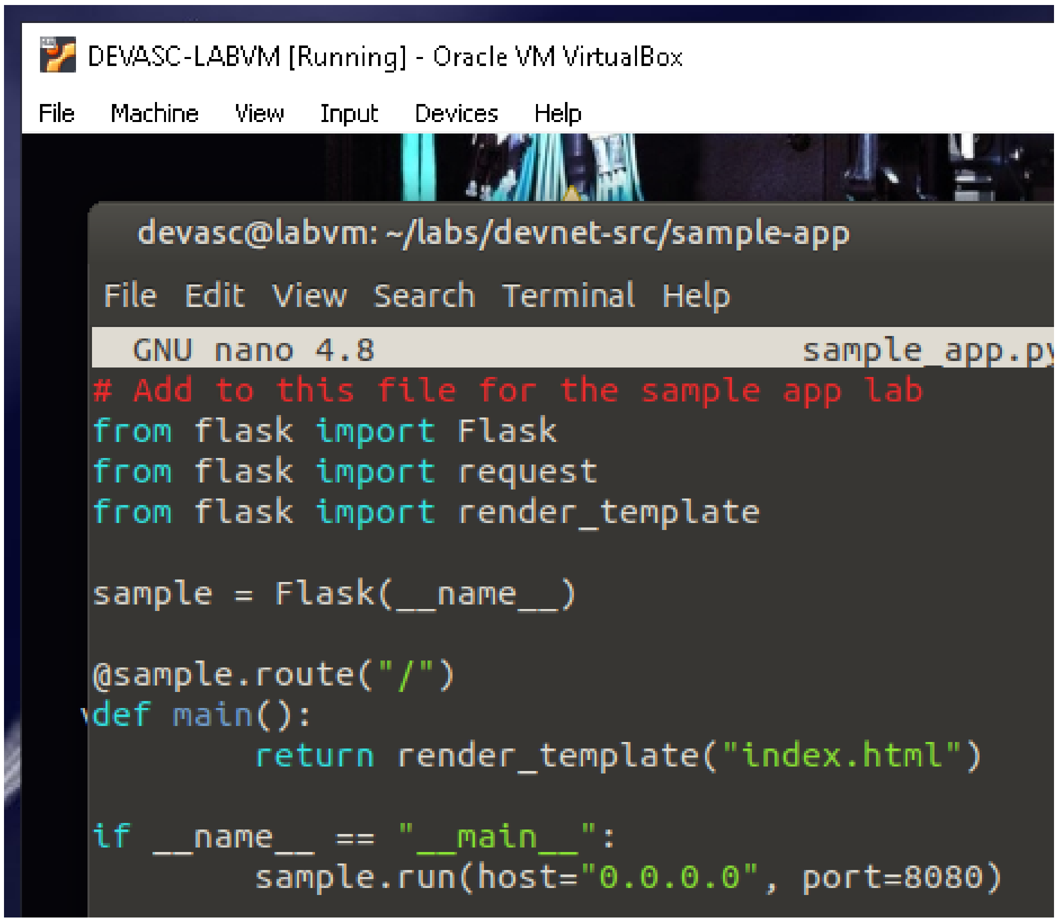
```
devasc@labvm:~/labs/devnet-src/sample-app$ cat templates/index.html
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>You are calling me from {{request.remote_addr}}</h1>
</body>
</html>
devasc@labvm:~/labs/devnet-src/sample-app$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

Figure 4.1.1 - Exploration of the index.html and style.css of the sample-all directory.

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

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


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
GNU nano 4.8 sample_app.py

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

sample = Flask(__name__)

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

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

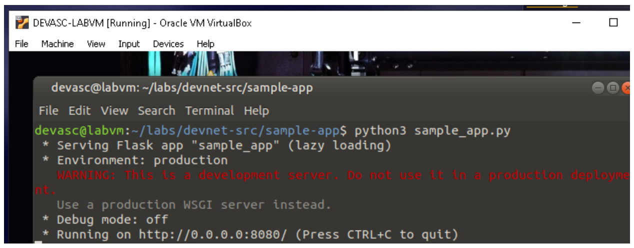
Figure 4.2.1 - Modification of the sample_app.py

Step 3: **Save and run your script.**

Save and run your **sampe-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)
```

Note: If you got Traceback output and an error with the message with something like **OSError: [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.


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

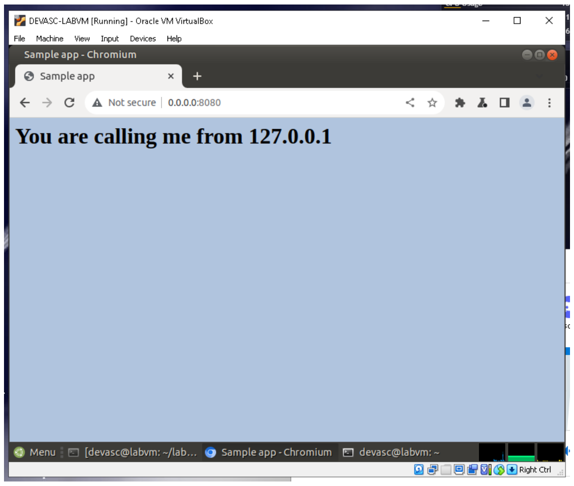
Figure 4.3.1 - Running of modified sample_app.py.

Step 4: **Verify your program is running.**

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

a. 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

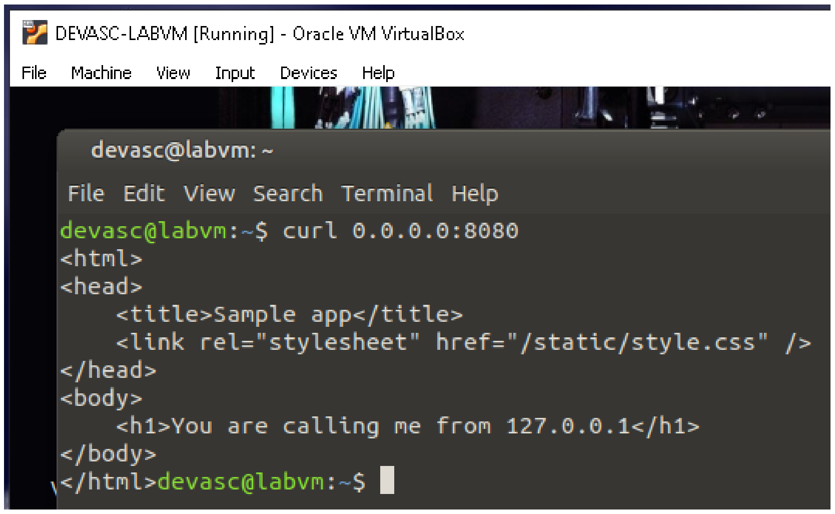

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
Sample app
Not secure | 0.0.0.0:8080
Menu
[devasc@labvm: ~/lab...
Sample app - Chromium
devasc@labvm: ~
Right Ctrl

You are calling me from 127.0.0.1

Figure 4.4.1 - Verification of web server running at **0.0.0.0:8080** via Chromium Browser.

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


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help

```
devasc@labvm:~$ curl 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:~$
```

Figure 4.4.2 - Verification of web server running through cURL.

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

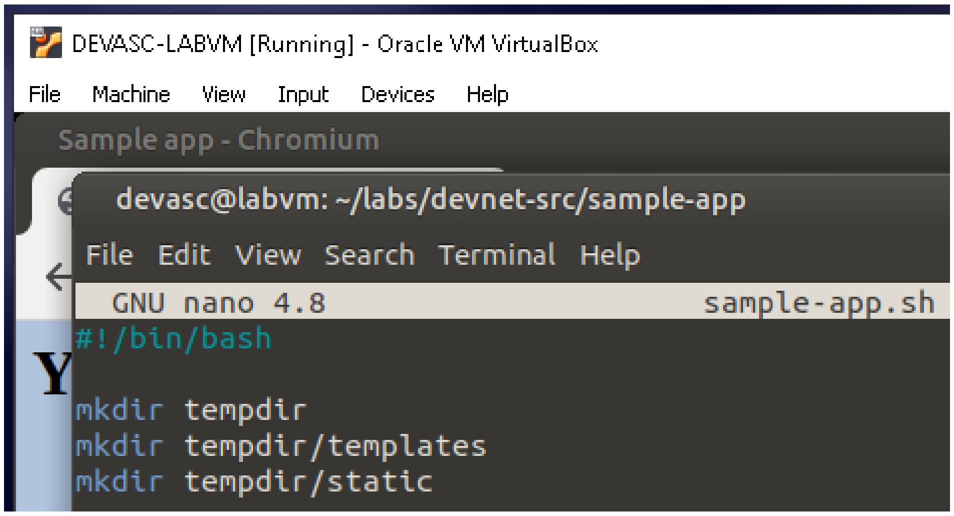
An application can be deployed on a bare metal server (physical server dedicated to a single-tenant environment) or in a virtual machine, like you just did in the previous Part. It can also be deployed in a containerized solution like Docker. In this part, you will create a bash script and add commands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


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 sample-app.sh

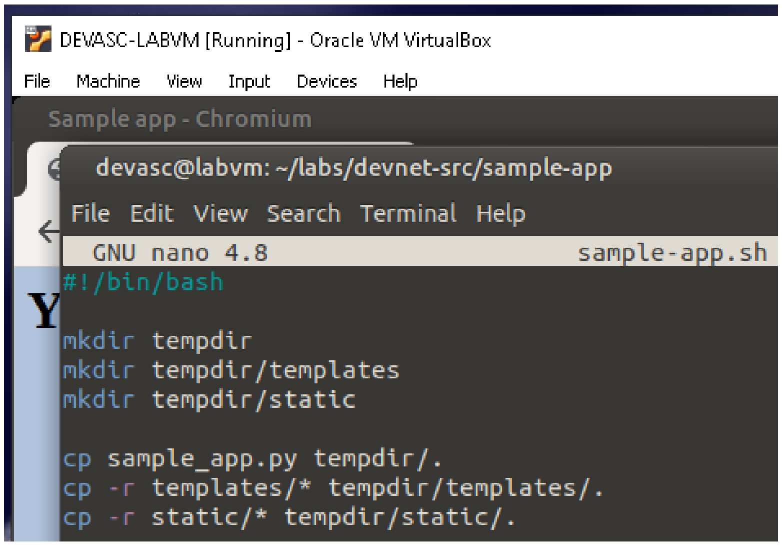
```
#!/bin/bash

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

Figure 5.1.1 - Modification of sample-app.sh to create directories with tempdir as parent folder.

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


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
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/.
```

Figure 5.2.1 - Modification of sample-app.sh to copy website directory and script to tempdir.

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.

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

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

b. 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
```

c. 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.

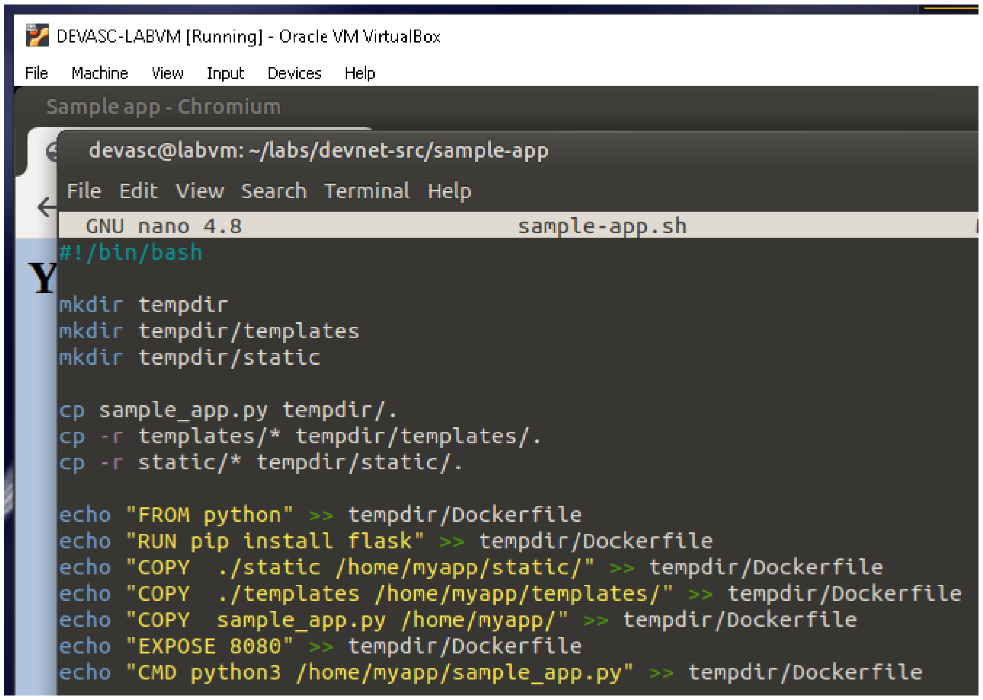
```
echo "COPY  ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY  ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY  sample_app.py /home/myapp/" >> tempdir/Dockerfile
```

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
```


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 sample-app.sh

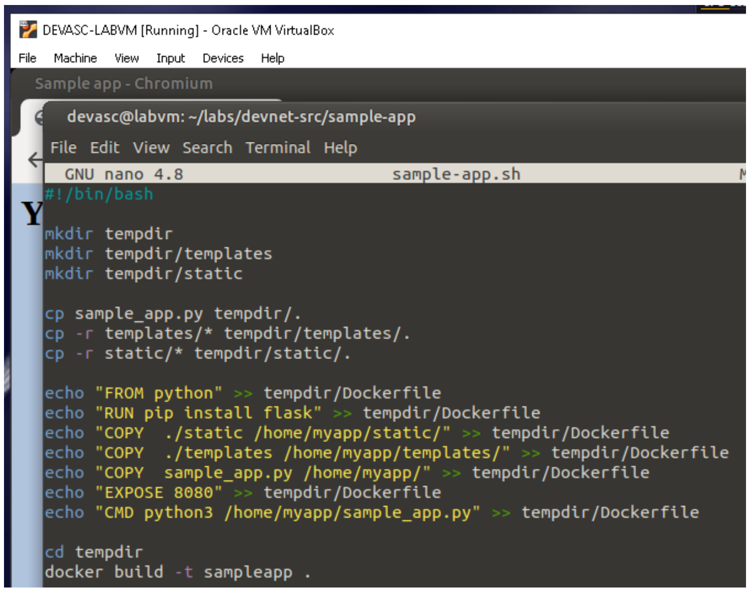
```
#!/bin/bash

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
```

Figure 5.3.1 - Creation of the Dockerfile using the sample-app.sh script.

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 .
```


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
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 .
```

Figure 5.4.1 - Adding the command to build the docker container in the tempdir.

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.

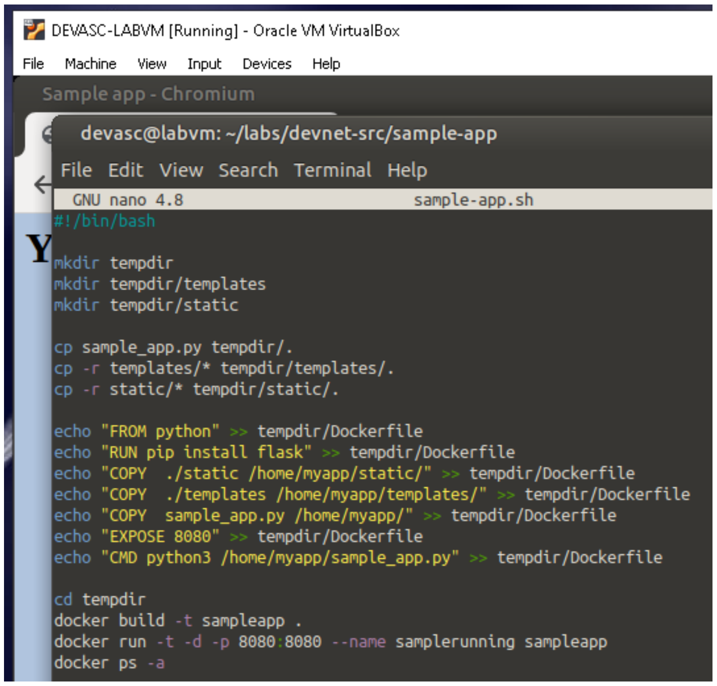
```
docker run -t -d -p 8080:8080 --name samplerunning sampleapp
```

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

- **-t** specifies that you want a terminal created for the container so the 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
```


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
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
```

Figure 5.5.1 - Adding the command to run the container and displaying the running container via sample-app.sh script.

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.

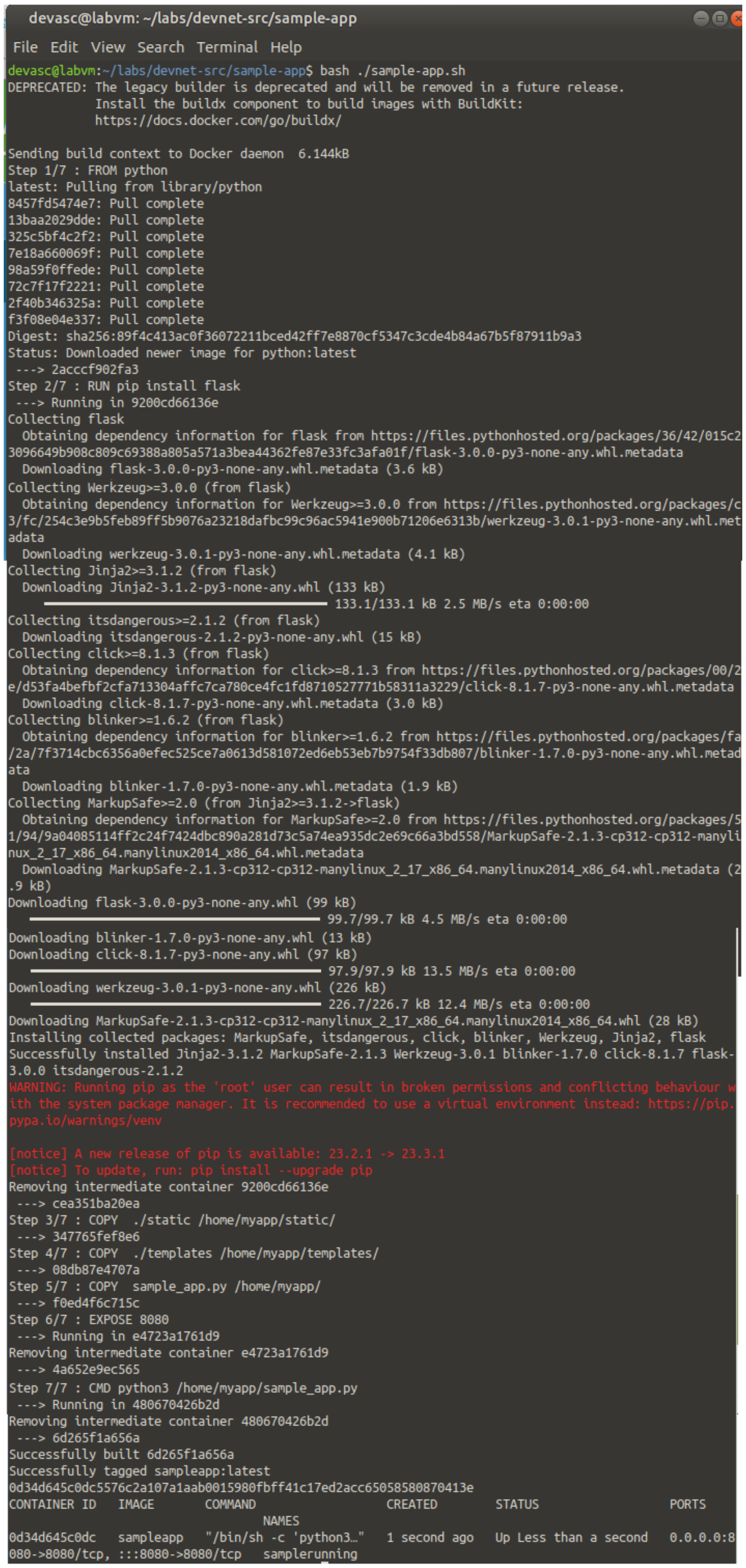
Note: Be sure you stopped any other web server processes you may still have running from the previous parts of this lab.

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.

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



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

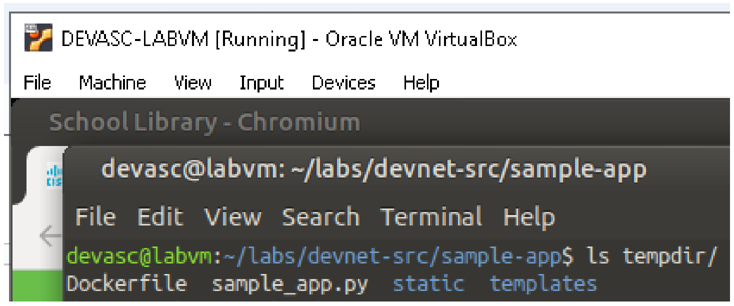
```
devasc@labvm:~/labs/devnet-src/sample-app$ bash ./sample-app.sh
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
latest: Pulling from library/python
8457fd5474e7: Pull complete
13baa2029dde: Pull complete
325c5bf4c2f2: Pull complete
7e18a660069f: Pull complete
98a59f0ffede: Pull complete
72c7f17f2221: Pull complete
2f40b346325a: Pull complete
f3f08e04e337: Pull complete
Digest: sha256:89f4c413ac0f36072211bced42ff7e8870cf5347c3cde4b84a67b5f87911b9a3
Status: Downloaded newer image for python:latest
 ---> 2acccf902fa3
Step 2/7 : RUN pip install flask
 ---> Running in 9200cd66136e
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c2
3096649b908c809c69388a805a571a3bea44362fe87e33fc3afa01f/flask-3.0.0-py3-none-any.whl.metadata
  Downloading flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Obtaining dependency information for Werkzeug>=3.0.0 from https://files.pythonhosted.org/packages/c
3/fc/254c3e9b5feb89ff5b9076a23218dafbc99c96ac5941e900b71206e6313b/werkzeug-3.0.1-py3-none-any.whl.met
adata
  Downloading werkzeug-3.0.1-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.1/133.1 kB 2.5 MB/s eta 0:00:00
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/00/2
e/d53fa4befbf2cfa713304affc7ca780ce4fc1fd8710527771b58311a3229/click-8.1.7-py3-none-any.whl.metadata
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Obtaining dependency information for blinker>=1.6.2 from https://files.pythonhosted.org/packages/fa
/2a/7f3714cbc6356a0efec525ce7a0613d581072ed6eb53eb7b9754f33db807/blinker-1.7.0-py3-none-any.whl.metad
ata
  Downloading blinker-1.7.0-py3-none-any.whl.metadata (1.9 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Obtaining dependency information for MarkupSafe>=2.0 from https://files.pythonhosted.org/packages/5
1/94/9a04085114ff2c24f7424dbc890a281d73c5a74ea935dc2e69c66a3bd558/MarkupSafe-2.1.3-cp312-cp312-manyli
nux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Downloading MarkupSafe-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2
.9 kB)
Downloading flask-3.0.0-py3-none-any.whl (99 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.7/99.7 kB 4.5 MB/s eta 0:00:00
Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Downloading click-8.1.7-py3-none-any.whl (97 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 97.9/97.9 kB 13.5 MB/s eta 0:00:00
Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.7/226.7 kB 12.4 MB/s eta 0:00:00
Downloading MarkupSafe-2.1.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (28 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 flask-
3.0.0 itsdangerous-2.1.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour w
ith the system package manager. It is recommended to use a virtual environment instead: https://pip.
pypa.io/warnings/venv

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: pip install --upgrade pip
Removing intermediate container 9200cd66136e
 ---> cea351ba20ea
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> 347765fef8e6
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> 08db87e4707a
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> f0ed4f6c715c
Step 6/7 : EXPOSE 8080
 ---> Running in e4723a1761d9
Removing intermediate container e4723a1761d9
 ---> 4a652e9ec565
Step 7/7 : CMD python3 /home/myapp/sample_app.py
 ---> Running in 480670426b2d
Removing intermediate container 480670426b2d
 ---> 6d265f1a656a
Successfully built 6d265f1a656a
Successfully tagged sampleapp:latest
0d34d645c0dc5576c2a107a1aab0015980fbff41c17ed2acc65058580870413e
CONTAINER ID   IMAGE       COMMAND                  CREATED        STATUS                  PORTS
                                      NAMES
0d34d645c0dc   sampleapp   "/bin/sh -c 'python3…"   1 second ago   Up Less than a second   0.0.0.0:8
080->8080/tcp, :::8080->8080/tcp   samplerunning
```

Figure 6.1.1 - Execution of sample-app.sh bash script.

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

a. 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$
```

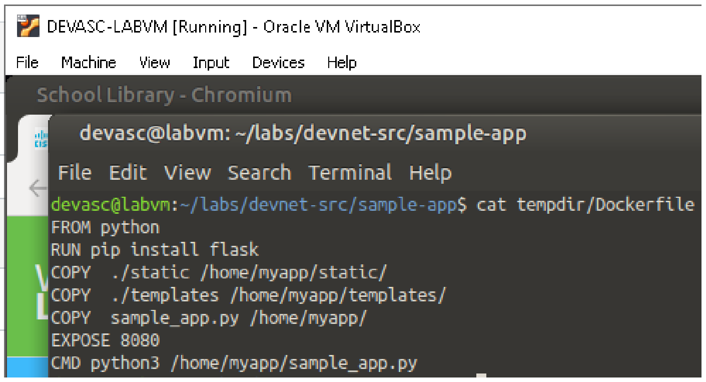

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
School Library - Chromium
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

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

Figure 6.2.1 - Verification of the tempdir directory creation.

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
```

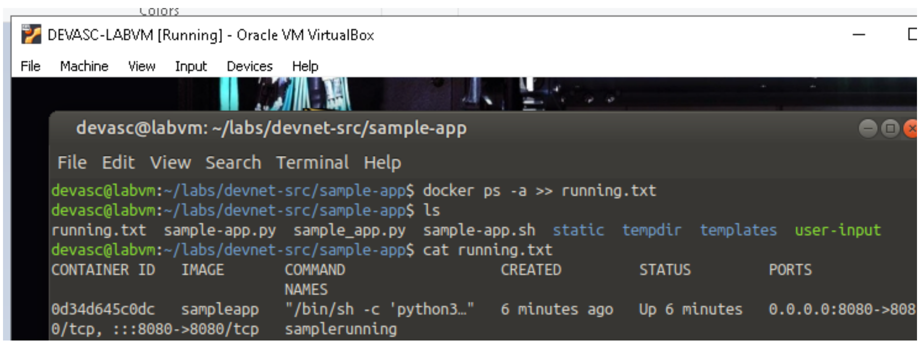

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
School Library - Chromium
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

```
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
```

Figure 6.2.2 - Verification of the Dockerfile in tempdir.

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$
```

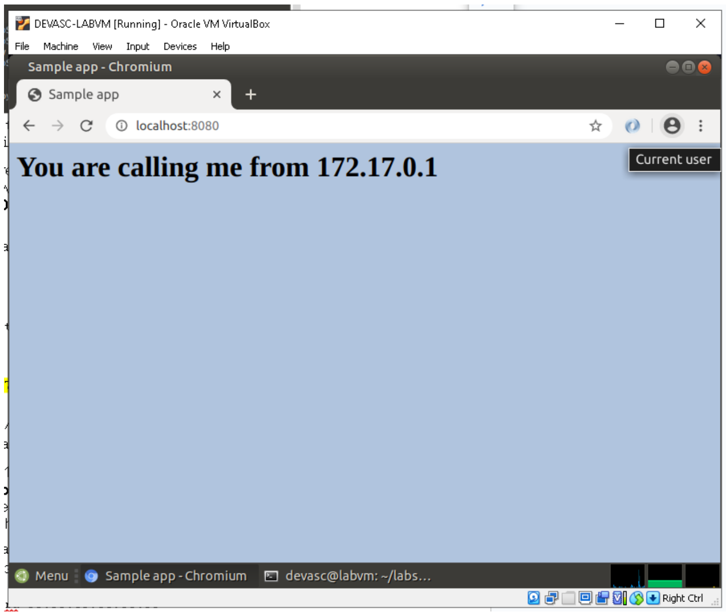


```
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/labs/devnet-src/sample-app$ ls
running.txt  sample-app.py  sample_app.py  sample-app.sh  static  tempdir  templates  user-input
devasc@labvm:~/labs/devnet-src/sample-app$ cat running.txt
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORTS
               NAMES
0d34d645c0dc   sampleapp    "/bin/sh -c 'python3…"   6 minutes ago   Up 6 minutes   0.0.0.0:8080->808
0/tcp, :::8080->8080/tcp    samplerunning
```

Figure 6.2.3 - Redirecting the output of the command *docker ps -a* file to *running.txt*, then verifying output of running.txt.

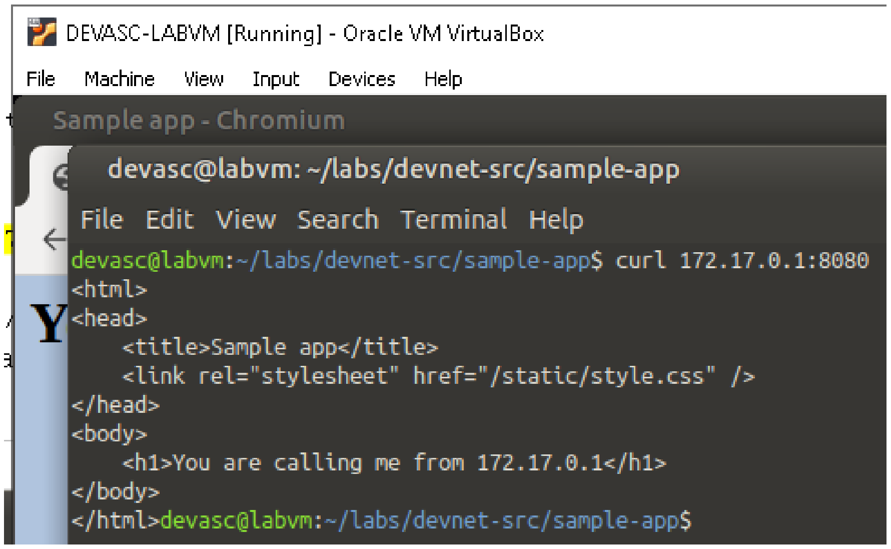
d. The Docker container creates its own IP address from a private network address space. Verify the web app is running and reporting the IP address. In a web browser at **http://localhost:8080**, you should see the message **You are calling me from 172.17.0.1** formatted as H1 on a light steel blue background. 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$
devasc@labvm:~/labs/devnet-src/sample-app$
```



You are calling me from 172.17.0.1

Figure 6.2.4 - Verification of the web server running in container at **localhost:8080** via chromium.

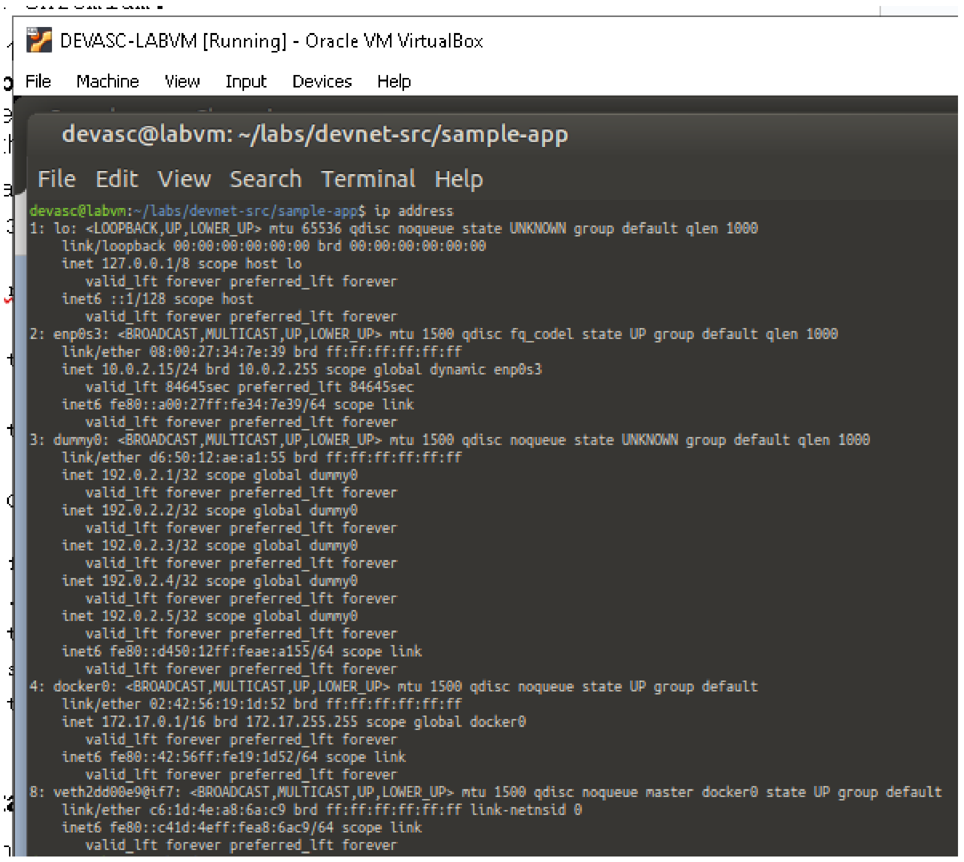

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sample app - Chromium
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

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

Figure 6.2.5 - Verification of the web server running in container at **localhost:8080** via Chromium.

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


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:~/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 08:00:27:34:7e:39 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 84645sec preferred_lft 84645sec
    inet6 fe80::a00:27ff:fe34:7e39/64 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 d6:50:12:ae:a1:55 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::d450:12ff:feae:a155/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:56:19:1d:52 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 fe80::42:56ff:fe19:1d52/64 scope link
       valid_lft forever preferred_lft forever
8: veth2dd00e9@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether c6:1d:4e:a8:6a:c9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::c41d:4eff:fea8:6ac9/64 scope link
       valid_lft forever preferred_lft forever
```

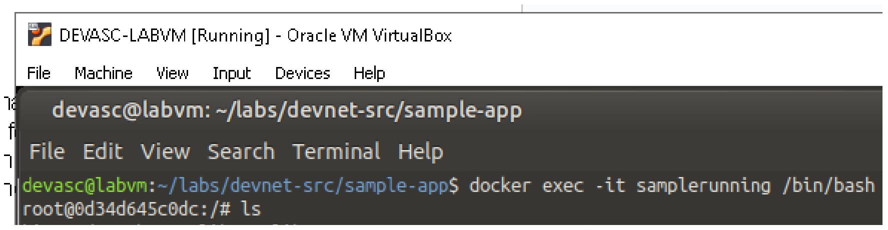
Figure 6.2.6 - Verification of the web server running in container via cURL.

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.

a. 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:/#
```

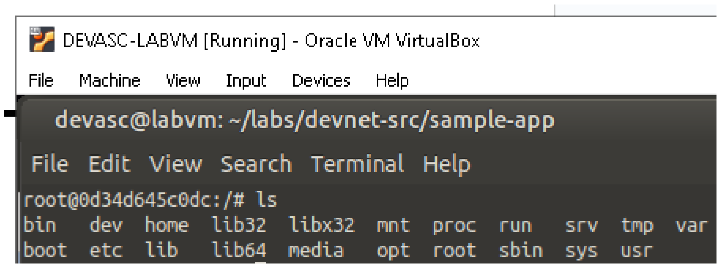

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@0d34d645c0dc:/# ls
```

Figure 6.3.1 - Accessing the docker container.

b. 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:/#
```

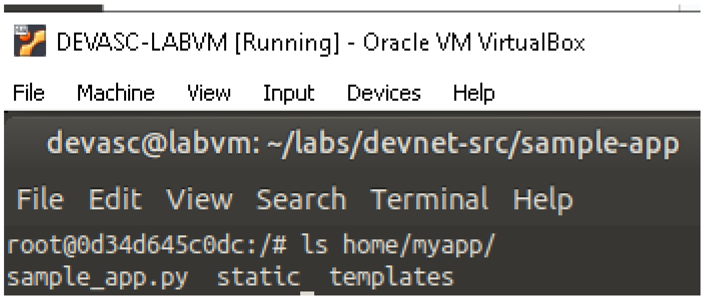

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

```
root@0d34d645c0dc:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

Figure 6.3.2 - Execution of ls to see directory structure of container.

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:/#
```

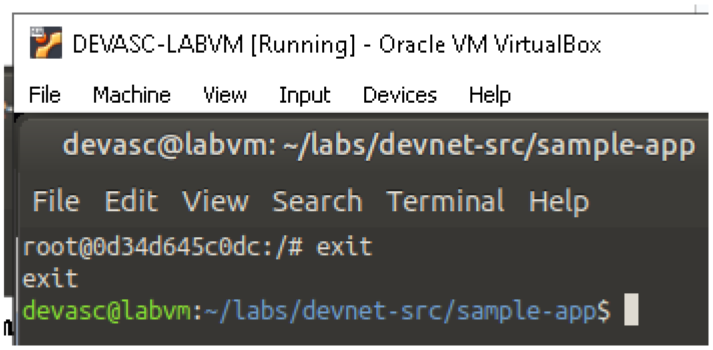

DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help

```
root@0d34d645c0dc:/# ls home/myapp/
sample_app.py  static  templates
```

Figure 6.3.3 - Verification of files in the **home/myapp/** directory.

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

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


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

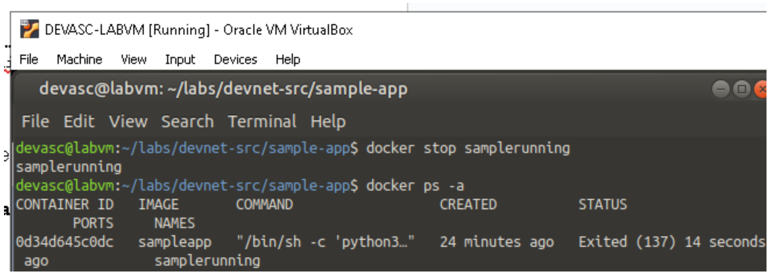
```
root@0d34d645c0dc:/# exit
exit
devasc@labvm:~/labs/devnet-src/sample-app$
```

Figure 6.3.4 - Exiting the container to return to DEVASC VM.

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

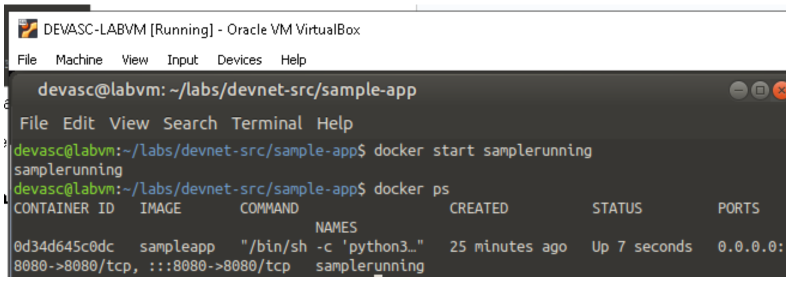


```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
       PORTS     NAMES
0d34d645c0dc   sampleapp   "/bin/sh -c 'python3…"   24 minutes ago   Exited (137) 14 seconds
 ago             samplerunning
```

Figure 6.4.1 - Stopping the docker container.

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$
```

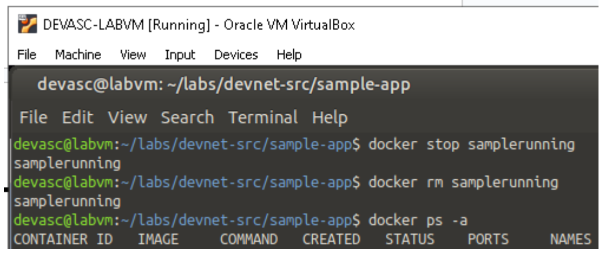


```
devasc@labvm:~/labs/devnet-src/sample-app$ docker start samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS         PORTS
                                      NAMES
0d34d645c0dc   sampleapp   "/bin/sh -c 'python3…"   25 minutes ago   Up 7 seconds   0.0.0.0:
8080->8080/tcp, :::8080->8080/tcp   samplerunning
```

Figure 6.4.2 - Starting the docker **samplerunning** container.

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


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

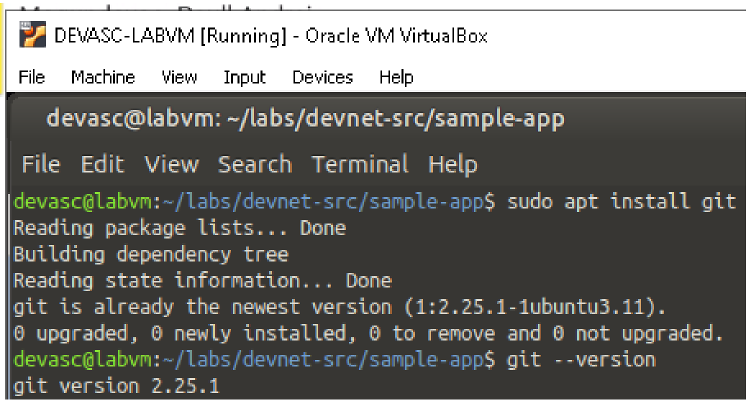
```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker rm samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Figure 6.4.3 - Removing the container.

Part 7: **Saving all work done to the GitHub Repository.**

Step 1: Install Git in the DEVASC VM.

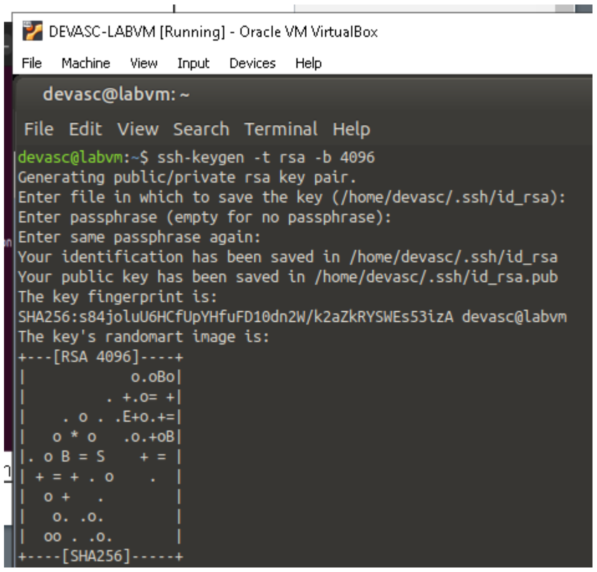
1. Install git by using *sudo apt install git* in the DEVASC VM.


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:~/labs/devnet-src/sample-app$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.25.1-1ubuntu3.11).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
devasc@labvm:~/labs/devnet-src/sample-app$ git --version
git version 2.25.1
```

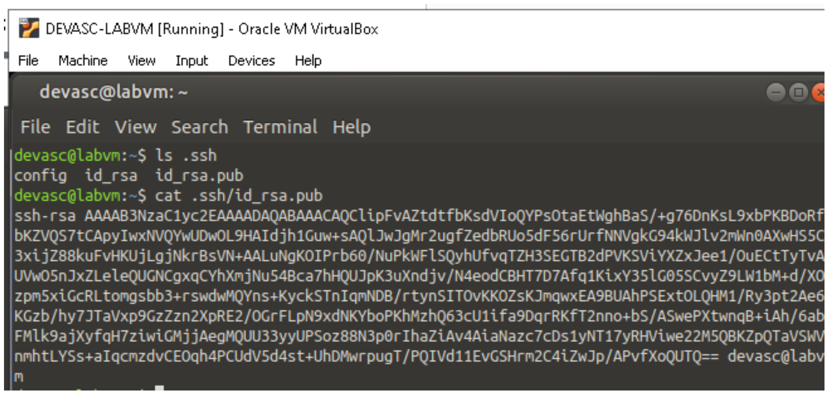
Figure 7.1.1 - Installation and verification of git in DEVASC VM.

2. Generate a public RSA key that will be used to connect the DEVASC VM to the GitHub account.


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help

```
devasc@labvm:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/devasc/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/devasc/.ssh/id_rsa
Your public key has been saved in /home/devasc/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:s84joluU6HCfUpYHfuFD10dn2W/k2aZkRYSWEs53izA devasc@labvm
The key's randomart image is:
+---[RSA 4096]----+
|            o.oBo|
|         . +.o= +|
|     . o . .E+o.+=|
|   o * o   .o.+oB|
|. o B = S    + = |
| + = + . o    .  |
|  o +   .        |
|   o. .o.        |
|  oo . .o.       |
+----[SHA256]-----+
```

Figure 7.2.1 - Creation of RSA Public Key for DEVASC VM.

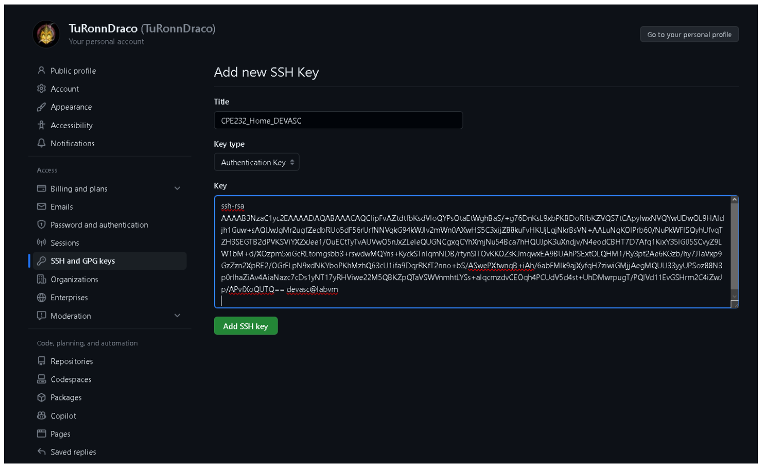


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help

```
devasc@labvm:~$ ls .ssh
config  id_rsa  id_rsa.pub
devasc@labvm:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQClipFvAZtdtfbKsdVIoQYPsOtaEtWghBaS/+g76DnKsL9xbPKBDoRf
bKZVQS7tCApyIwxNVQYwUDwOL9HAIdjh1Guw+sAQlJwJgMr2ugfZedbRUo5dF56rUrfNNVgkG94kWJlv2mWn0AXwHS5C
3xijZ88kuFvHKUjLgjNkrBsVN+AALuNgKOIPrb60/NuPkWFlSQyhUfvqTZH3SEGTB2dPVKSViYXZxJee1/OuECtTyTvA
UVwO5nJxZLeleQUGNCgxqCYhXmjNu54Bca7hHQUJpK3uXndjv/N4eodCBHT7D7Afq1KixY35lG05SCvyZ9LW1bM+d/XO
zpm5xiGcRLtomgsbb3+rswdwMQYns+KyckSTnIqmNDB/rtynSITOvKKOZsKJmqwxEA9BUAhPSExtOLQHM1/Ry3pt2Ae6
KGzb/hy7JTaVxp9GzZzn2XpRE2/OGrFLpN9xdNKYboPKhMzhQ63cU1ifa9DqrRKfT2nno+bS/ASwePXtwnqB+iAh/6ab
FMlk9ajXyfqH7ziwiGMjjAegMQUU33yyUPSoz88N3p0rIhaZiAv4AiaNazc7cDs1yNT17yRHViwe22M5QBKZpQTaVSWV
nmhtLYSs+aIqcmzdvCEOqh4PCUdV5d4st+UhDMwrpugT/PQIVd11EvGSHrm2C4iZwJp/APvfXoQUTQ== devasc@labv
m
```

Figure 7.2.2 - Verification of RSA Public Key creation for DEVASC VM.

3. Create a GPG/SSH key to connect the GitHub to the DEVASC VM.



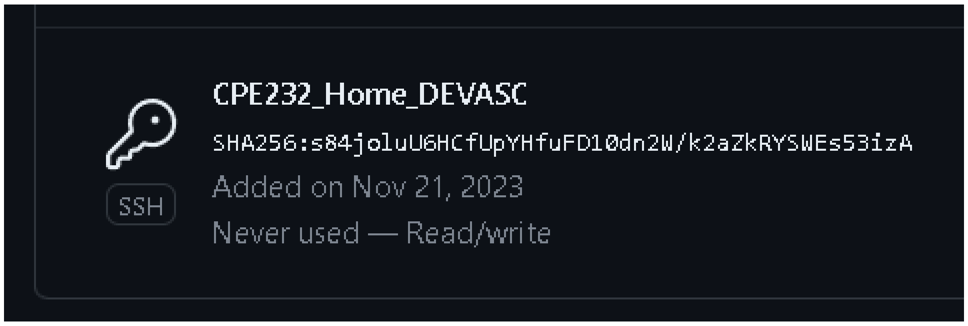
Title: CPE232_Home_DEVASC

Key type: Authentication Key

Key:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQClipFvAZtdtfbKsdVIoQYPsOtaEtWghBaS/+g76DnKsL9xbPKBDoRfbKZVQS7tCApyIwxNVQYwUDwOL9HAId
jh1Guw+sAQlJwJgMr2ugfZedbRUo5dF56rUrfNNVgkG94kWJlv2mWn0AXwHS5C3xijZ88kuFvHKUjLgjNkrBsVN+AALuNgKOIPrb60/NuPkWFlSQyhUfvqT
ZH3SEGTB2dPVKSViYXZxJee1/OuECtTyTvAUVwO5nJxZLeleQUGNCgxqCYhXmjNu54Bca7hHQUJpK3uXndjv/N4eodCBHT7D7Afq1KixY35lG05SCvyZ9L
W1bM+d/XOzpm5xiGcRLtomgsbb3+rswdwMQYns+KyckSTnIqmNDB/rtynSITOvKKOZsKJmqwxEA9BUAhPSExtOLQHM1/Ry3pt2Ae6KGzb/hy7JTaVxp9
GzZzn2XpRE2/OGrFLpN9xdNKYboPKhMzhQ63cU1ifa9DqrRKfT2nno+bS/ASwePXtwnqB+iAh/6abFMlk9ajXyfqH7ziwiGMjjAegMQUU33yyUPSoz88N3
p0rIhaZiAv4AiaNazc7cDs1yNT17yRHViwe22M5QBKZpQTaVSWVnmhtLYSs+aIqcmzdvCEOqh4PCUdV5d4st+UhDMwrpugT/PQIVd11EvGSHrm2C4iZwJ
p/APvfXoQUTQ== devasc@labvm
```

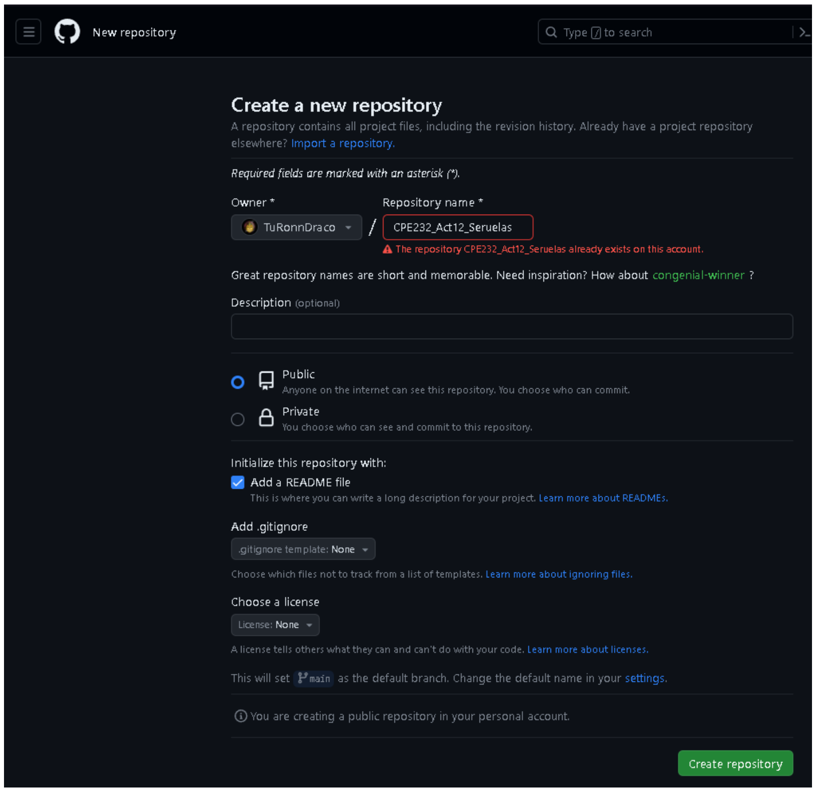
Figure 7.3.1 - Creation of GPG/SSH Key in GitHub using public key of DEVASC VM.



CPE232_Home_DEVASC
SHA256:s84joluU6HCfUpYHfuFD10dn2W/k2aZkRYSWEs53izA
Added on Nov 21, 2023
Never used — Read/write

Figure 7.3.2 - Verification of creation of GPG/SSH Key in GitHub.

4. Create a new repository dedicated for the activity.



New repository

Type / to search

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * TuRonnDraco / Repository name * CPE232_Act12_Seruelas

The repository CPE232_Act12_Seruelas already exists on this account.

Great repository names are short and memorable. Need inspiration? How about congenial-winner ?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

[✓ Add a README file] This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None

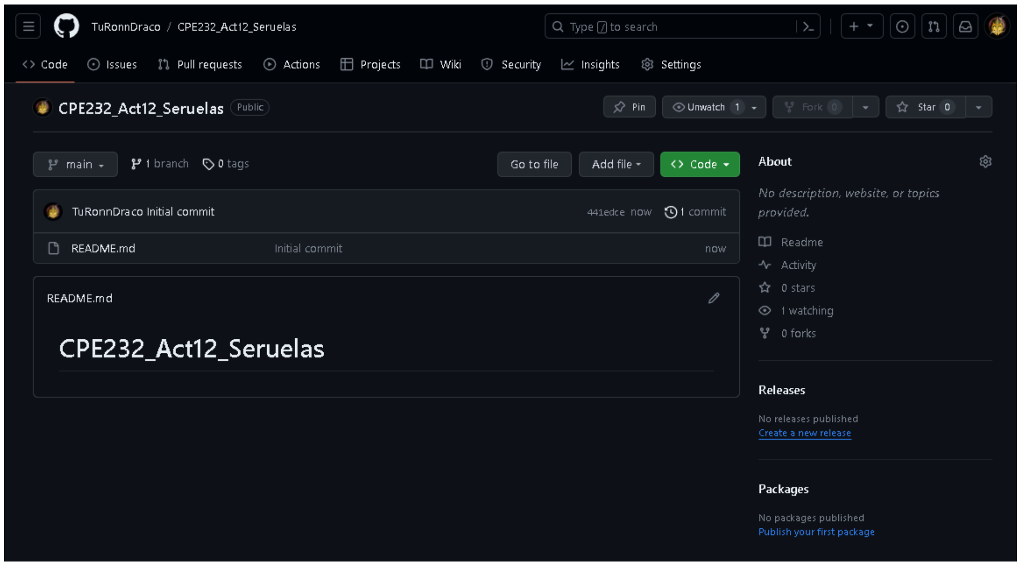
A license tells others what they can and can't do with your code. Learn more about licenses.

This will set main as the default branch. Change the default name in your settings.

You are creating a public repository in your personal account.

Create repository

Figure 7.4.1 - Creation of the CPE232_Act12_Seruelas repository.



TuRonnDraco / CPE232_Act12_Seruelas

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

CPE232_Act12_Seruelas Public

Pin Unwatch 1 Fork 0 Star 0

main 1 branch 0 tags Go to file Add file Code

TuRonnDraco Initial commit 441edce now 1 commit

README.md Initial commit now

README.md

CPE232_Act12_Seruelas

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Figure 7.4.2 - Verification of the creation of the CPE232_Act12_Seruelas repository.

5. Clone the GitHub repository to the local workstation or the DEVASC VM.

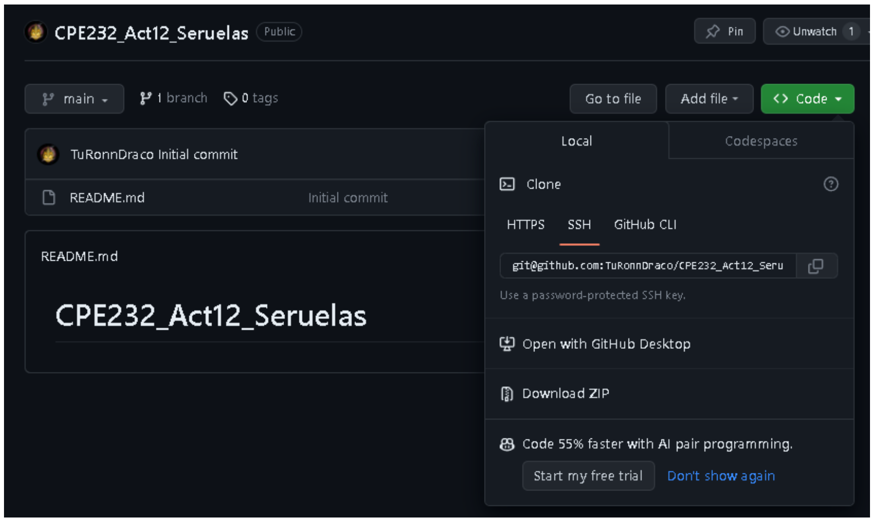
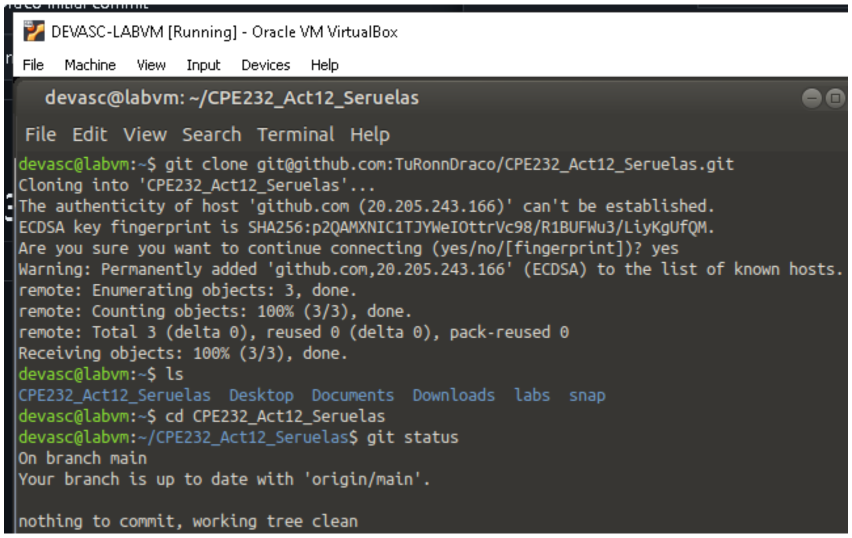


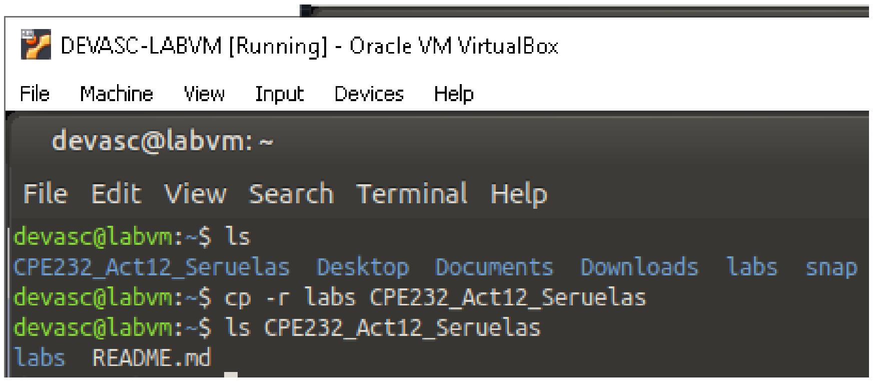
Figure 7.5.1 - Getting the SSH key for cloning the GitHub repository to the DEVASC VM.


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/CPE232_Act12_Seruelas
File Edit View Search Terminal Help

```
devasc@labvm:~$ git clone git@github.com:TuRonnDraco/CPE232_Act12_Seruelas.git
Cloning into 'CPE232_Act12_Seruelas'...
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,20.205.243.166' (ECDSA) to the list of known hosts.
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
devasc@labvm:~$ ls
CPE232_Act12_Seruelas  Desktop  Documents  Downloads  labs  snap
devasc@labvm:~$ cd CPE232_Act12_Seruelas
devasc@labvm:~/CPE232_Act12_Seruelas$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Figure 7.5.2 - Cloning the GitHub repository to the DEVASC VM.

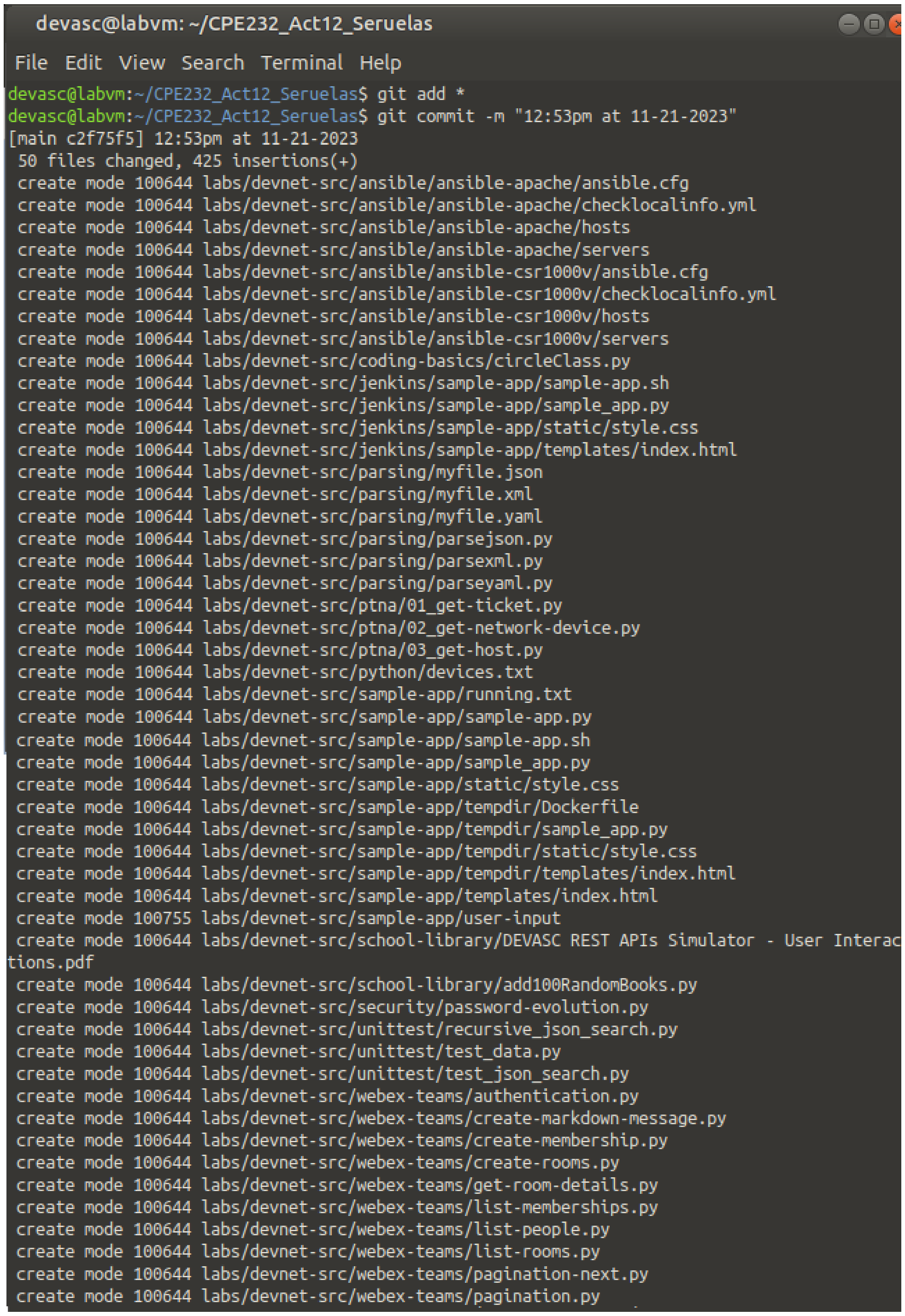
6. Copy all files done via ***cp -r labs <local repository>***.


DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help

```
devasc@labvm:~$ ls
CPE232_Act12_Seruelas  Desktop  Documents  Downloads  labs  snap
devasc@labvm:~$ cp -r labs CPE232_Act12_Seruelas
devasc@labvm:~$ ls CPE232_Act12_Seruelas
labs  README.md
```

Figure 7.6.1 - Copying all files created/done in DEVASC VM to the local repository.

7. Push the local repository to the GitHub repository to save all files created/done.

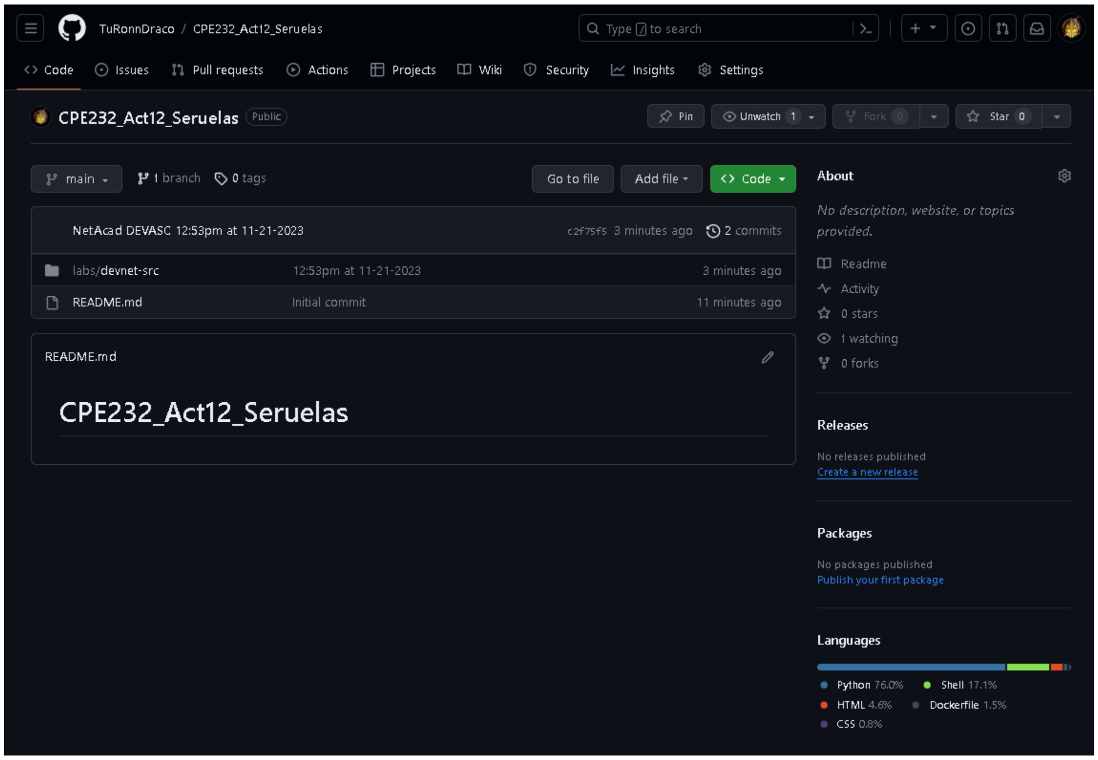


devasc@labvm: ~/CPE232_Act12_Seruelas
File Edit View Search Terminal Help

```
devasc@labvm:~/CPE232_Act12_Seruelas$ git add *
devasc@labvm:~/CPE232_Act12_Seruelas$ git commit -m "12:53pm at 11-21-2023"
[main c2f75f5] 12:53pm at 11-21-2023
 50 files changed, 425 insertions(+)
 create mode 100644 labs/devnet-src/ansible/ansible-apache/ansible.cfg
 create mode 100644 labs/devnet-src/ansible/ansible-apache/checklocalinfo.yml
 create mode 100644 labs/devnet-src/ansible/ansible-apache/hosts
 create mode 100644 labs/devnet-src/ansible/ansible-apache/servers
 create mode 100644 labs/devnet-src/ansible/ansible-csr1000v/ansible.cfg
 create mode 100644 labs/devnet-src/ansible/ansible-csr1000v/checklocalinfo.yml
 create mode 100644 labs/devnet-src/ansible/ansible-csr1000v/hosts
 create mode 100644 labs/devnet-src/ansible/ansible-csr1000v/servers
 create mode 100644 labs/devnet-src/coding-basics/circleClass.py
 create mode 100644 labs/devnet-src/jenkins/sample-app/sample-app.sh
 create mode 100644 labs/devnet-src/jenkins/sample-app/sample_app.py
 create mode 100644 labs/devnet-src/jenkins/sample-app/static/style.css
 create mode 100644 labs/devnet-src/jenkins/sample-app/templates/index.html
 create mode 100644 labs/devnet-src/parsing/myfile.json
 create mode 100644 labs/devnet-src/parsing/myfile.xml
 create mode 100644 labs/devnet-src/parsing/myfile.yaml
 create mode 100644 labs/devnet-src/parsing/parsejson.py
 create mode 100644 labs/devnet-src/parsing/parsexml.py
 create mode 100644 labs/devnet-src/parsing/parseyaml.py
 create mode 100644 labs/devnet-src/ptna/01_get-ticket.py
 create mode 100644 labs/devnet-src/ptna/02_get-network-device.py
 create mode 100644 labs/devnet-src/ptna/03_get-host.py
 create mode 100644 labs/devnet-src/python/devices.txt
 create mode 100644 labs/devnet-src/sample-app/running.txt
 create mode 100644 labs/devnet-src/sample-app/sample-app.py
 create mode 100644 labs/devnet-src/sample-app/sample-app.sh
 create mode 100644 labs/devnet-src/sample-app/sample_app.py
 create mode 100644 labs/devnet-src/sample-app/static/style.css
 create mode 100644 labs/devnet-src/sample-app/tempdir/Dockerfile
 create mode 100644 labs/devnet-src/sample-app/tempdir/sample_app.py
 create mode 100644 labs/devnet-src/sample-app/tempdir/static/style.css
 create mode 100644 labs/devnet-src/sample-app/tempdir/templates/index.html
 create mode 100644 labs/devnet-src/sample-app/templates/index.html
 create mode 100755 labs/devnet-src/sample-app/user-input
 create mode 100644 labs/devnet-src/school-library/DEVASC REST APIs Simulator - User Interac
tions.pdf
 create mode 100644 labs/devnet-src/school-library/add100RandomBooks.py
 create mode 100644 labs/devnet-src/security/password-evolution.py
 create mode 100644 labs/devnet-src/unittest/recursive_json_search.py
 create mode 100644 labs/devnet-src/unittest/test_data.py
 create mode 100644 labs/devnet-src/unittest/test_json_search.py
 create mode 100644 labs/devnet-src/webex-teams/authentication.py
 create mode 100644 labs/devnet-src/webex-teams/create-markdown-message.py
 create mode 100644 labs/devnet-src/webex-teams/create-membership.py
 create mode 100644 labs/devnet-src/webex-teams/create-rooms.py
 create mode 100644 labs/devnet-src/webex-teams/get-room-details.py
 create mode 100644 labs/devnet-src/webex-teams/list-memberships.py
 create mode 100644 labs/devnet-src/webex-teams/list-people.py
 create mode 100644 labs/devnet-src/webex-teams/list-rooms.py
 create mode 100644 labs/devnet-src/webex-teams/pagination-next.py
 create mode 100644 labs/devnet-src/webex-teams/pagination.py
```

Figure 7.7.1 - Pushing all created files/changes done in the GitHub repository (was not able to include pushing of the local repository but it was successful, verified via GitHub)

8. Verify all committed files in GitHub.



CPE232_Act12_Seruelas Public

1 branch · 0 tags

NetAcad DEVASC 12:53pm at 11-21-2023 · c2f75f5 3 minutes ago · 2 commits

labs/devnet-src	12:53pm at 11-21-2023	3 minutes ago
README.md	Initial commit	11 minutes ago

README.md

CPE232_Act12_Seruelas

About

No description, website, or topics provided.

Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

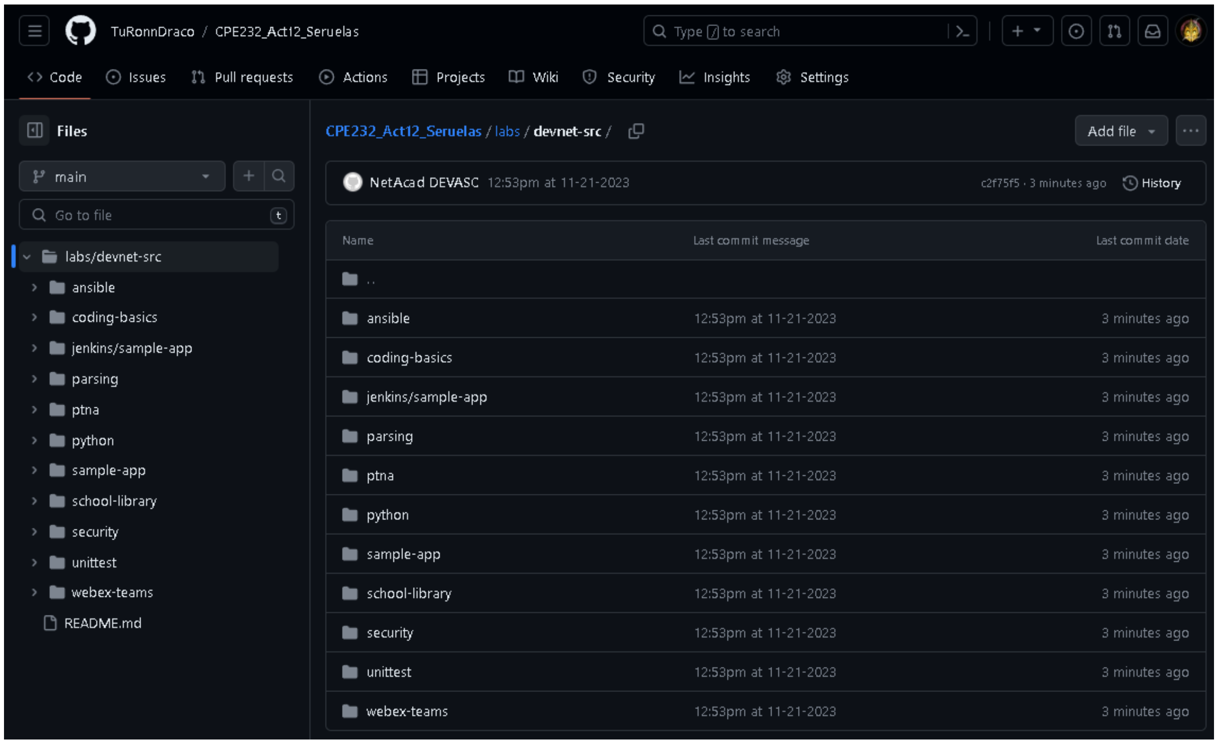
Packages

No packages published
Publish your first package

Languages

● Python 76.0%	● Shell 17.1%
● HTML 4.6%	● Dockerfile 1.5%
● CSS 0.8%	

Figure 7.8.1 - Verification of committed files in GitHub.



- labs/devnet-src
  - ansible
  - coding-basics
  - jenkins/sample-app
  - parsing
  - ptna
  - python
  - sample-app
  - school-library
  - security
  - unittest
  - webex-teams
- README.md

CPE232_Act12_Seruelas / labs / devnet-src /

NetAcad DEVASC 12:53pm at 11-21-2023 · c2f75f5 · 3 minutes ago

Name	Last commit message	Last commit date
..		
ansible	12:53pm at 11-21-2023	3 minutes ago
coding-basics	12:53pm at 11-21-2023	3 minutes ago
jenkins/sample-app	12:53pm at 11-21-2023	3 minutes ago
parsing	12:53pm at 11-21-2023	3 minutes ago
ptna	12:53pm at 11-21-2023	3 minutes ago
python	12:53pm at 11-21-2023	3 minutes ago
sample-app	12:53pm at 11-21-2023	3 minutes ago
school-library	12:53pm at 11-21-2023	3 minutes ago
security	12:53pm at 11-21-2023	3 minutes ago
unittest	12:53pm at 11-21-2023	3 minutes ago
webex-teams	12:53pm at 11-21-2023	3 minutes ago

Figure 7.8.2 - Verification of all committed files in GitHub (Directory form)

Conclusion:

- To conclude this activity, we were able to use another operating system, DEVASC VM, in the creation of a sample web app in a Docker Container. By using bash script, html, and python language, we were able to create a sample web app and were able to successfully display it via browser or by verifying it by using the cURL command. We were able to create a bash script that focuses on the creation of a dockerfile by using a bash script, and were able to create a dockerfile that contains a web server, hosting the sample web app. To conclude this activity, we were able to successfully create a docker container that hosts a web server that is verified through the browser of the DEVASC VM.