

Laxamana, Abigail - CPE31S6

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.

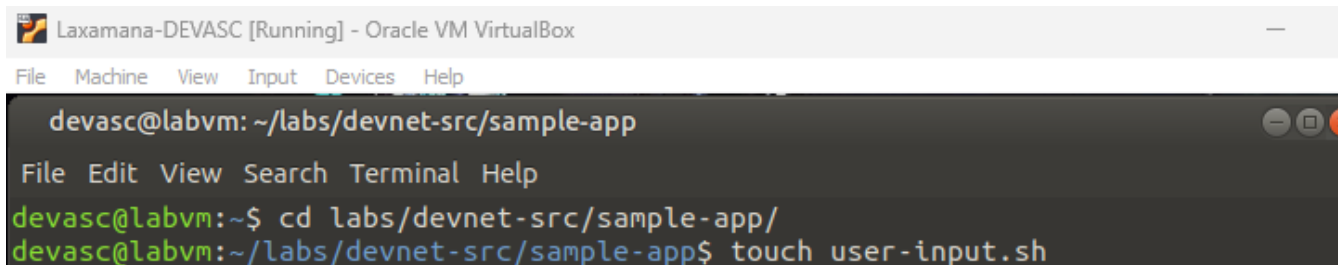
Part 2: Create a Simple Bash Script

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
```

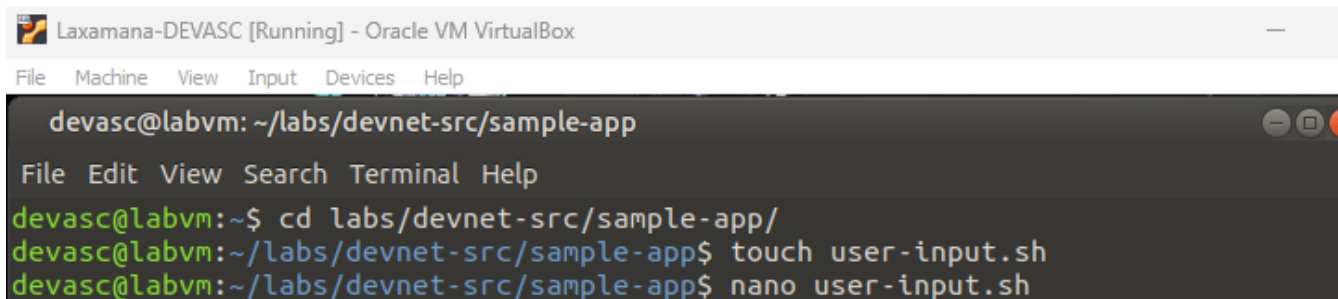
A screenshot of a terminal window titled "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". The terminal shows the user navigating to the directory `~/labs/devnet-src/sample-app` and running the command `touch user-input.sh` to create a new file.

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

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
```

A screenshot of a terminal window titled "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". The terminal shows the user navigating to the directory `~/labs/devnet-src/sample-app`, running `touch user-input.sh`, and then running `nano user-input.sh` to open the file in the nano text editor.

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

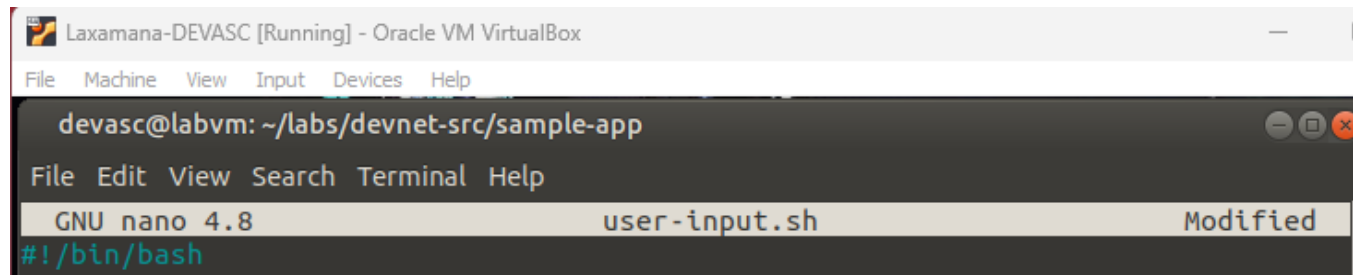
Step 3: Add the 'she-bang' to the top of the script.

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.

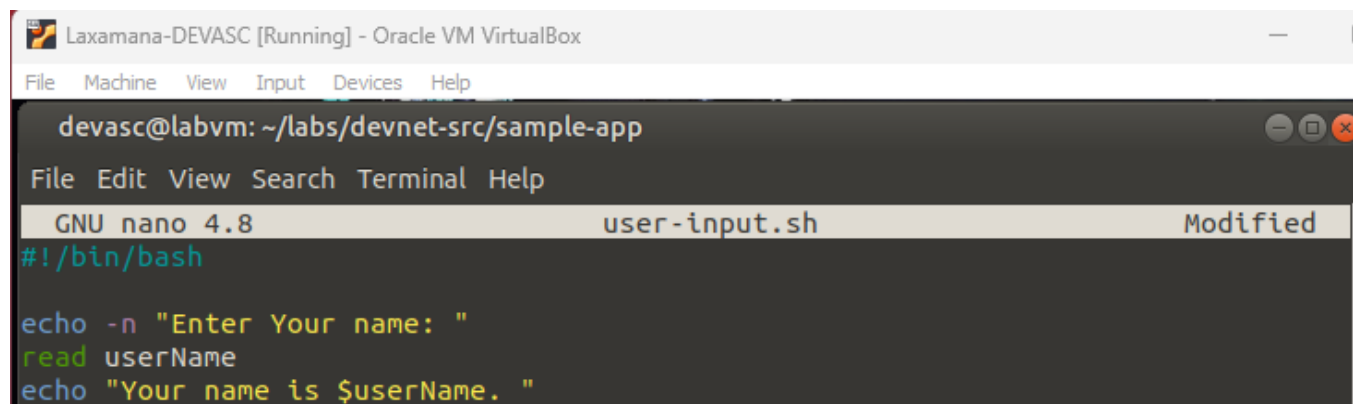


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 user-input.sh Modified
#!/bin/bash
```

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."
```



```
Laxamana-DEVASC [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 Modified
#!/bin/bash
echo -n "Enter Your name: "
read userName
echo "Your name is $userName. "
```

Step 5: Exit nano and save your script.

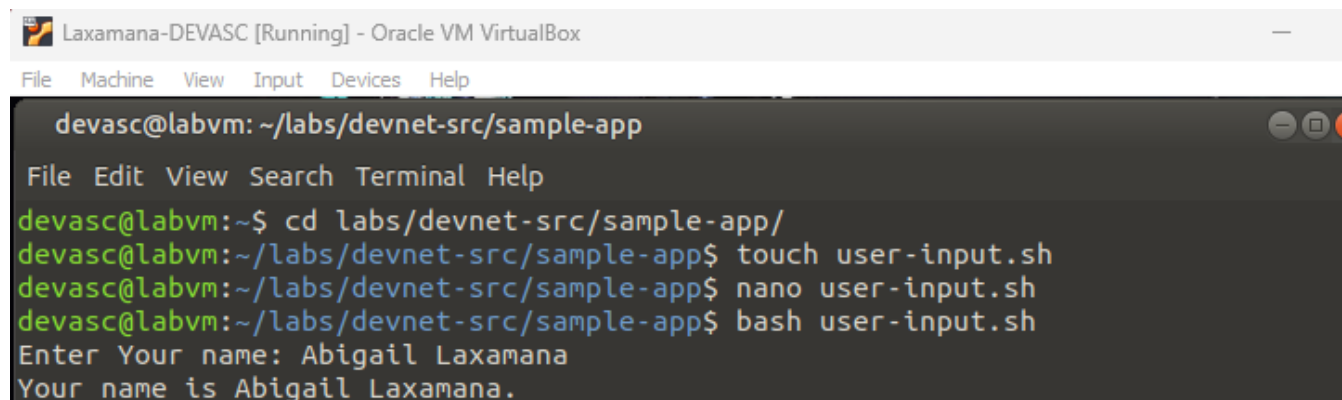
Press **CTRL+X**, then **Y**, then **ENTER** to exit nano and save your script.

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$
```

Lab - Build a Sample Web App in a Docker Container



```
Laxamana-DEVASC [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/
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: Abigail Laxamana
Your name is Abigail Laxamana.
```

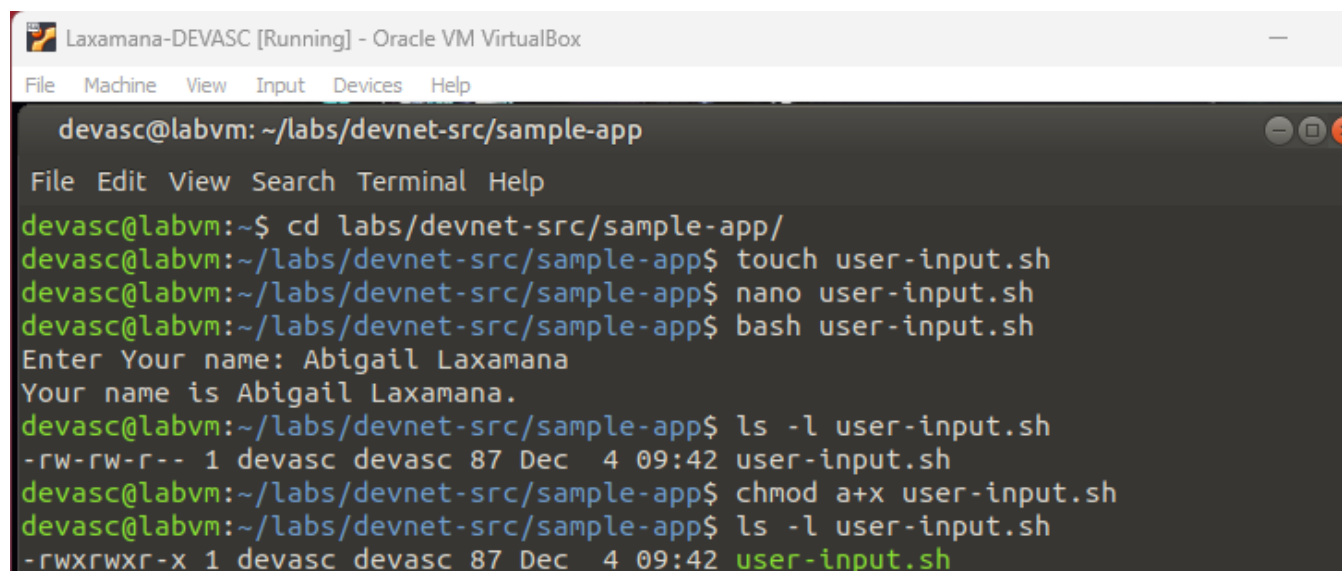
Step 7: Change the mode of the script to an executable file for all users.

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
```



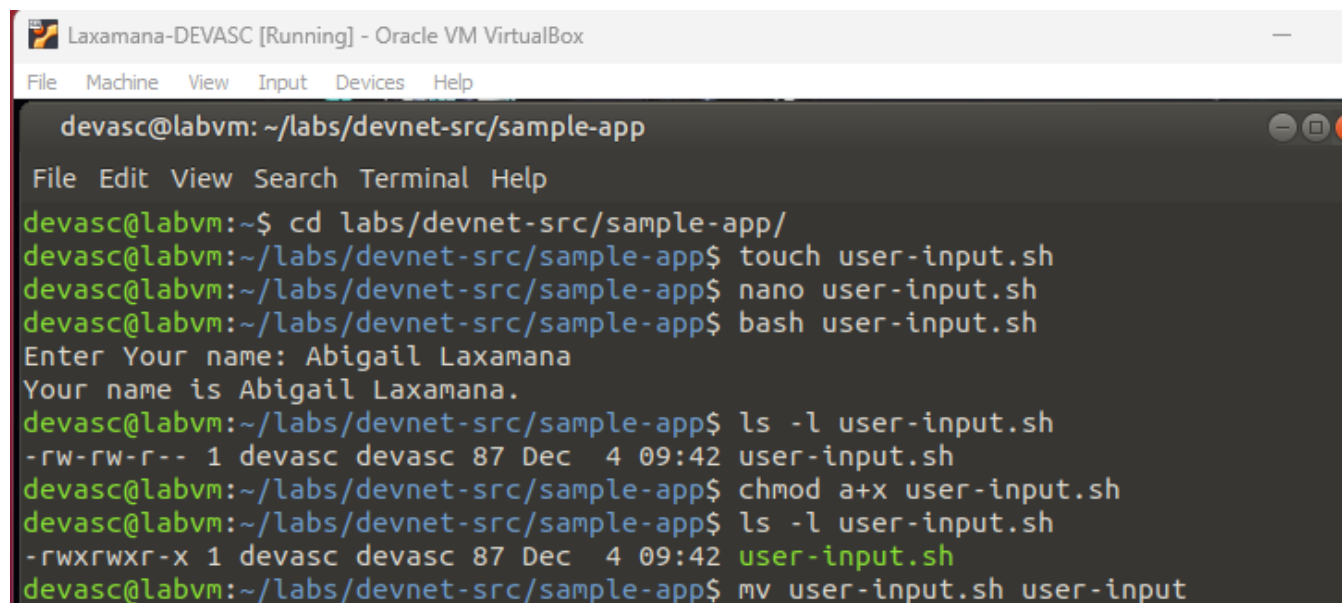
```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd labs/devnet-src/sample-app/
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: Abigail Laxamana
Your name is Abigail Laxamana.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 87 Dec  4 09:42 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 87 Dec  4 09:42 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.

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



```
Laxamana-DEVASC [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/
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: Abigail Laxamana
Your name is Abigail Laxamana.
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 87 Dec  4 09:42 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 87 Dec  4 09:42 user-input.sh
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
```

Step 9: Execute the script from the command line.

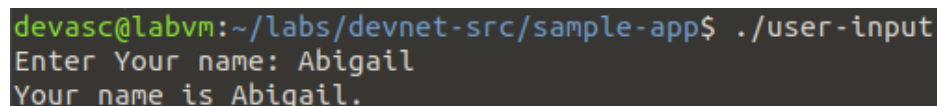
Now the script can be run from the command line without the source command or an extension. To run a bash script without the source command, you must preface the script with `./`.

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

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.

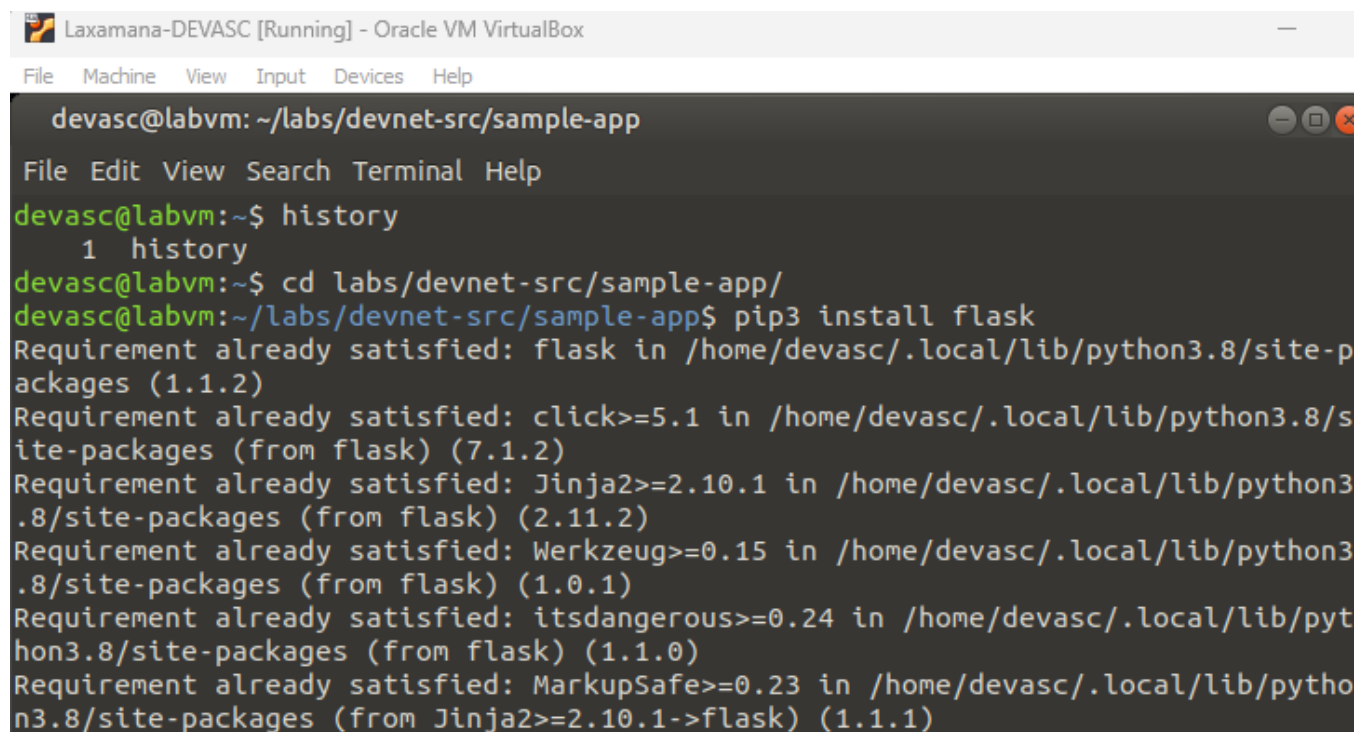
Web application developers using Python typically leverage a framework. A framework is a code library to make it easier for developers to create reliable, scalable and maintainable web applications. Flask is a web application framework written in Python. Other frameworks include Tornado and Pyramid.

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
```



```
Laxamana-DEVASC [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:~$ history
1 history
devasc@labvm:~$ cd labs/devnet-src/sample-app/
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: Jinja2>=2.10.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
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: MarkupSafe>=0.23 in /home/devasc/.local/lib/python3.8/site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
```

Step 2: Open the sample_app.py file.

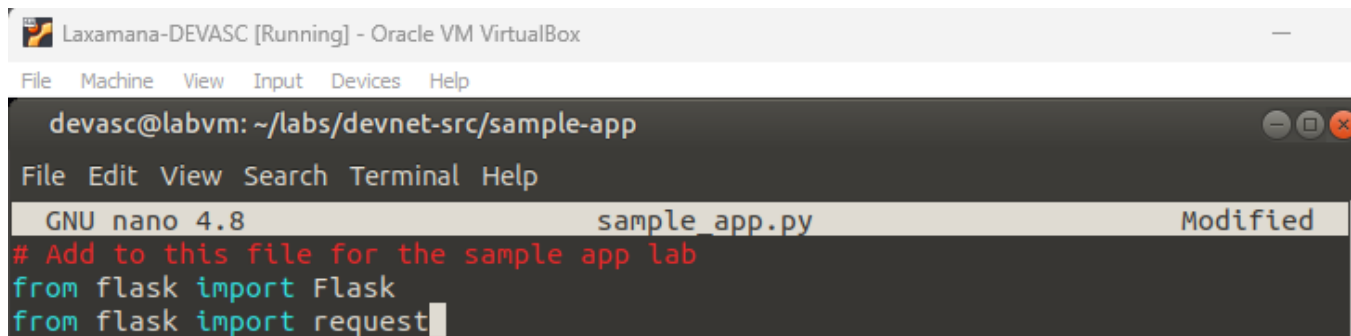
Open the sample_app.py file located in the /sample-app directory. You can do this inside VS Code or you can use a command-line text editor like nano or vim.

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
```

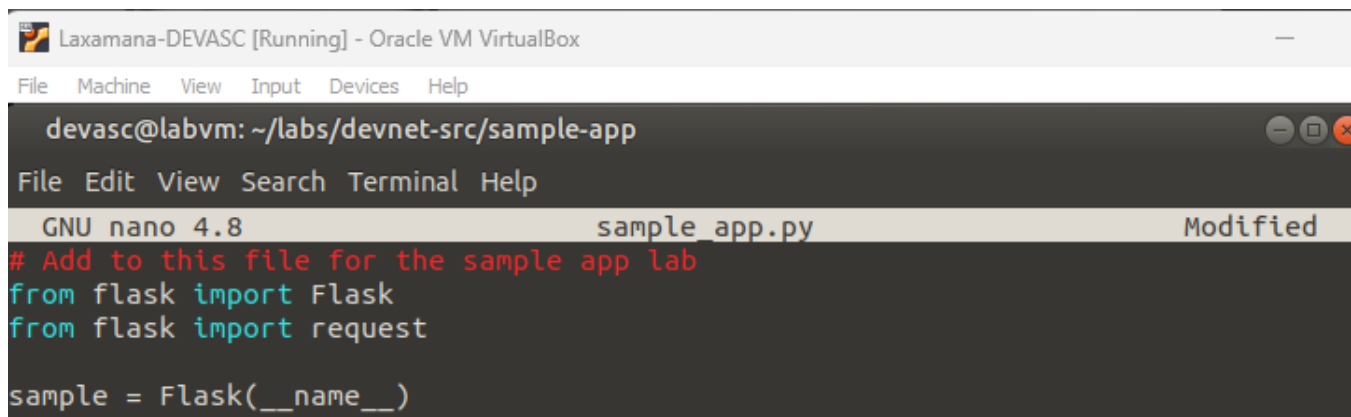


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 sample_app.py Modified
# Add to this file for the sample app lab
from flask import Flask
from flask import request
```

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__)
```

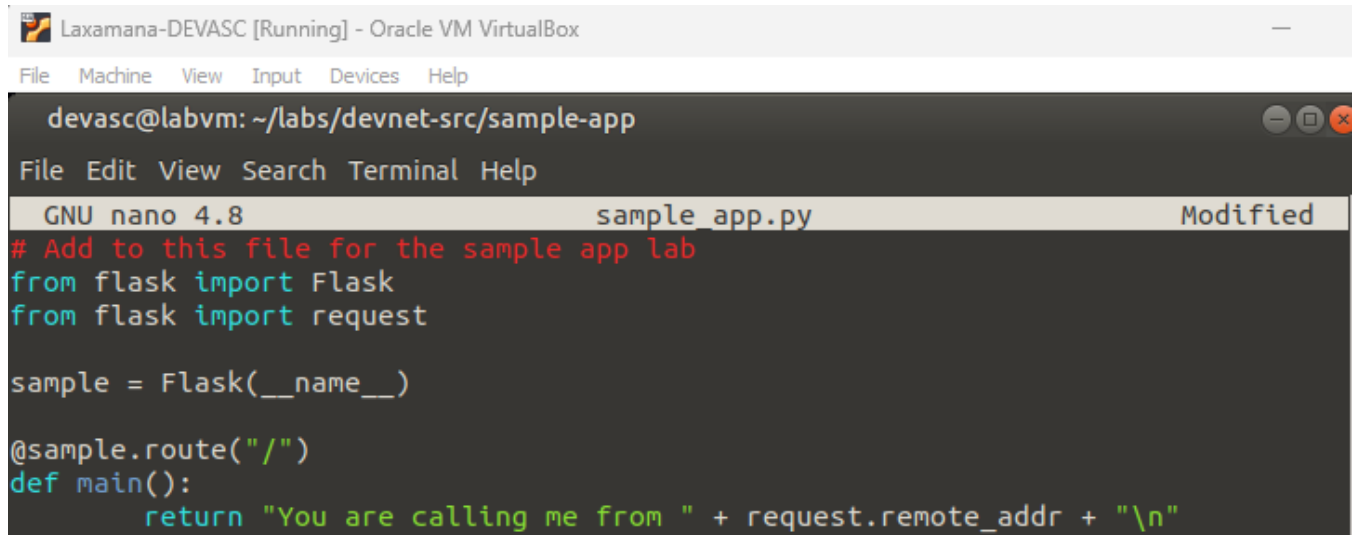


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
GNU nano 4.8 sample_app.py Modified
# Add to this file for the sample app lab
from flask import Flask
from flask import request
sample = Flask(__name__)
```

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"
```

```
Laxamana-DEVASC [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 Modified
# Add to this file for the sample app lab
from flask import Flask
from flask import request

sample = Flask(__name__)

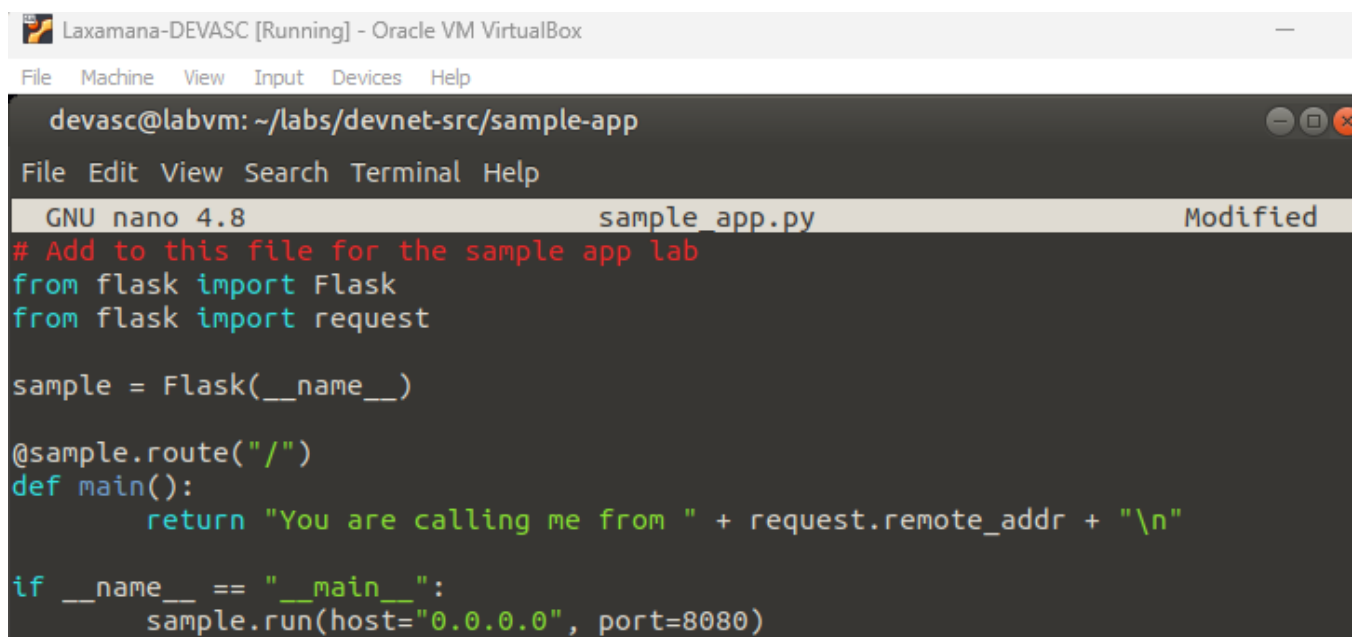
@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"
```

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)
```



```
Laxamana-DEVASC [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 Modified
# Add to this file for the sample app lab
from flask import Flask
from flask import request

sample = Flask(__name__)

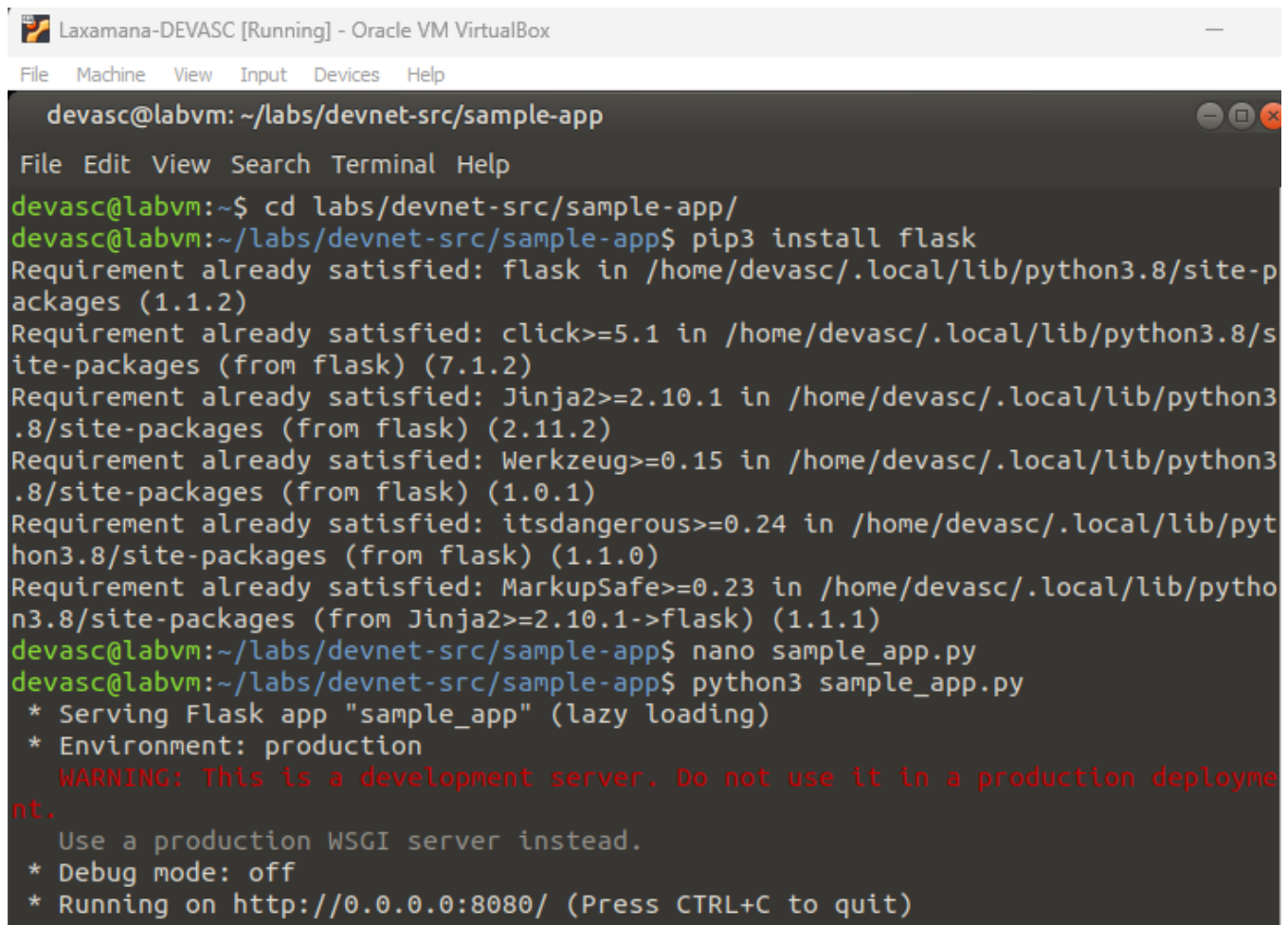
@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"

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


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 "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". The terminal output shows the following commands and results:

```
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~$ cd labs/devnet-src/sample-app/
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: Jinja2>=2.10.1 in /home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
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: 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
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)
```

Step 8: Verify the server is running.

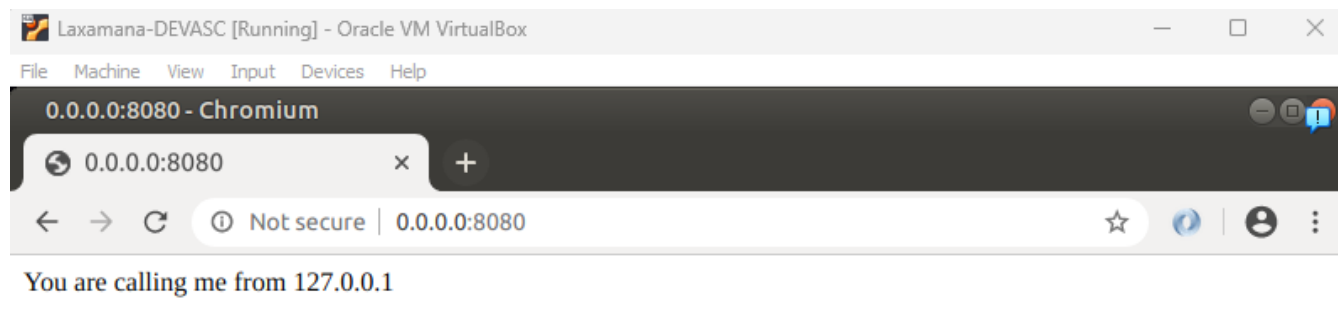
You can verify the server is running in one of two ways.

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

Lab - Build a Sample Web App in a Docker Container

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.

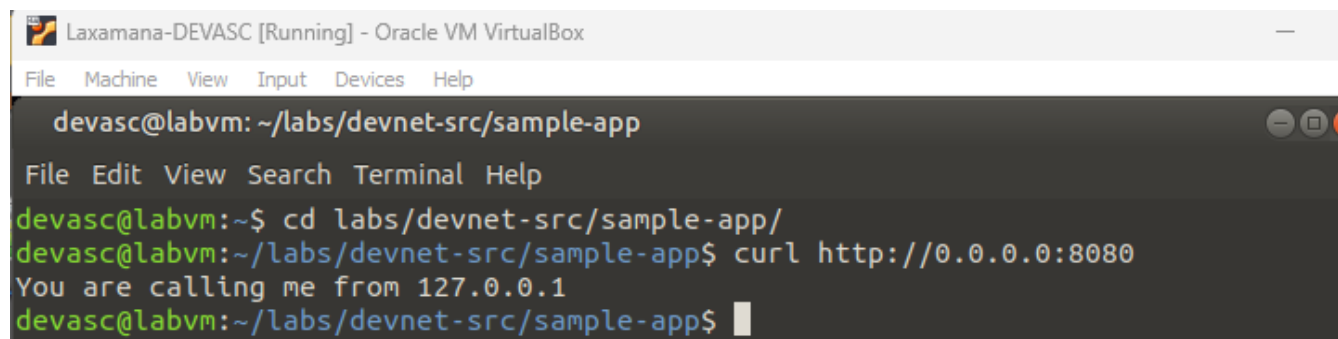


- b. Open another terminal window and use the command-line URL tool (cURL) to verify the server's response.

```
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$
```



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.

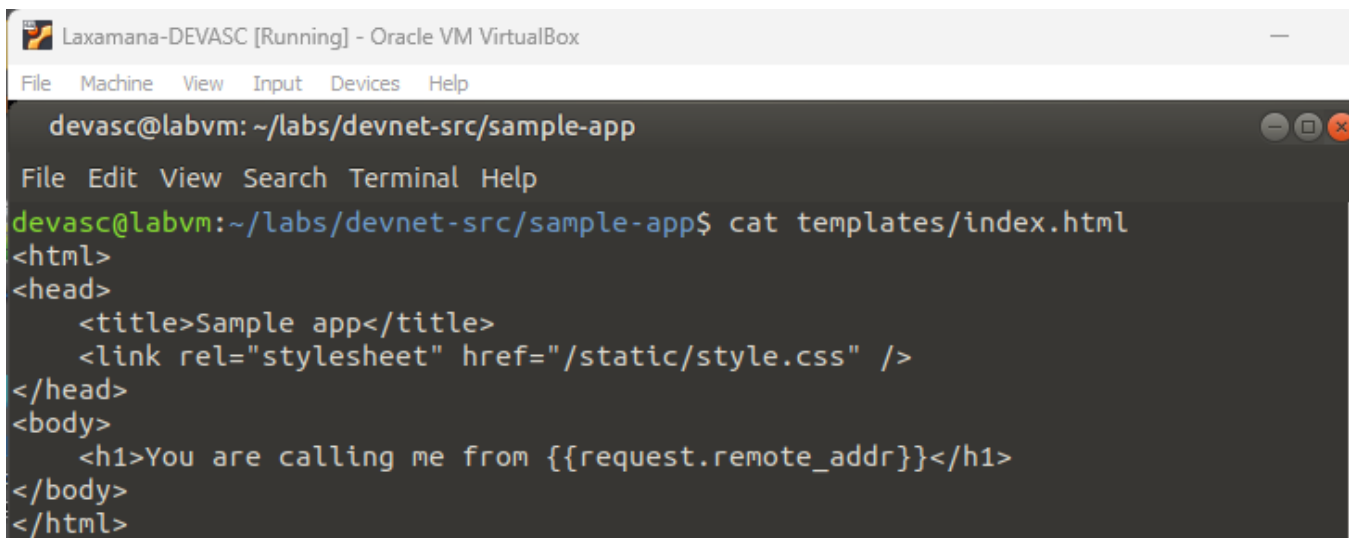
The directories templates and static are already in the sample-app directory. Open the index.html and style.css to view their contents. If you are familiar with HTML and CSS, feel free to customize these directories and files as much as you like. However, be sure you keep the embedded `{{request.remote_addr}}` 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$
```

A screenshot of a terminal window titled "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". The terminal shows the command `cat templates/index.html` and its output, which is the HTML code from the previous block. The terminal window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The prompt is `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$ 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>
```

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

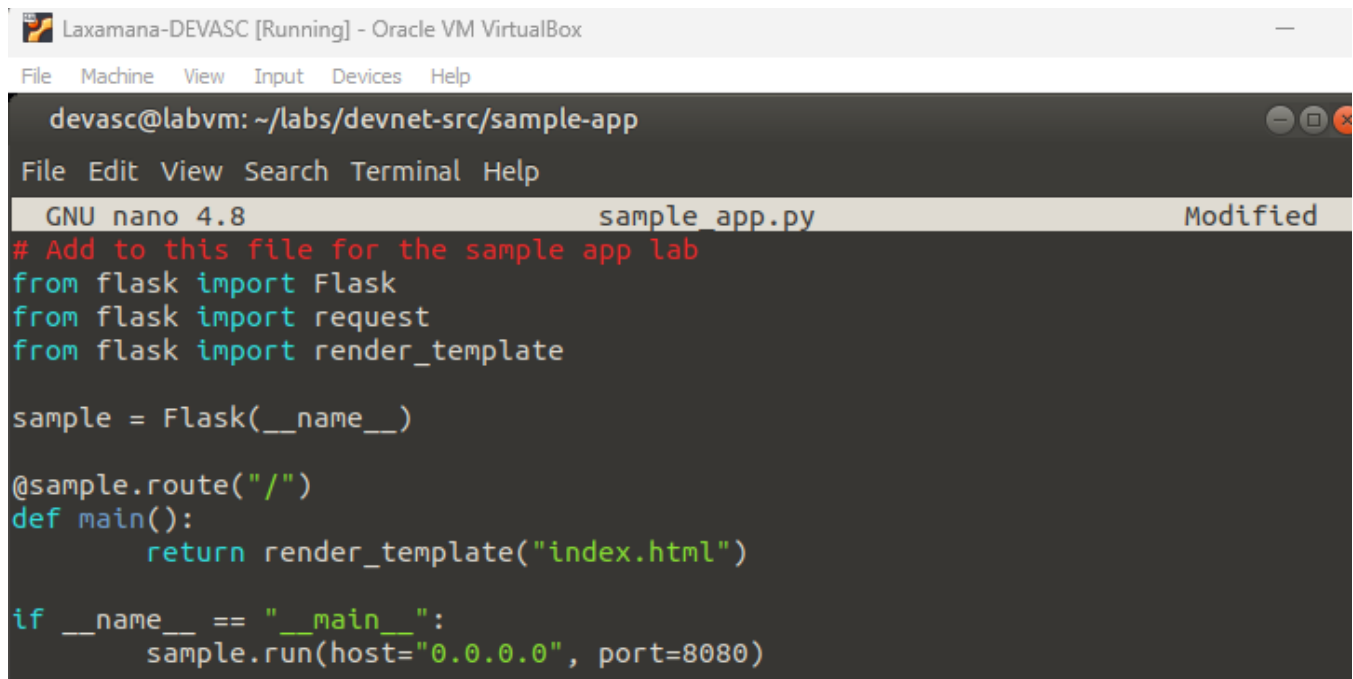
Now that you have explored the basic website files, you need to update the `sample_app.py` file so that it renders the `index.html` file instead of just returning data. Generating HTML content using Python code can be cumbersome, especially when using conditional statements or repeating structures. The 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)
```



The screenshot shows a terminal window titled "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". Inside the terminal, the nano editor is open, editing the file "sample_app.py". The code in the editor is as follows:

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

sample = Flask(__name__)

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

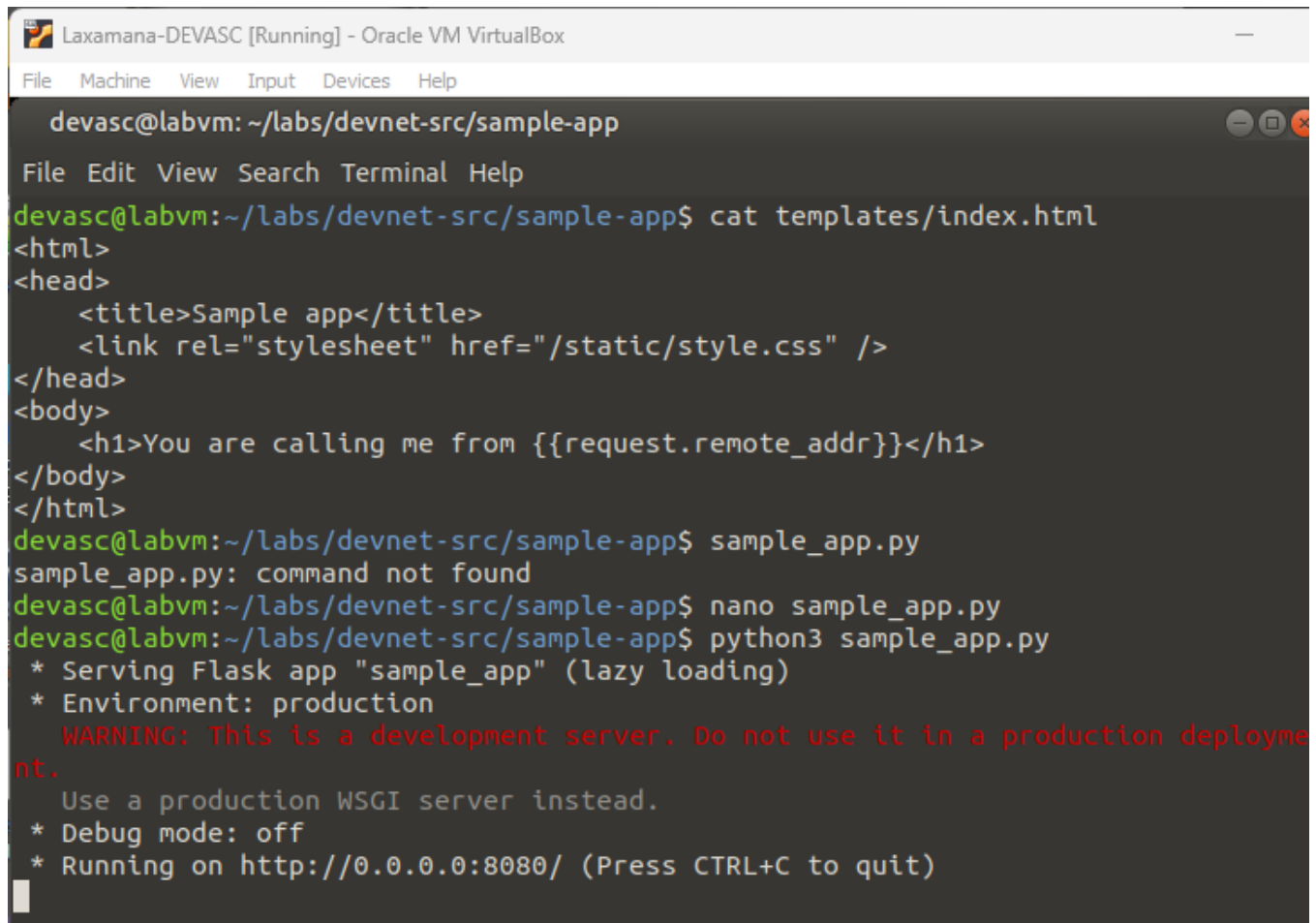
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

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)
```

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.



```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

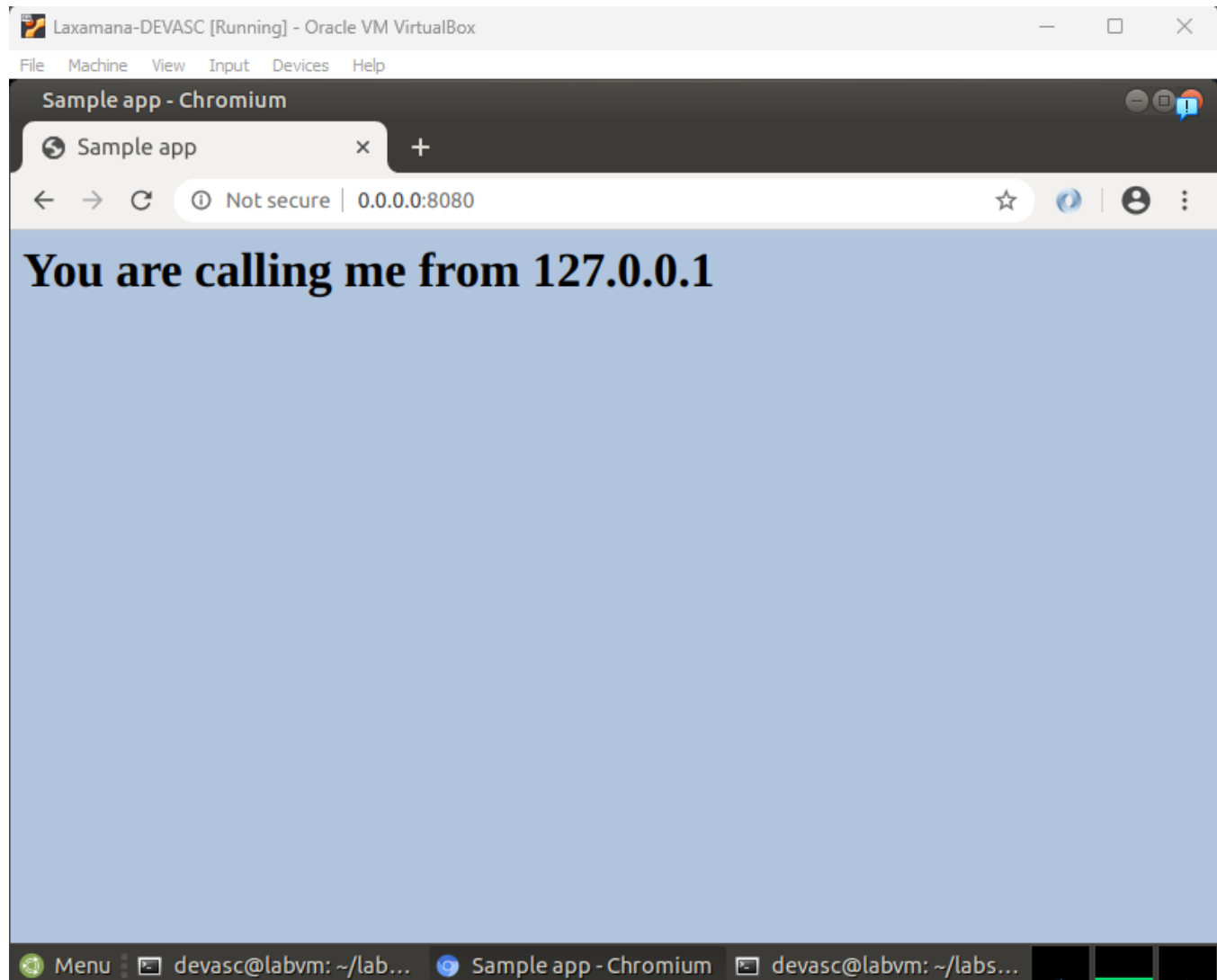
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ cat templates/index.html
<html>
<head>
  <title>Sample app</title>
  <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
  <h1>You are calling me from {{request.remote_addr}}</h1>
</body>
</html>
devasc@labvm:~/labs/devnet-src/sample-app$ sample_app.py
sample_app.py: command not found
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)
```

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

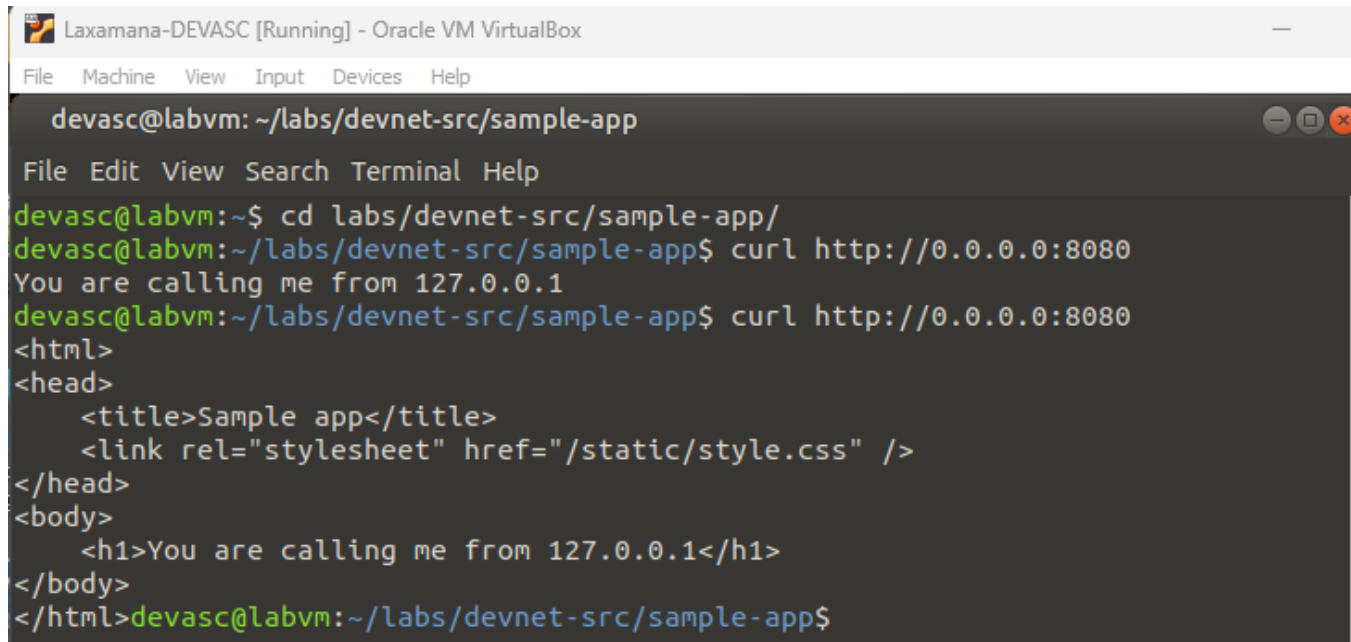


- b. Open another terminal window and use the curl command to verify the server's response. This is where you will see the result of the HTML code rendered automatically using the `render_template` function. In this case, you will get all the HTML content. However, the dynamic Python code will be replaced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



```
Laxamana-DEVASC [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/  
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$ 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.
- Start the container and verify it is running.

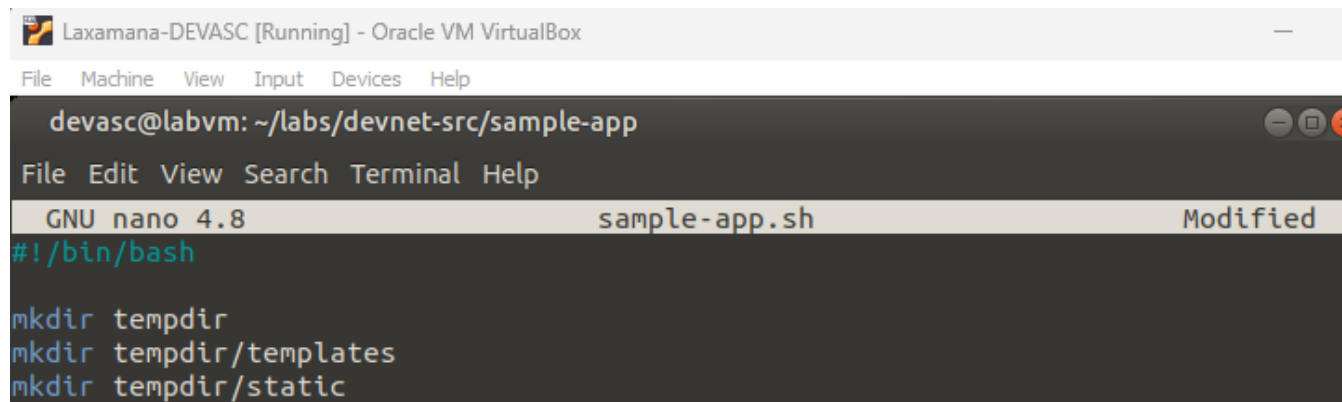
Step 1: Create temporary directories to store the website files.

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 tmpdir as the parent folder.

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



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

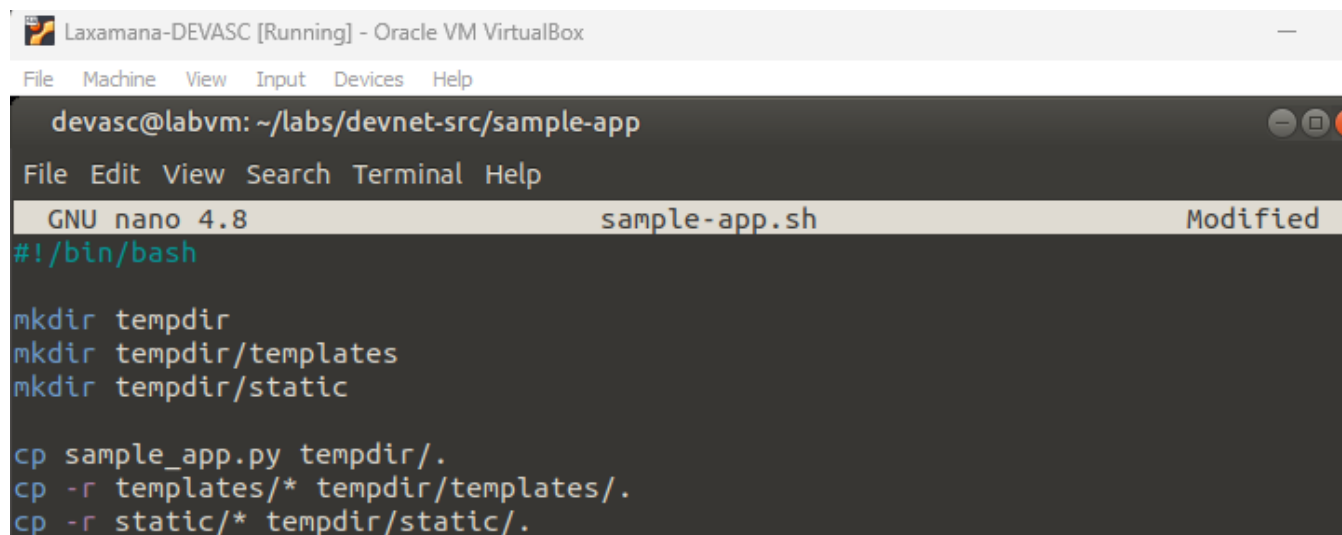


The screenshot shows a terminal window titled "devasc@labvm: ~/labs/devnet-src/sample-app". The terminal output shows the execution of three commands: `mkdir tempdir`, `mkdir tempdir/templates`, and `mkdir tempdir/static`. The terminal is running GNU nano 4.8 editing the file `sample-app.sh`.

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/.
```



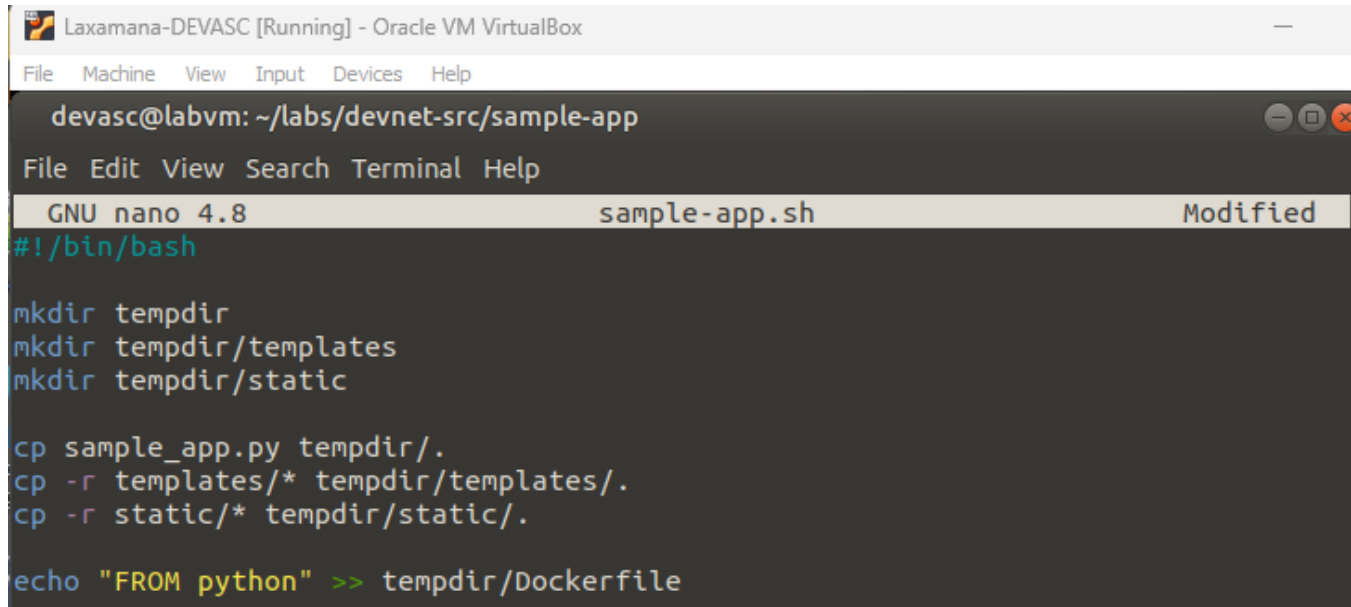
The screenshot shows a terminal window titled "devasc@labvm: ~/labs/devnet-src/sample-app". The terminal output shows the execution of three commands: `mkdir tempdir`, `mkdir tempdir/templates`, and `mkdir tempdir/static`. The terminal is running GNU nano 4.8 editing the file `sample-app.sh`. The output also shows the execution of three commands: `cp sample_app.py tempdir/.`, `cp -r templates/* tempdir/templates/.`, and `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
```



```
Laxamana-DEVASC [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.sh Modified
#!/bin/bash

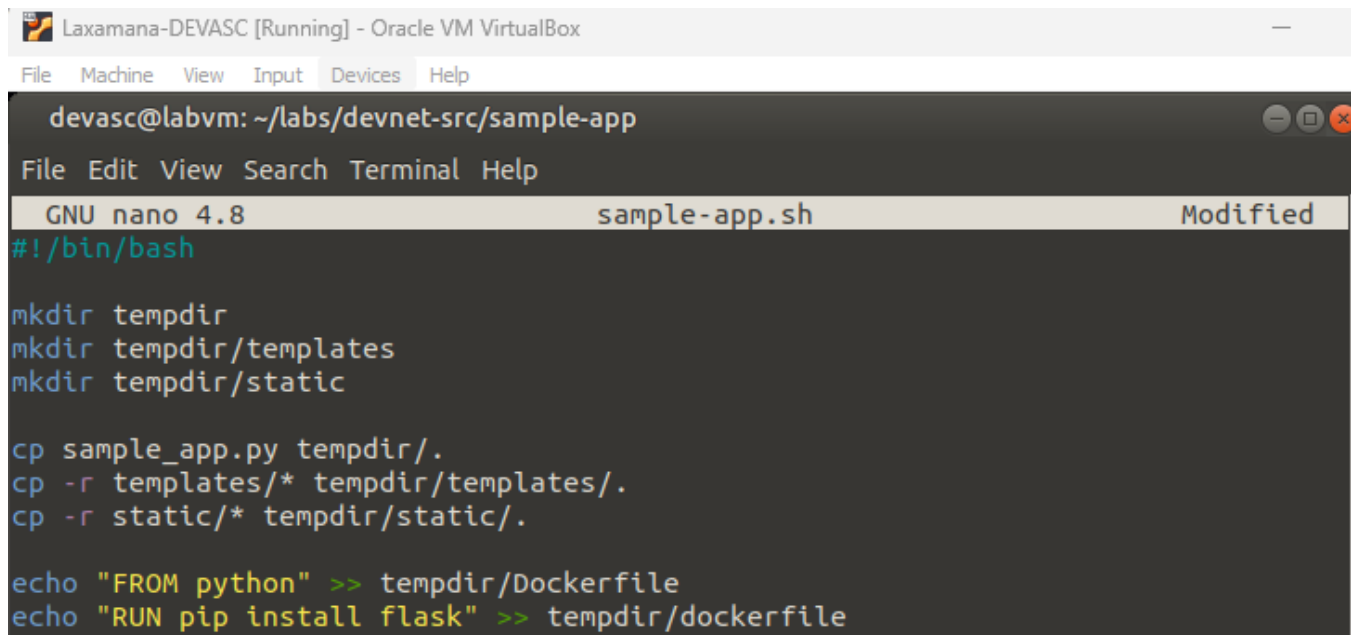
mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

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

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

- 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
```



```
Laxamana-DEVASC [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.sh Modified
#!/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/.

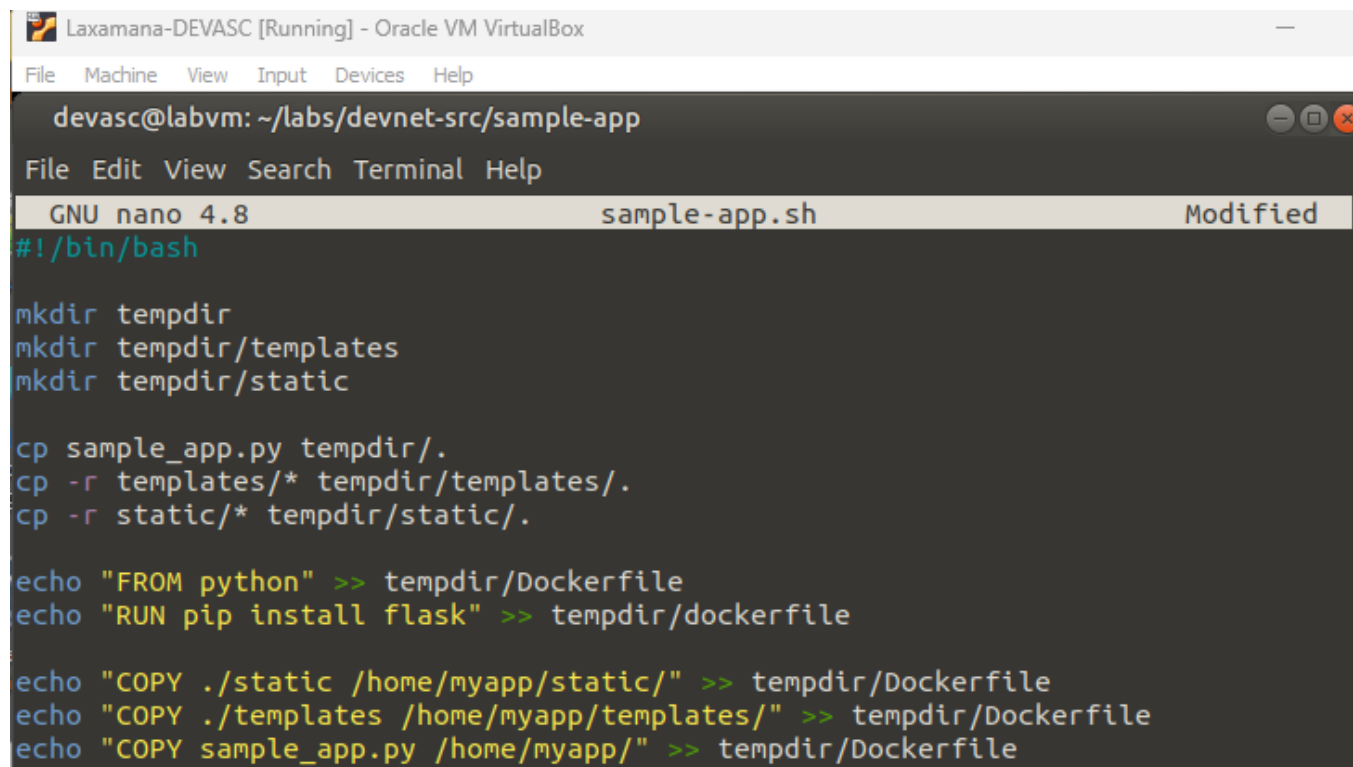
echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/dockerfile
```

- 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
```



The screenshot shows a terminal window titled "devasc@labvm: ~/labs/devnet-src/sample-app". The terminal is running the GNU nano 4.8 editor, editing a file named "sample-app.sh". The file contains the following commands:

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

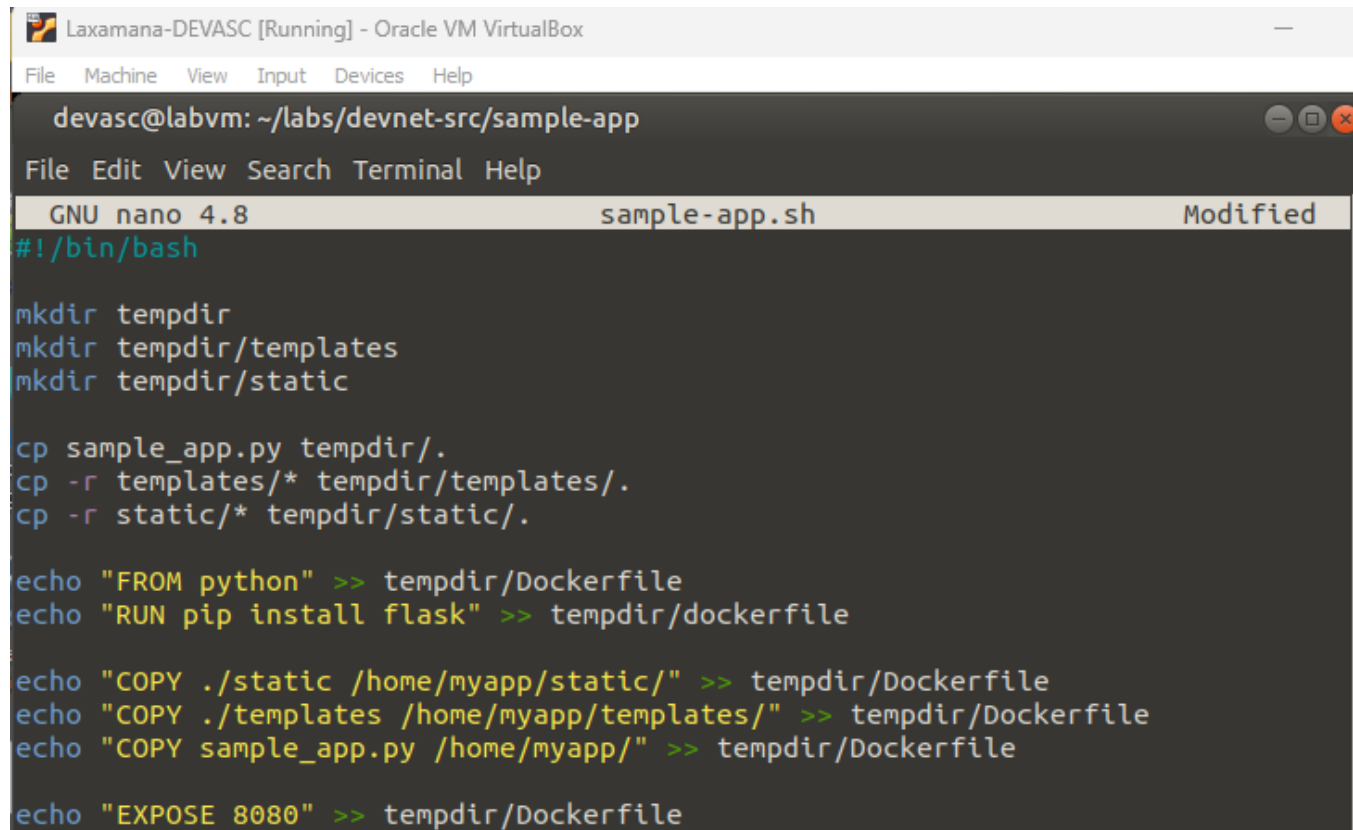
cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

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
```

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

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



The screenshot shows a terminal window titled "Laxamana-DEVASC [Running] - Oracle VM VirtualBox". The terminal prompt is "devasc@labvm: ~/labs/devnet-src/sample-app". The terminal is running the GNU nano 4.8 editor on a file named "sample-app.sh". The file content is as follows:

```
#!/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/.

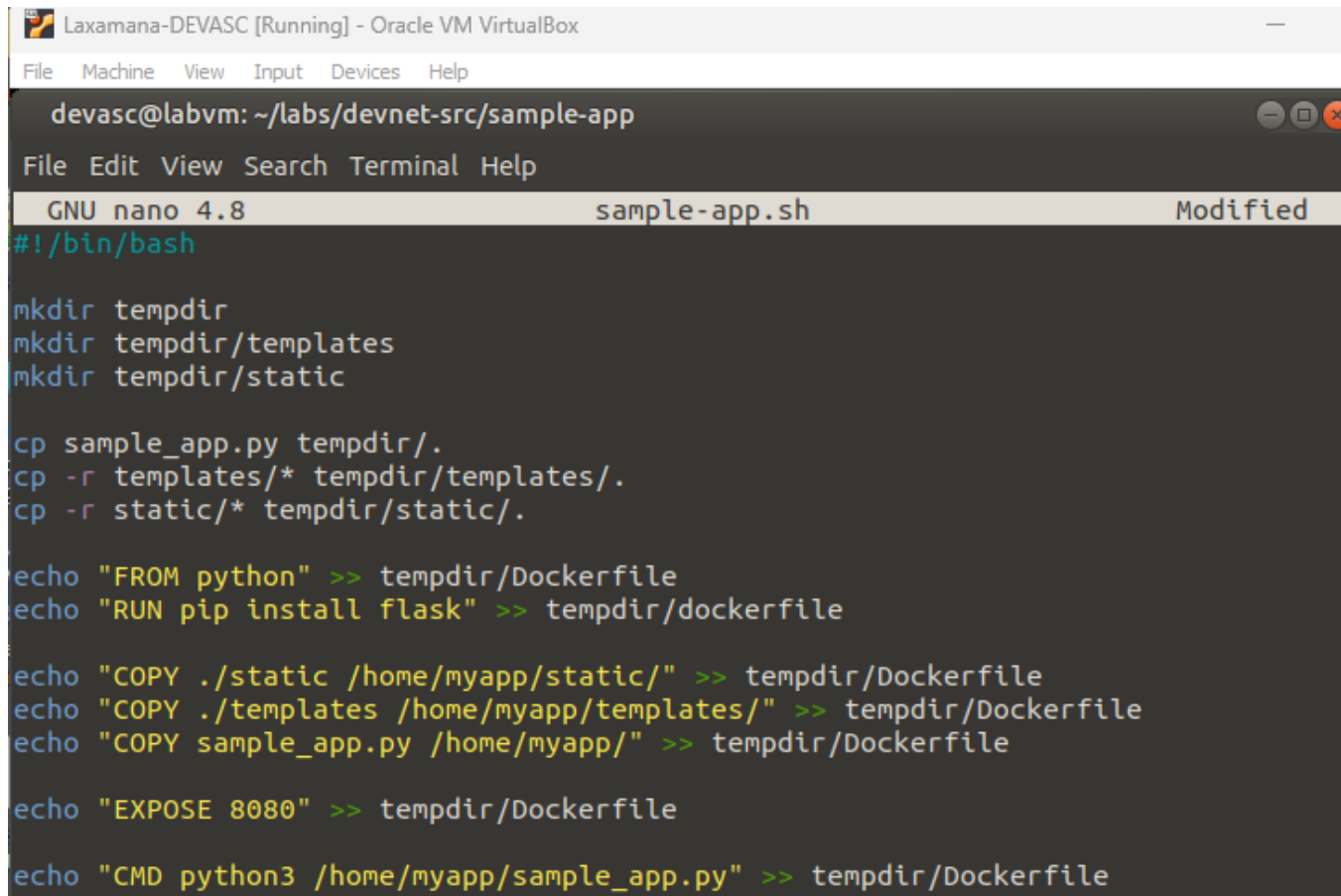
echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile

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
```

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

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



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

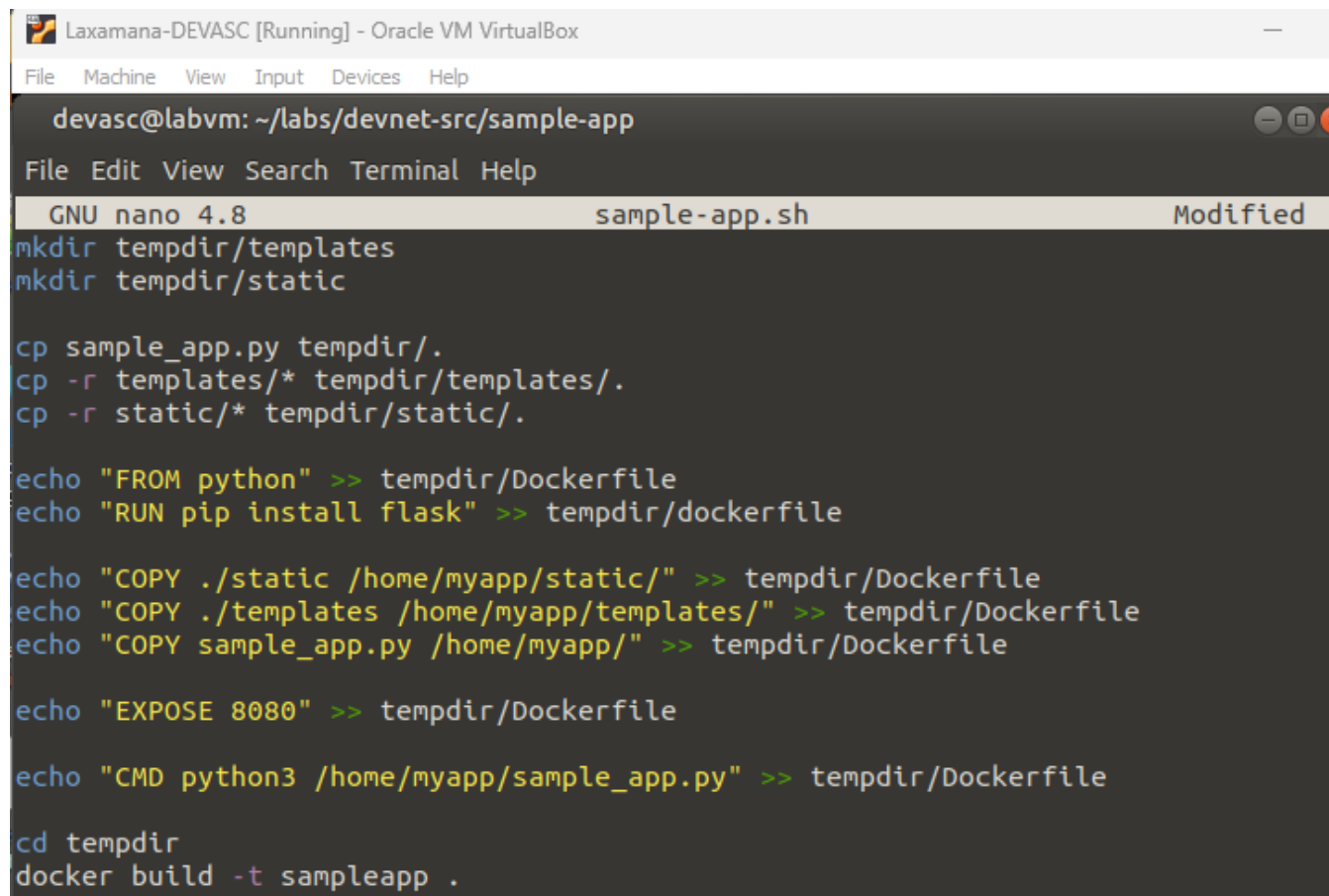
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile
```

Step 4: Build the Docker container. `echo "EXPOSE 8080" >> tempdir/Dockerfile`

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 .
```



```
Laxamana-DEVASC [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.sh Modified
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 .
```

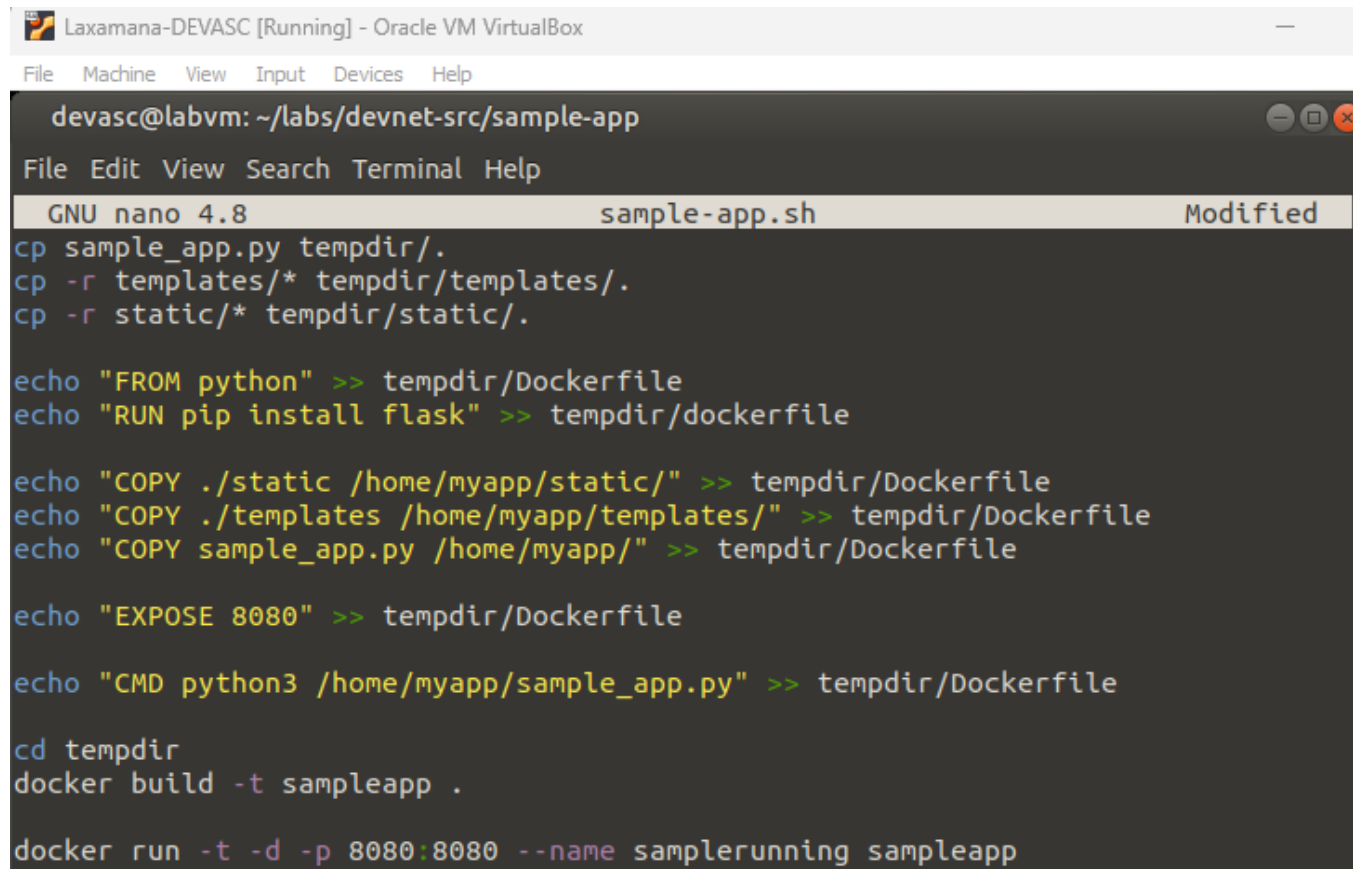
Step 5: Start the container and verify it is running.

- 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).



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

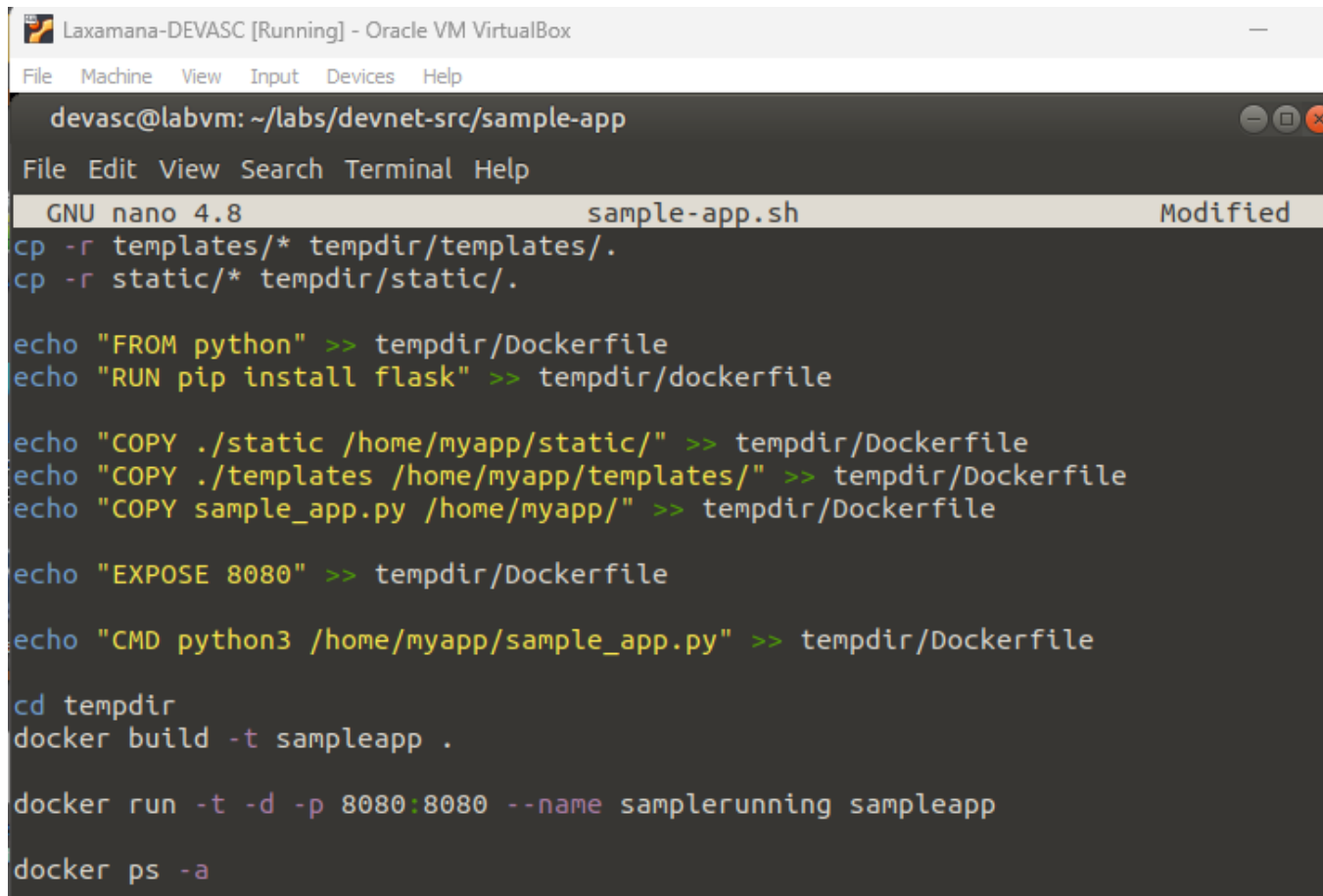
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir
docker build -t sampleapp .

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

- 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`



```
Laxamana-DEVASC [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.sh Modified
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
```

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
```

Lab - Build a Sample Web App in a Docker Container

---> 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

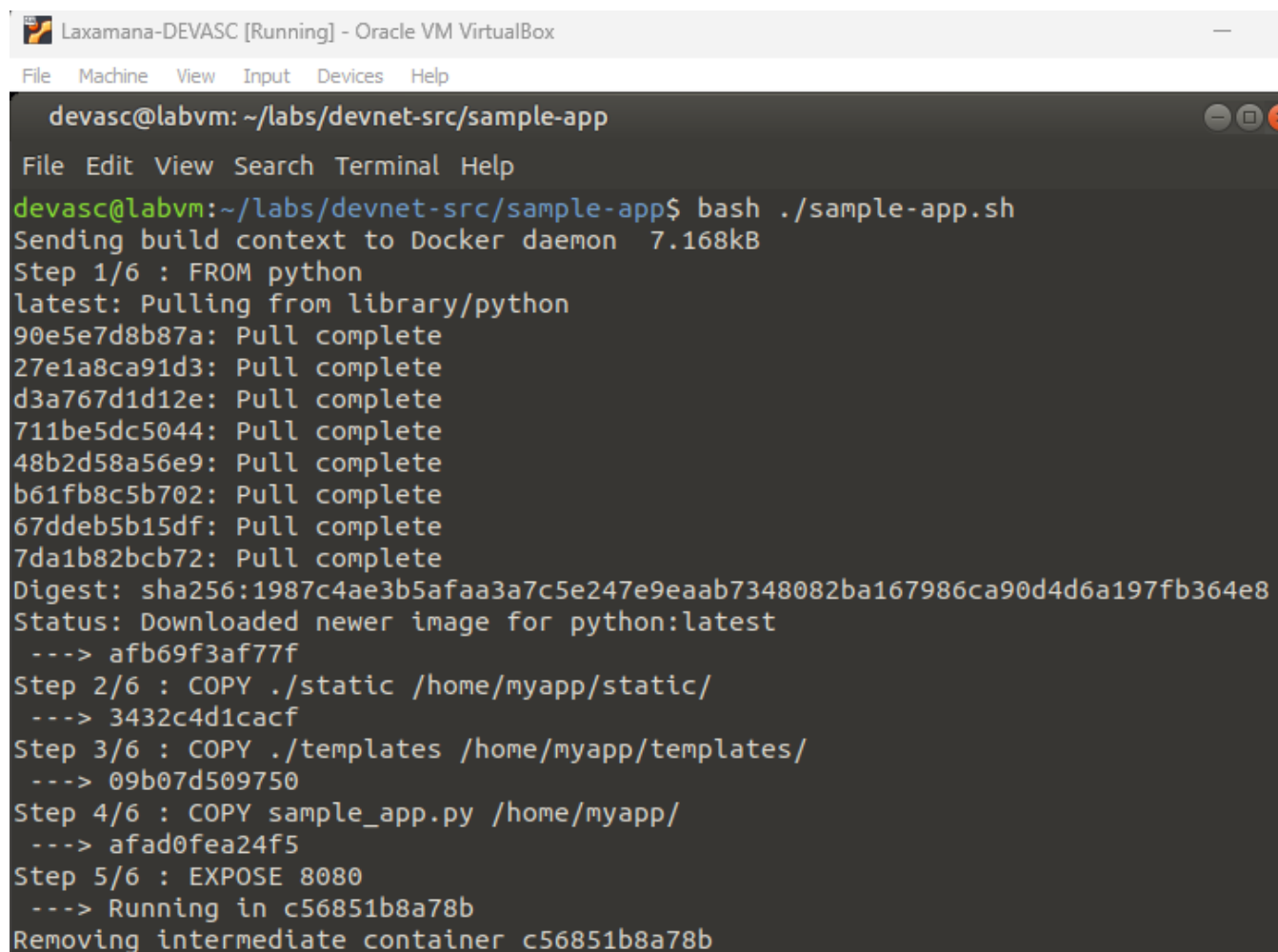
Successfully tagged sampleapp:latest

8953a95374ff8ebc203059897774465312acc8f0ed6abd98c4c2b04448a56ba5

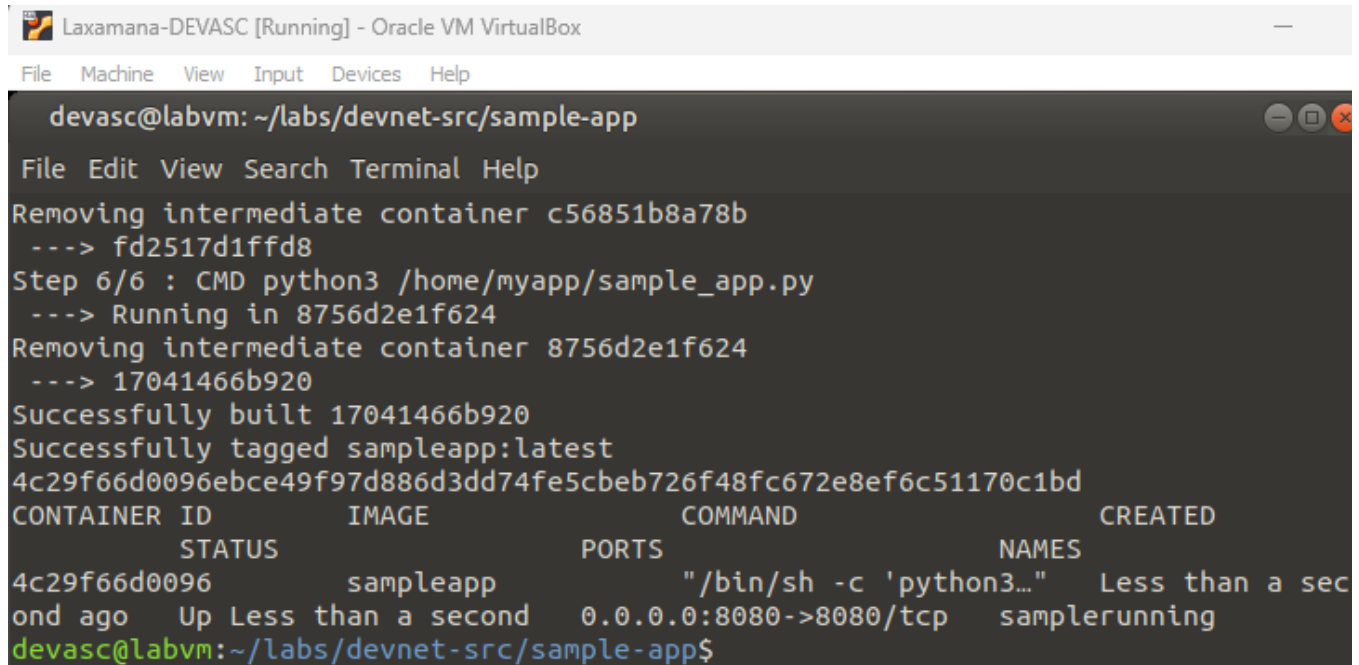
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
8953a95374ff	sampleapp	"/bin/sh -c 'python ..."	1 second ago	Up Less than a second	0.0.0.0:8080->8080/tcp

sampleapp:latest

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



```
Laxamana-DEVASC [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 ./sample-app.sh
Sending build context to Docker daemon 7.168kB
Step 1/6 : FROM python
latest: Pulling from library/python
90e5e7d8b87a: Pull complete
27e1a8ca91d3: Pull complete
d3a767d1d12e: Pull complete
711be5dc5044: Pull complete
48b2d58a56e9: Pull complete
b61fb8c5b702: Pull complete
67ddeb5b15df: Pull complete
7da1b82bcb72: Pull complete
Digest: sha256:1987c4ae3b5afaa3a7c5e247e9eaab7348082ba167986ca90d4d6a197fb364e8
Status: Downloaded newer image for python:latest
---> afb69f3af77f
Step 2/6 : COPY ./static /home/myapp/static/
---> 3432c4d1cacf
Step 3/6 : COPY ./templates /home/myapp/templates/
---> 09b07d509750
Step 4/6 : COPY sample_app.py /home/myapp/
---> afad0fea24f5
Step 5/6 : EXPOSE 8080
---> Running in c56851b8a78b
Removing intermediate container c56851b8a78b
```



```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
Removing intermediate container c56851b8a78b
---> fd2517d1ffd8
Step 6/6 : CMD python3 /home/myapp/sample_app.py
---> Running in 8756d2e1f624
Removing intermediate container 8756d2e1f624
---> 17041466b920
Successfully built 17041466b920
Successfully tagged sampleapp:latest
4c29f66d0096ebce49f97d886d3dd74fe5cbeb726f48fc672e8ef6c51170c1bd
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS             PORTS              NAMES
4c29f66d0096        sampleapp           "/bin/sh -c 'python3..." Less than a sec
ond ago            Up Less than a second 0.0.0.0:8080->8080/tcp  samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$
```

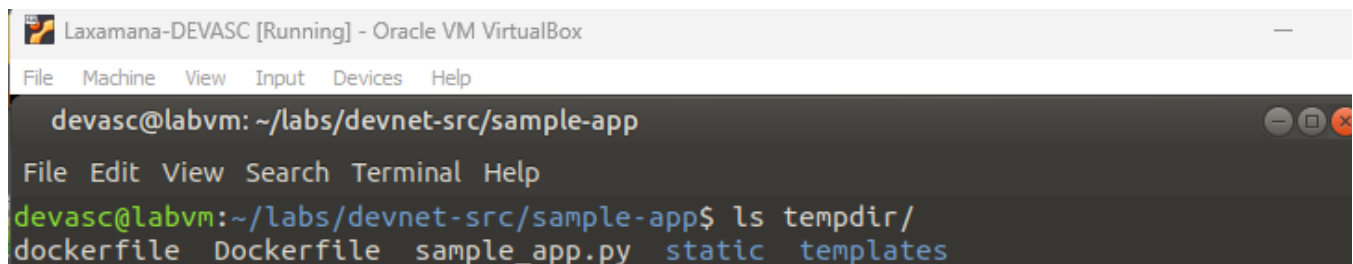
Step 2: Investigate the running Docker container and the web app.

- The creation of the tmpdir 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 tmpdir/
```

```
Dockerfile sample_app.py static templates
```

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



```
Laxamana-DEVASC [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 tmpdir/
Dockerfile Dockerfile sample_app.py static templates
```

- 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 tmpdir/Dockerfile
```

```
FROM python
```

```
RUN pip install flask
```

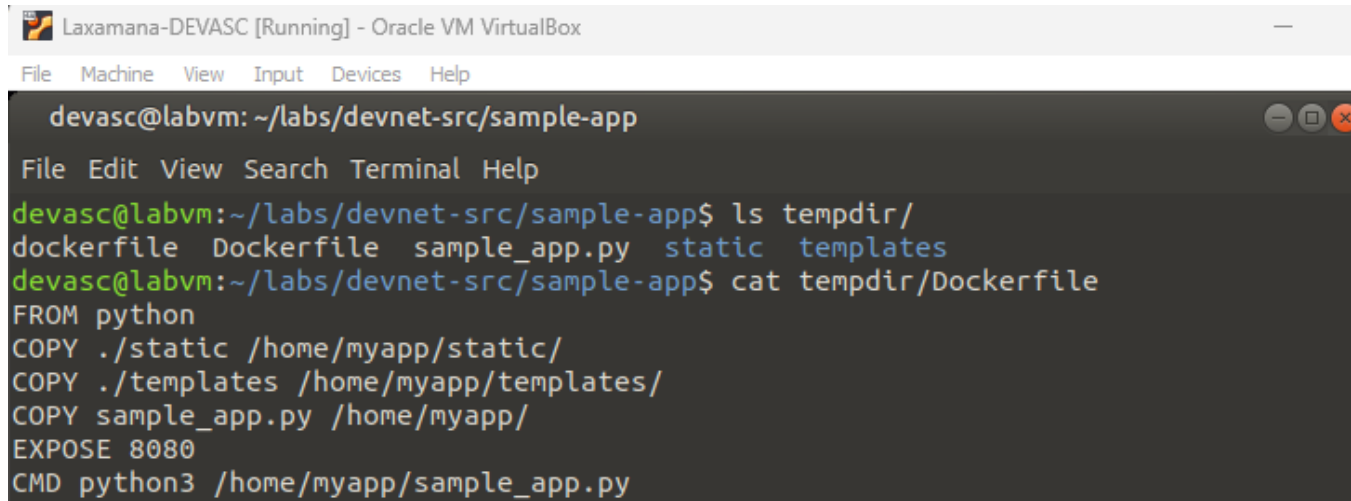
```
COPY ./static /home/myapp/static/
```

```
COPY ./templates /home/myapp/templates/
```

```
COPY sample_app.py /home/myapp/
```

EXPOSE 8080

CMD python3 /home/myapp/sample_app.py

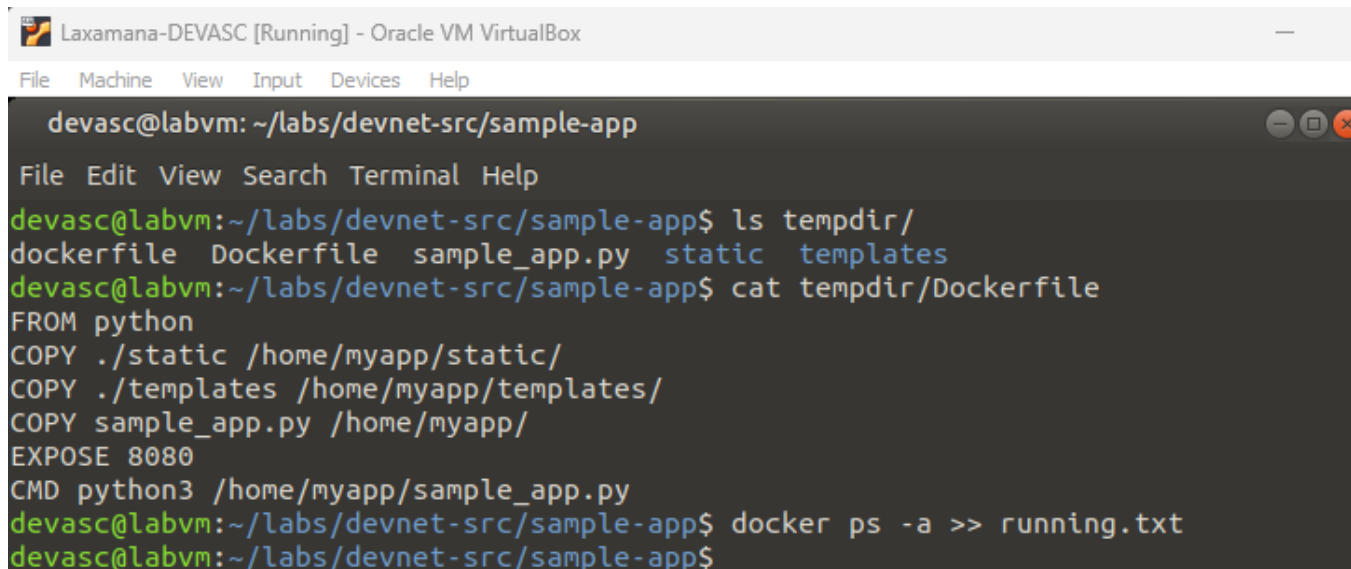


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/sample-app
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/sample-app$ ls tempdir/
dockerfile Dockerfile sample_app.py static templates
devasc@labvm:~/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
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
```

- 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$
```



```
Laxamana-DEVASC [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 tempdir/
dockerfile Dockerfile sample_app.py static templates
devasc@labvm:~/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
COPY ./static /home/myapp/static/
COPY ./templates /home/myapp/templates/
COPY sample_app.py /home/myapp/
EXPOSE 8080
CMD python3 /home/myapp/sample_app.py
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/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$
devasc@labvm:~/labs/devnet-src/sample-app$
```

```
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$
```

- 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>
```



```
</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 08:00:27:e9:3d:e6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 85143sec preferred_lft 85143sec
    inet6 fe80::a00:27ff:fee9:3de6/64 scope link
        valid_lft forever preferred_lft forever
3: dummy0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether a6:42:60:1d:10:29 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
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 02:42:bc:84:8b:a4 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:bcff:fe84:8ba4/64 scope link
        valid_lft forever preferred_lft forever
8: veth53ae3fa@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 02:ff:f0:34:7e:ba brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ff:f0ff:fe34:7eba/64 scope link
        valid_lft forever preferred_lft forever
```

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:~/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@c36fddd0bc59:/#
```


Lab - Build a Sample Web App in a 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:~/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@c36fddd0bc59:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
```

- 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:/#
```

```
root@c36fddd0bc59:/# ls home/myapp/
sample_app.py static templates
```

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

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

```
root@c36fddd0bc59:/# 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	STATUS	PORTS
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$
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS         NAMES
c36fddd0bc59   sampleapp     "/bin/sh -c 'python3..." 8 minutes ago  Exited (137) 6 second
s ago         samplerunning
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$
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker start samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
NAMES
c36fddd0bc59   sampleapp     "/bin/sh -c 'python3..." 8 minutes ago  Up 3 seconds  0.0.0.
0:8080->8080/tcp, :::8080->8080/tcp  samplerunning
```

- 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
```

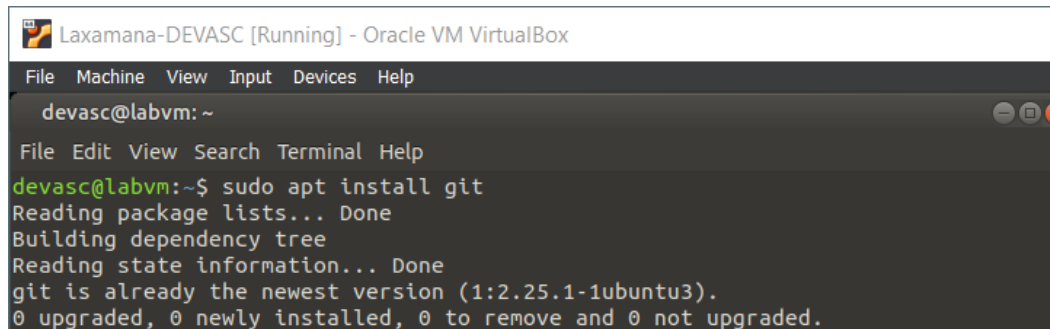
```
devasc@labvm:~/labs/devnet-src/sample-app$
```

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

Part 7: Git Repository and Conclusion

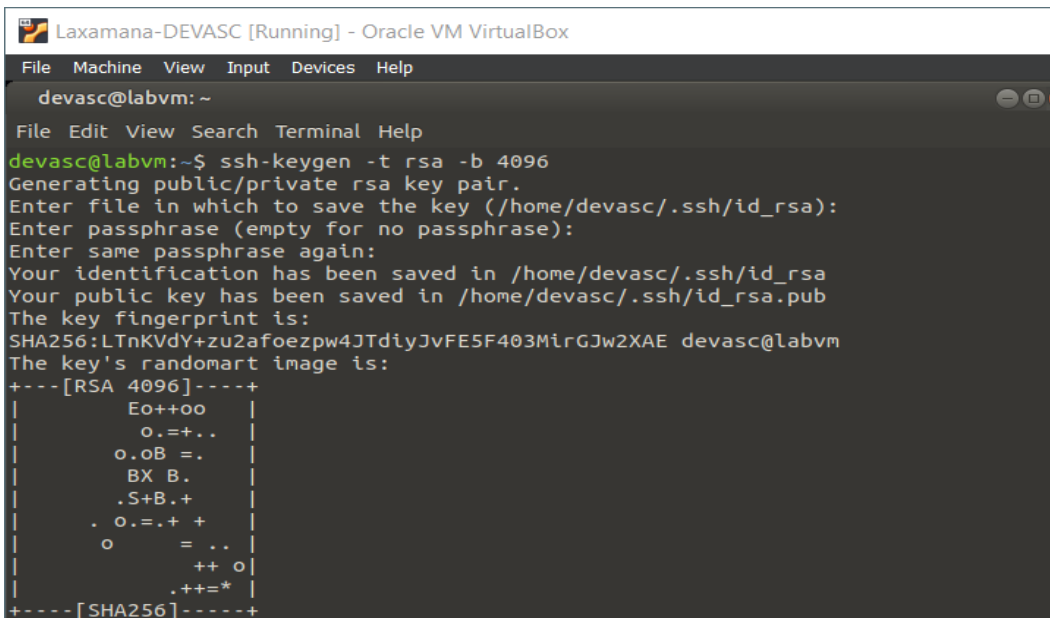
Step 1: Saving to Git repository

- Install git

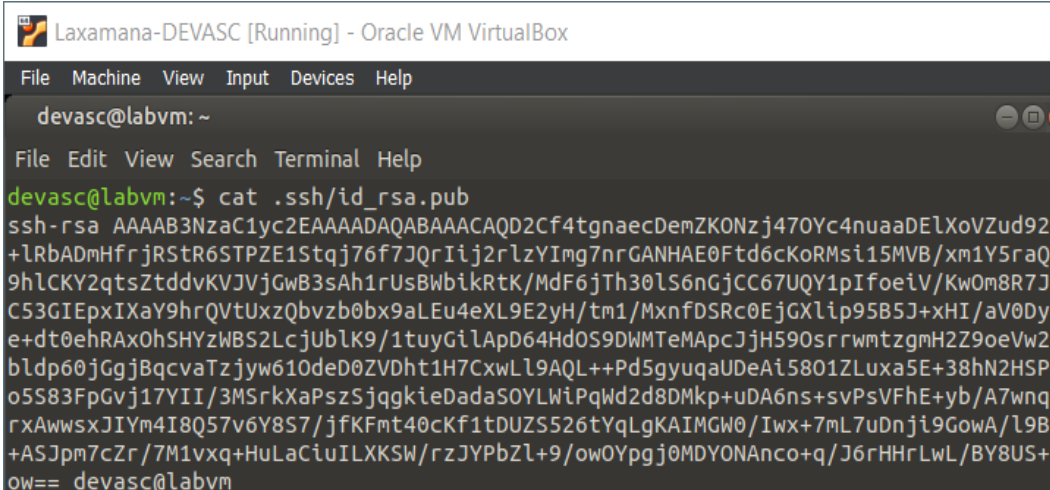


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help
devasc@labvm:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.25.1-1ubuntu3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

- Generate RSA Key

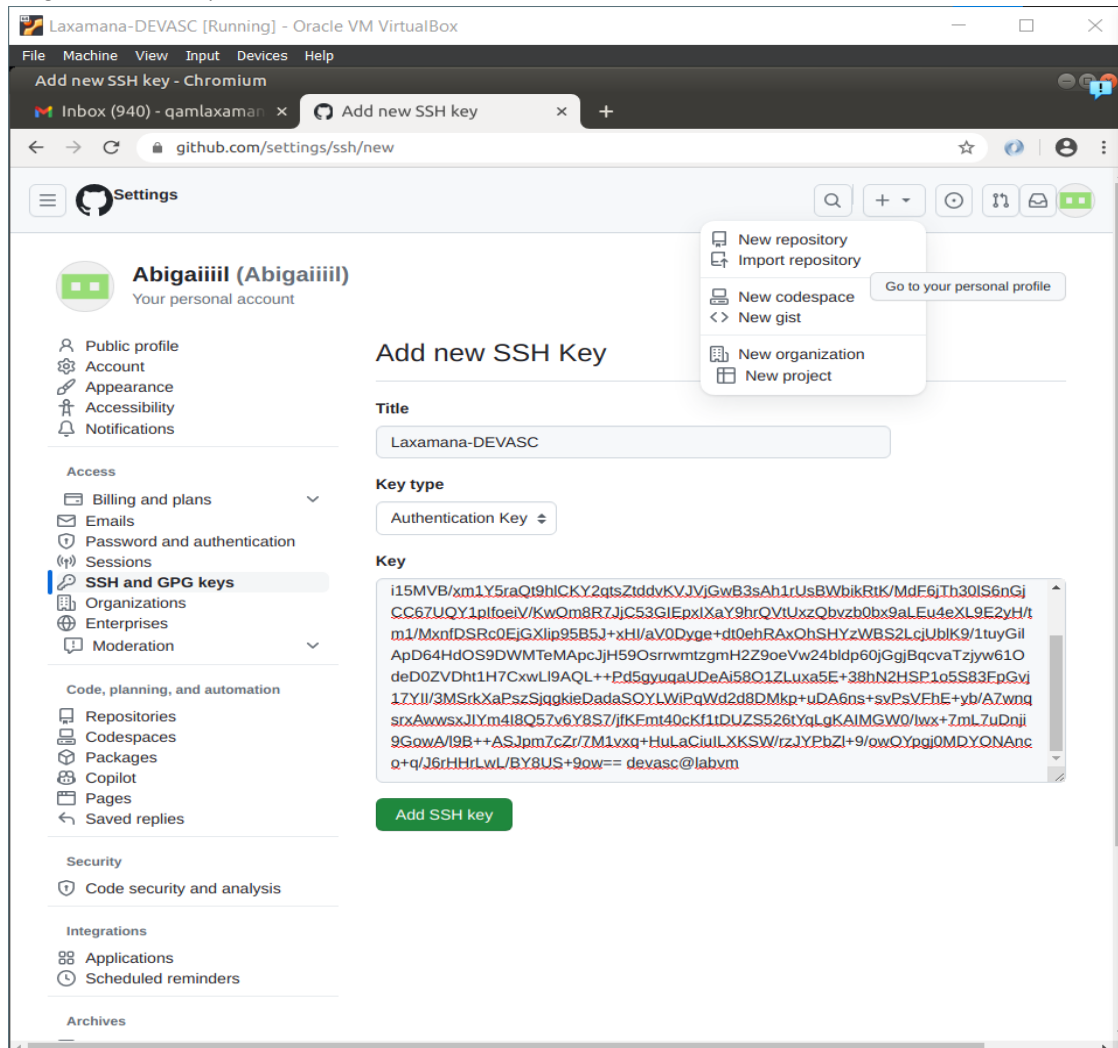


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help
devasc@labvm:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/devasc/.ssh/id_rsa):
Enter passphrase (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:LTnKVdY+zu2afoezpw4JTdiyJvFE5F403MirGJw2XAE devasc@labvm
The key's randomart image is:
+---[RSA 4096]---+
|      Eo++oo      |
|      o.=+..      |
|      o.oB =.      |
|      BX B.        |
|      .S+B.+        |
|      . o.=.+ +      |
|      o      = ..    |
|      ++ o         |
|      .++=*        |
+---[SHA256]-----+
```

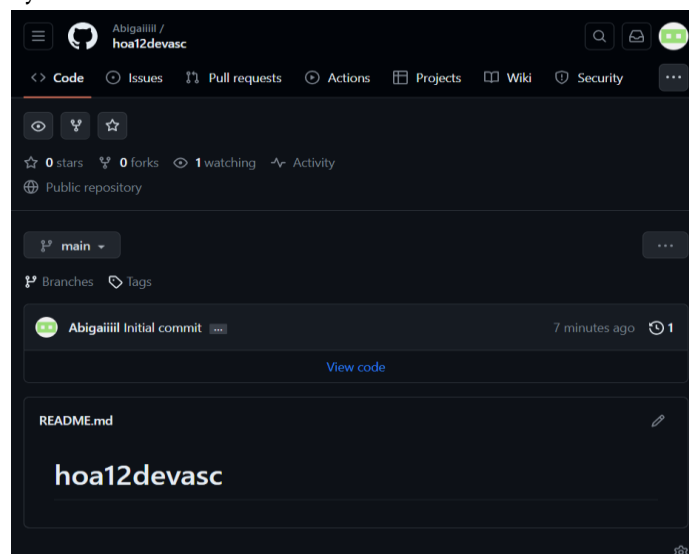


```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help
devasc@labvm:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAD2Cf4tгнаecDemZKONzj470Yc4nuaaDElXoVZud92v
+LRbADmHfrjRStR6STPZE1Stqj76f7JQrIij2rlzYImg7nrGANHAE0Ftd6cKoRmsi15MVB/xm1Y5raQt
9hlCKY2qtsZtdvKVJvJGWB3sAh1rUsBwbkRtK/MdF6jTh30LS6nGjCC67UQY1pIfoeiV/KwOm8R7Jj
C53GIEpxIXaY9hrQVtUxzQbvzb0bx9aLEu4eXL9E2yH/tm1/MxnfdSRc0EjGXlip95B5J+XH1/aV0Dyg
e+dt0ehRAX0hSHYzWBS2LcjUblK9/1tuyGilApD64Hd0S9DWMTeMApcJjH590srrwmtzgmH2Z9oeVw24
blpd60jGgjBqcvaTzjyw610deD0ZVDht1H7CxlL9AQL++Pd5gyuqaUDEAi5801ZLuxa5E+38hN2HSP1
o5S83FpGvj17YII/3MSrkXaPszSjgqkieDadaSOYLWlPqWd2d8DMkp+uDA6ns+svPsVFhE+yb/A7wnqs
rxAwwsxJIYm4I8Q57v6Y8S7/jfKFmt40cKf1tDUZS526tYqLgKAIMGW0/Iwx+7mL7uDnjl9GowA/l9B+
+ASJpm7cZr/7M1vxq+HuLaCiuILXKSW/rzJYPbZl+9/owOYpgj0MDYONAnco+q/J6rHHRlWl/BY8US+9
ow== devasc@labvm
```

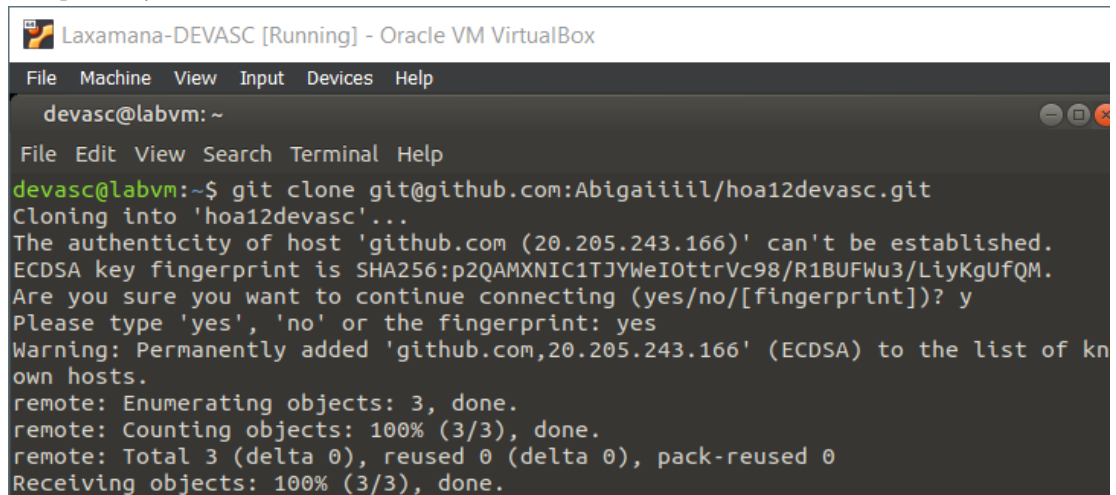
c. Adding new SSH Key in Git



d. Create a new repository

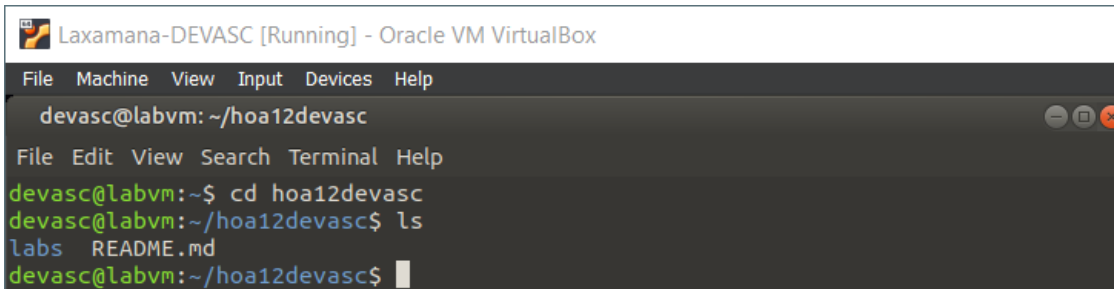


- e. Clone repository to local machine (devasc)



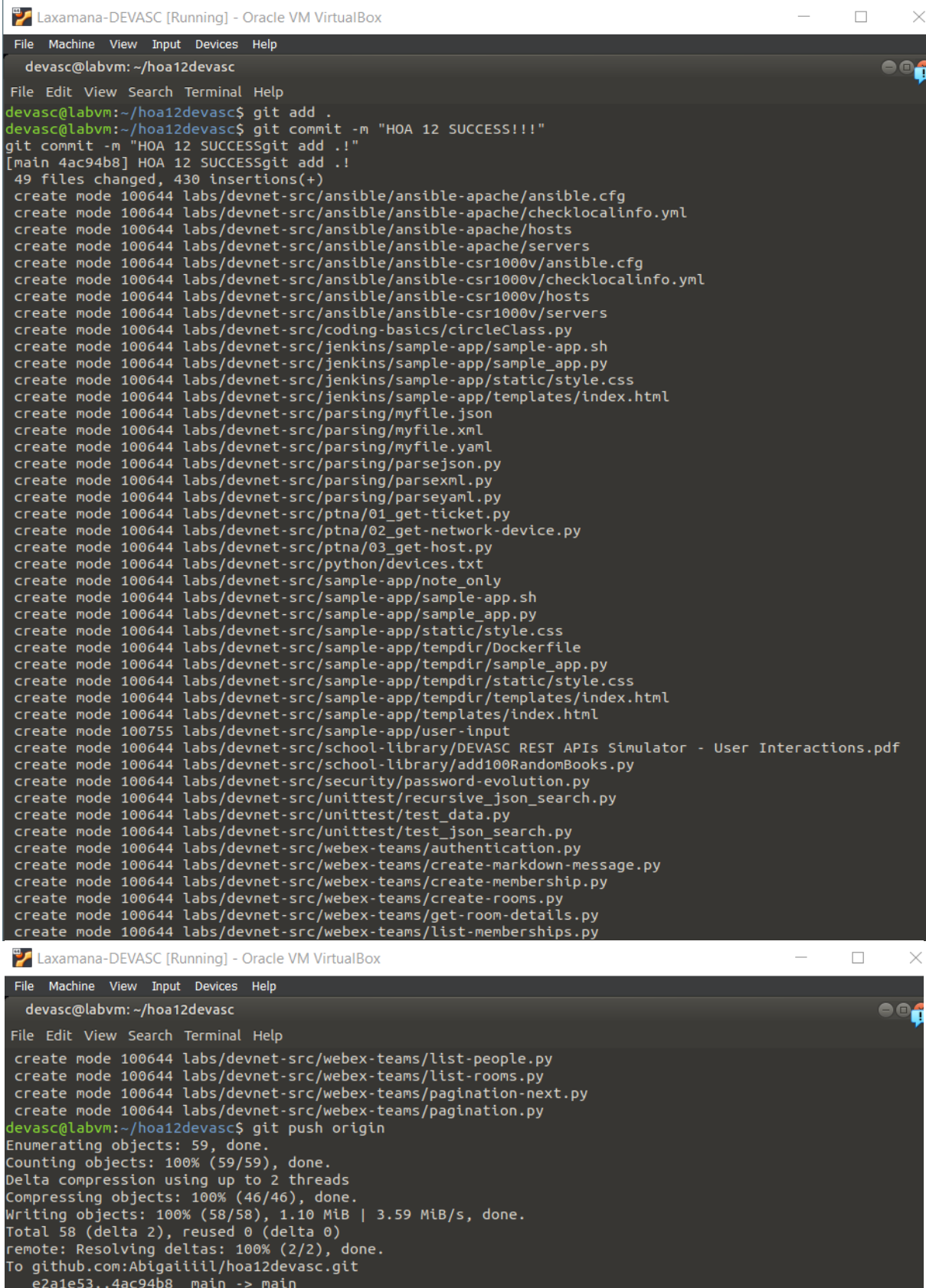
```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~
File Edit View Search Terminal Help
devasc@labvm:~$ git clone git@github.com:Abigaiiil/hoa12devasc.git
Cloning into 'hoa12devasc'...
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'github.com,20.205.243.166' (ECDSA) to the list of kn
own 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.
```

- f. Copy the labs file to the repository



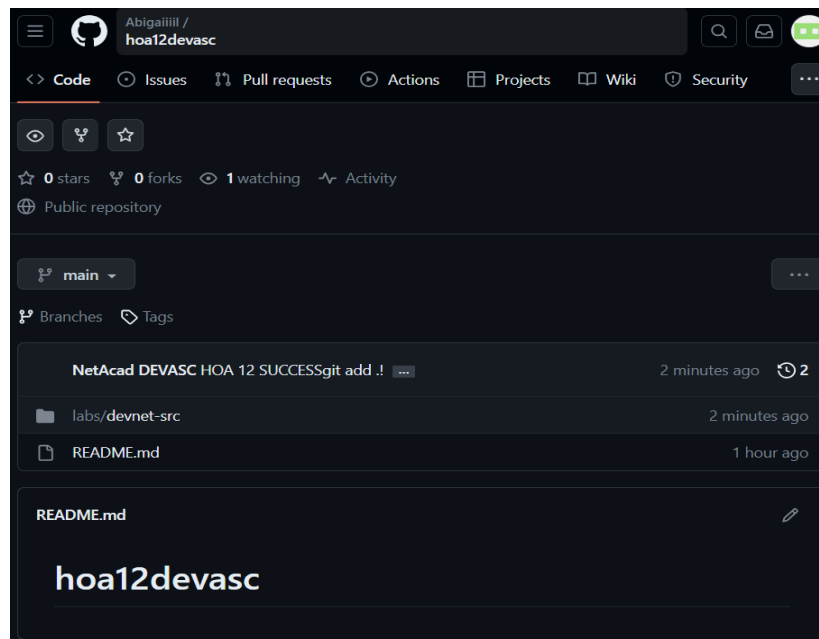
```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/hoa12devasc
File Edit View Search Terminal Help
devasc@labvm:~$ cd hoa12devasc
devasc@labvm:~/hoa12devasc$ ls
labs  README.md
devasc@labvm:~/hoa12devasc$
```


g. Sync with Git repository



```
Laxamana-DEVASC [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/hoa12devasc
File Edit View Search Terminal Help
devasc@labvm:~/hoa12devasc$ git add .
devasc@labvm:~/hoa12devasc$ git commit -m "HOA 12 SUCCESS!!!"
git commit -m "HOA 12 SUCCESS"
git add .!
[main 4ac94b8] HOA 12 SUCCESS
49 files changed, 430 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/note_only
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/tmpdir/Dockerfile
create mode 100644 labs/devnet-src/sample-app/tmpdir/sample_app.py
create mode 100644 labs/devnet-src/sample-app/tmpdir/static/style.css
create mode 100644 labs/devnet-src/sample-app/tmpdir/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 Interactions.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
devasc@labvm:~/hoa12devasc
File Machine View Input Devices Help
File Edit View Search Terminal Help
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
devasc@labvm:~/hoa12devasc$ git push origin
Enumerating objects: 59, done.
Counting objects: 100% (59/59), done.
Delta compression using up to 2 threads
Compressing objects: 100% (46/46), done.
Writing objects: 100% (58/58), 1.10 MiB | 3.59 MiB/s, done.
Total 58 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To github.com:Abigaiiil/hoa12devasc.git
e2a1e53..4ac94b8 main -> main
```

h. Updated Git repository



Part 8: Conclusion

To summarize it all up, I have learned about creating web application using docker container. The challenging part in this activity is the utilization of different virtual machine. Instead of accomplishing this activity in ubuntu, we have been required to do this using devasc which is a new environment for us. Even so, it is not very difficult to work in this environment as it is like ubuntu environment. Upon following the step-by-step instructions, tasks have been clearer and more understandable. However, we cannot avoid errors but I was able to fix it. Overall, this activity made me realize how many different functions we can do using docker containers.