

<b>Name:</b> Buenvendida, Ken Benedict D.	<b>Date Performed:</b> Nov 20, 2023
<b>Course / Section:</b> CpE31S4 - CPE232	<b>Date Submitted:</b> Nov 20, 2023
<b>Instructor:</b> Engr. Jonathan Taylar	<b>Semester and SY:</b> 1st Semester 2023-2024

## 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
```

```
ken@controlNode:~/H0A12$ mkdir labs  
ken@controlNode:~/H0A12$ cd labs  
ken@controlNode:~/H0A12/labs$ mkdir devnet-src  
ken@controlNode:~/H0A12/labs$ cd devnet-src  
ken@controlNode:~/H0A12/labs/devnet-src$ mkdir sample-app  
ken@controlNode:~/H0A12/labs/devnet-src$ cd sample-app  
ken@controlNode:~/H0A12/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
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ sudo 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
```

```
GNU nano 6.2  
#!/bin/bash
```

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

### Step 4: Add simple bash commands to the script.

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

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

```
GNU nano 6.2  
#!/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$
```

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ bash user-input.sh
Enter your name: ken
Your name is ken
```

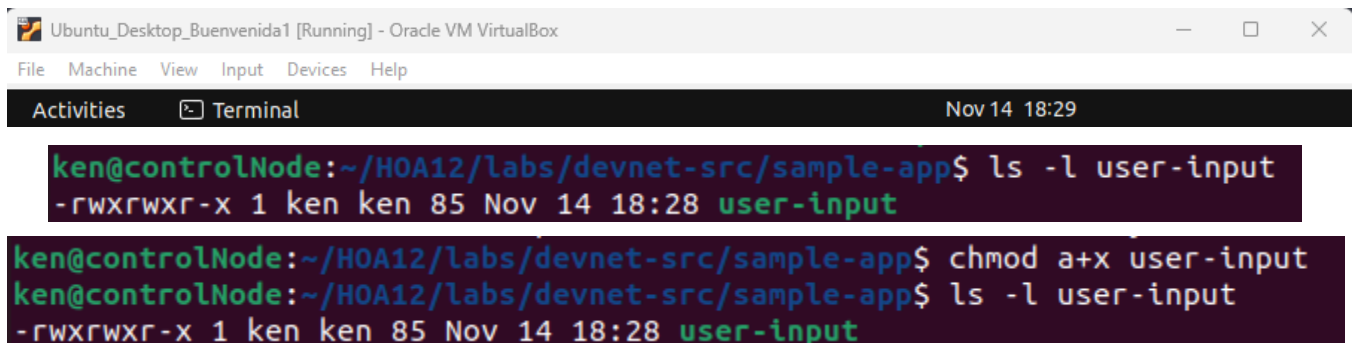
### Step 7: Change the mode of the script to an executable file for all users.

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
```



I don't know why my VM is displaying November 14 instead of the current date today po sir.

### 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
```

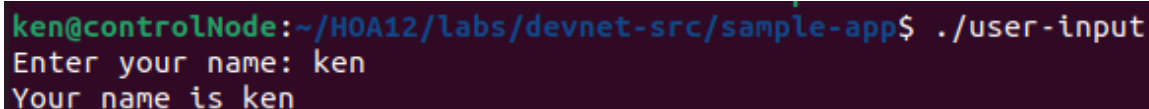
```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ mv user-input.sh user-input
```

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ ls -l user-input
-rwxrwxr-x 1 ken ken 85 Nov 14 18:28 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$
```



```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ ./user-input
Enter your name: ken
Your name is ken
```

### 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
```

```
ken@controlNode:~$ pip3 install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
    _____ 99.7/99.7 KB 690.2 kB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
    _____ 226.7/226.7 KB 39.5 MB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    _____ 97.9/97.9 KB 9.1 MB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.1.2
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/lib/python3/dist-packages
(from Jinja2>=3.1.2->flask) (2.0.1)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
pipx 1.0.0 requires argcomplete>=1.9.4, but you have argcomplete 1.8.1 which is incompatible.
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
```

### 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**.

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ vim sample_app.py
```

### 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
```

```
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__)
```

```
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"
```

```
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)
```

```
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)
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ python3 sample_app.py
* Serving Flask app 'sample_app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://10.0.2.15:8080
Press CTRL+C to quit
```

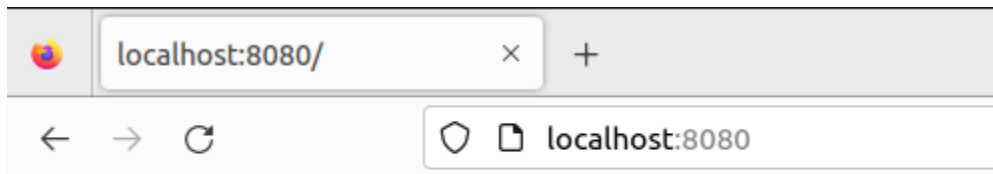
### Step 8: Verify the server is running.

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:

**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.



- 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$
```

```
ken@controlNode:~$ curl http://0.0.0.0:8080
You are calling me from 127.0.0.1/nken@controlNode:~$
```

### Step 9: Stop the server.

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



```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ python3 sample_app.py
* Serving Flask app 'sample_app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://10.0.2.15:8080
Press CTRL+C to quit
127.0.0.1 - - [13/Nov/2023 22:59:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 22:59:41] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [13/Nov/2023 23:00:29] "GET / HTTP/1.1" 200 -
```

### 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>
```

```
GNU nano 6.2 index.h
<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 showing the GNU nano 6.2 editor. The code displayed is: 

```
body {  
    background: blue;  
}
```

### 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)
```

A screenshot of a terminal window showing the updated Python code for sample\_app.py. The code is: 

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

## Lab - Build a Sample Web App in a Docker Container

\* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ python3 sample_app.py
* Serving Flask app 'sample_app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://10.0.2.15:8080
Press CTRL+C to quit
127.0.0.1 - - [13/Nov/2023 22:59:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 22:59:41] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [13/Nov/2023 23:00:29] "GET / HTTP/1.1" 200 -
```

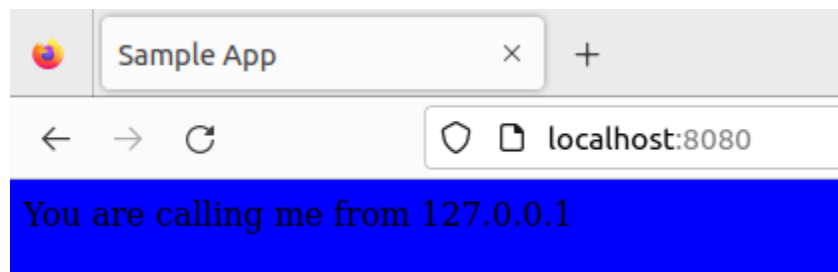
**Note:** If you got Traceback output and an error with the message with something like **OSError: [Errno 98] Address already in use**, then you did not shutdown your previous server. Return to the terminal window where that server is running and press CTRL+C to end the server process. Re-run your script.

### 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**



- 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$
```

```
ken@controlNode:~$ curl http://0.0.0.0:8080
<html>
<head>
  <title>Sample App</title>
  <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
  <hi>You are calling me from 127.0.0.1</hi>
</body>
</html>ken@controlNode:~$
```

### 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
```

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ sudo nano sample-app.sh
```

```
GNU nano 6.2
#!/bin/bash

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

### Step 2: Copy the website directories and sample\_app.py to the temporary directory.

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

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



```
GNU nano 6.2
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

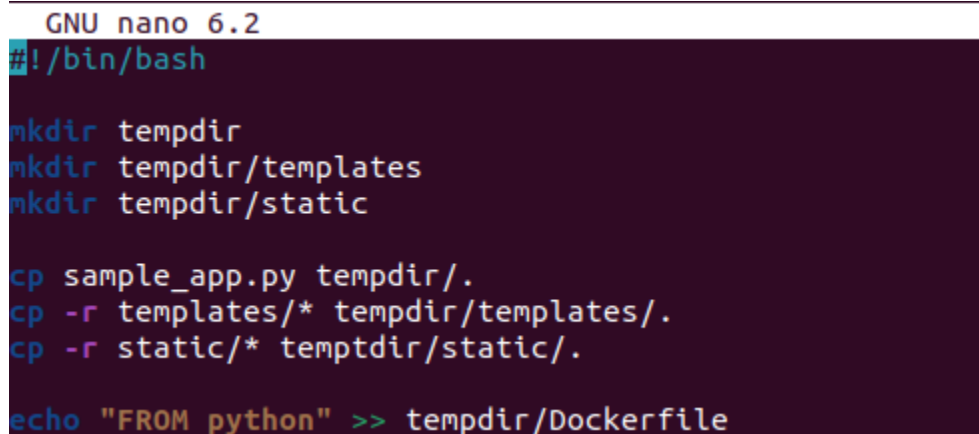
cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.
```

### 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
```



```
GNU nano 6.2
#!/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
```

- 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
```

```
GNU nano 6.2
#!/bin/bash

mkdir tmpdir
mkdir tmpdir/templates
mkdir tmpdir/static

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

echo "FROM python" >> tmpdir/Dockerfile
echo "RUN pip install flask" >> tmpdir/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/" >> tmpdir/Dockerfile
echo "COPY ./templates /home/myapp/templates/" >> tmpdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tmpdir/Dockerfile
```

```
GNU nano 6.2 sample-
#!/bin/bash

mkdir tmpdir
mkdir tmpdir/templates
mkdir tmpdir/static

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

echo "FROM python" >> tmpdir/Dockerfile
echo "RUN pip install flask" >> tmpdir/Dockerfile
echo "COPY ./static /home/myapp/static/" >> tmpdir/Dockerfile
echo "COPY ./templates /home/myapp/templates/" >> tmpdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tmpdir/Dockerfile
```

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

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

```
GNU nano 6.2 sample-  
#!/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
```

```
GNU nano 6.2 sample-app  
#!/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.

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 .
```

```
GNU nano 6.2 sample-app
#!/bin/bash

mkdir tmpdir
mkdir tmpdir/templates
mkdir tmpdir/static

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

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

cd tmpdir
docker build -t sampleapp .
```

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

- 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**).



```

GNU nano 6.2 sample-a
#!/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

```

- 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
```

```

GNU nano 6.2 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

```

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
```

## Lab - Build a Sample Web App in a Docker Container

```
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          NAMES          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$
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ bash ./sample-app.sh
mkdir: cannot create directory 'tmpdir': File exists
cp: cannot create regular file 'temptdir/static/.': No such file or directory
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   5.12kB
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: sha256:7b8d65a924f596eb65306214f559253c468336bcae09fd575429774563460caf
Status: Downloaded newer image for python:latest
---> 2acccf902fa3
Step 2/7 : RUN pip install flask
---> Running in 340ed6d9af60
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c23096649b908c809c69388a805a571a3
bea44362fe87e33fc3afa01f/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/c3/fc/254c3e9b5feb89ff5b9076a232
18dafbc99c96ac5941e900b71206e6313b/werkzeug-3.0.1-py3-none-any.whl.metadata
  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 3.5 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/00/2e/d53fa4befbf2cfa713304affc7ca7
80ce4fc1fd8710527771b58311a3229/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.metadata
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/51/94/9a04085114ff2c24f7424dbc890a281d73c5a74ea935dc2e69c66a3bd558/MarkupSafe-2.1.3-cp312-cp312-manylinux_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 1.7 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 1.5 MB/s eta 0:00:00
Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
226.7/226.7 kB 18.7 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 with 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 340ed6d9af60
--> c903bab38481
Step 3/7 : COPY ./static /home/myapp/static/
--> a83df96e171f
Step 4/7 : COPY ./templates /home/myapp/templates
--> 42a37f663dd9
Step 5/7 : COPY sample_app.py /home/myapp/
--> 648fb37fb69c
Step 6/7 : EXPOSE 8080
--> Running in 262dd0f1937d
Removing intermediate container 262dd0f1937d
--> e46d7d495a52
Step 7/7 : CMD python3 /home/myapp/sample_app.py
--> Running in ea7998ccfcbf
Removing intermediate container ea7998ccfcbf
--> 326be499430c

Removing intermediate container 340ed6d9af60
--> c903bab38481
Step 3/7 : COPY ./static /home/myapp/static/
--> a83df96e171f
Step 4/7 : COPY ./templates /home/myapp/templates
--> 42a37f663dd9
Step 5/7 : COPY sample_app.py /home/myapp/
--> 648fb37fb69c
Step 6/7 : EXPOSE 8080
--> Running in 262dd0f1937d
Removing intermediate container 262dd0f1937d
--> e46d7d495a52
Step 7/7 : CMD python3 /home/myapp/sample_app.py
--> Running in ea7998ccfcbf
Removing intermediate container ea7998ccfcbf
--> 326be499430c
Successfully built 326be499430c
Successfully tagged sampleapp:latest
300f8ccf2b31504c85d8e68c4a259073518ce743dc921cd1dceaf1b0c57e49b
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
300f8ccf2b31   sampleapp     "/bin/sh -c 'python3..." 15 seconds ago Up Less than a second    0.0.0.0:8080->8080/tcp, :::8080->
8080/tcp      samplerunning
96213b60dccc   hello-world   "/hello"                 5 minutes ago  Exited (0) 4 minutes ago
sleepy_bhaskara
e4b1dc764fbf   cb65079f7a9a  "/bin/sh -c 'apt ins..." 12 hours ago   Up 12 hours
naughty_buck
334a51613b68   nginx        "/docker-entrypoint..." 12 hours ago   Exited (0) 12 hours ago
quirky_mclaren
e773e6009da0   ubuntu       "/bin/bash"              12 hours ago   Exited (0) 12 hours ago
happy_chebyshev
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$
```

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

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

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

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ ls tempdir/  
Dockerfile sample_app.py static templates
```

- 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
```

```
ken@controlNode:~/H0A12/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
```

- 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.

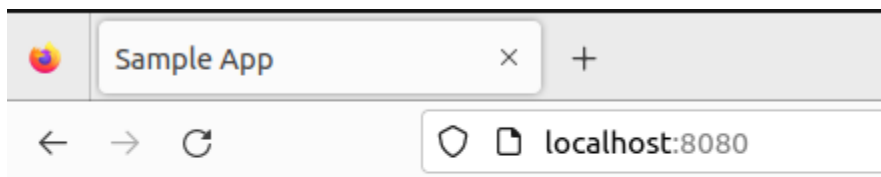
```
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt  
devasc@labvm:~/labs/devnet-src/sample-app$
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker ps -a >> running.txt  
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ cat running.txt  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS  
300f8ccf2b31   sampleapp     "/bin/sh -c 'python3..." 2 minutes ago  Up About a minute  0.0.0.0:8080->8080/tcp, :::8080->8  
080/tcp       samplerunning  
96213b60dccc   hello-world   "/hello"                7 minutes ago  Exited (0) 6 minutes ago  
e4b1dc764fbf   cb65079f7a9a "/bin/sh -c 'apt ins..." 12 hours ago   Up 12 hours  
naughty_buck  
334a51613b68   nginx         "/docker-entrypoint..." 12 hours ago   Exited (0) 12 hours ago  
quirky_mclaren  
e773e6009da0   ubuntu       "/bin/bash"              12 hours ago   Exited (0) 12 hours ago  
happy_chebyshev  
ken@controlNode:~/H0A12/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$
```



- 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>ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ ip a
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:ce:c5:06 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
        valid_lft 82588sec preferred_lft 82588sec
    inet6 fe80::a5f5:6b4:433f:54bd/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:be:94:df brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.104/24 brd 192.168.56.255 scope global noprefixroute enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::9d7:75e9:f9f0:dbd5/64 scope link noprefixroute
        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:af:9f:5d:cd 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:a5ff:fe9f:5dcd/64 scope link
        valid_lft forever preferred_lft forever
12: veth820e1c1@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether ba:af:4f:93:3f:8a brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b8af:4fff:fe93:3f8a/64 scope link
        valid_lft forever preferred_lft forever
22: veth0dc42af@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 8e:1f:67:9a:c9:8b brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::8c1f:67ff:fe9a:c98b/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.

- 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:/#
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@300f8ccf2b31:/#
```

- 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:/#
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@300f8ccf2b31:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@300f8ccf2b31:/#
```

- 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:/#
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@300f8ccf2b31:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@300f8ccf2b31:/# ls home/myapp/
sample_app.py  static  templates
root@300f8ccf2b31:/#
```

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

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

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker exec -it samplerunning /bin/bash
root@300f8ccf2b31:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@300f8ccf2b31:/# ls home/myapp/
sample_app.py  static  templates
root@300f8ccf2b31:/#
```

### 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$
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
300f8ccf2b31	sampleapp	"/bin/sh -c 'python3..."	5 minutes ago	Exited (137) 18 seconds ago		samplerunning
96213b60dccc	hello-world	"/hello"	10 minutes ago	Exited (0) 10 minutes ago		sleepy_bhaskara
e4b1dc764fbf	cb65079f7a9a	"/bin/sh -c 'apt ins..."	12 hours ago	Up 12 hours		naughty_buck
334a51613b68	nginx	"/docker-entrypoint..."	12 hours ago	Exited (0) 12 hours ago		quirky_mclaren
e773e6009da0	ubuntu	"/bin/bash"	12 hours ago	Exited (0) 12 hours ago		happy_chebyshev

```
ken@controlNode:~/H0A12/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$
```

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
300f8ccf2b31	sampleapp	"/bin/sh -c 'python3..."	7 minutes ago	Up About a minute	0.0.0.0:8080->8080/tcp, :::8080->8080/tcp

```
ken@controlNode:~/H0A12/labs/devnet-src/sample-app$
```

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

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker rm samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
devasc@labvm:~/labs/devnet-src/sample-app\$			

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ docker rm samplerunning
samplerunning
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
96213b60dccc	hello-world	"/hello"	12 minutes ago	Exited (0) 12 minutes ago		sleepy_bhaskara
e4b1dc764fbf	cb65079f7a9a	"/bin/sh -c 'apt ins...'"	12 hours ago	Exited (137) 48 seconds ago		naughty_buck
334a51613b68	nginx	"/docker-entrypoint..."	12 hours ago	Exited (0) 12 hours ago		quirky_mclaren
e773e6009da0	ubuntu	"/bin/bash"	12 hours ago	Exited (0) 12 hours ago		happy_chebyshev

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$
```

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ken/HOA12/labs/devnet-src/sample-app/.git/
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$
```

```
ken@controlNode:~/HOA12/labs/devnet-src/sample-app$ ll
total 36
drwxrwxr-x 5 ken ken 4096 Nov 13 23:49 ./
drwxrwxr-x 3 ken ken 4096 Nov 13 13:37 ../
-rw-rw-r-- 1 ken ken 938 Nov 13 23:37 running.txt
-rw-rw-r-- 1 ken ken 254 Nov 13 23:08 sample_app.py
-rw-r--r-- 1 root root 678 Nov 13 23:23 sample-app.sh
drwxrwxr-x 2 ken ken 4096 Nov 13 23:06 static/
drwxrwxr-x 4 ken ken 4096 Nov 13 23:32 tempdir/
drwxrwxr-x 2 ken ken 4096 Nov 13 23:04 templates/
-rwxrwxr-x 1 ken ken 85 Nov 13 13:38 user-input*
```

```
ken@controlNode:~/HOA12$ git add *
ken@controlNode:~/HOA12$ git commit -m "HOA12"
[main 561996c] HOA12
 9 files changed, 87 insertions(+)
 create mode 100644 labs/devnet-src/sample-app/running.txt
 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/templates/index.html
 create mode 100644 labs/devnet-src/sample-app/templates/index.html
 create mode 100755 labs/devnet-src/sample-app/user-input
ken@controlNode:~/HOA12$ git push origin
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.97 KiB | 1.97 MiB/s, done.
Total 15 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:KBDBuenvenida/HOA12.git
   56f47f1..561996c  main -> main
ken@controlNode:~/HOA12$
```

### Conclusion:

This activity has made me learn about the usage of docker by launching it to a remote server. I was able to learn about how to create a bash script that can launch a fully functional simple web application using dockerfile. I was also able to learn about the usage of flask which is a web framework that is installed on the top of a python container that allows it to run a web page. Lastly, I had encountered some problems along the way but I was able to troubleshoot to make this activity work.