WEB-APP

**Build a Sample Web App in a Docker Container**

**Objectives**

Part 1: Launch a code editor (pycharm, itellij)  
Part 2: Create a Simple Bash Script  
Part 3: Create a Sample Web App  
Part 4: Create a Bash Script to Build and Run a Docker Container

Part 5: Build, Run, and Verify the Docker Container

**Part 2: Create a Simple Bash Script**

Bash script 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.

run-docker.sh file creates the Dockerfile and creates necessary directories for the docker folder.

When we execute the run-docker.sh file it creates the below items:

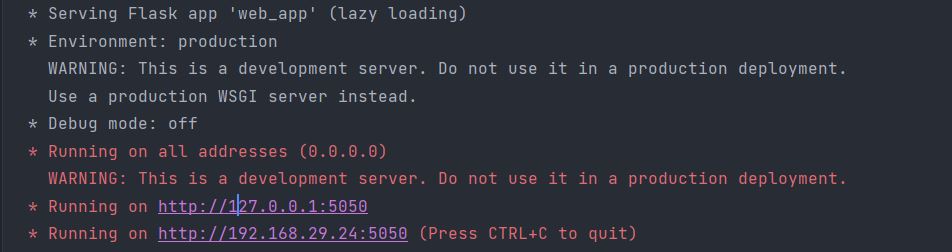
* Creates the “Dockerfile” under the root folder of the project.
* Creates all the directories like layers , Static.
* Copies all the require documents to the directories under the docker folder

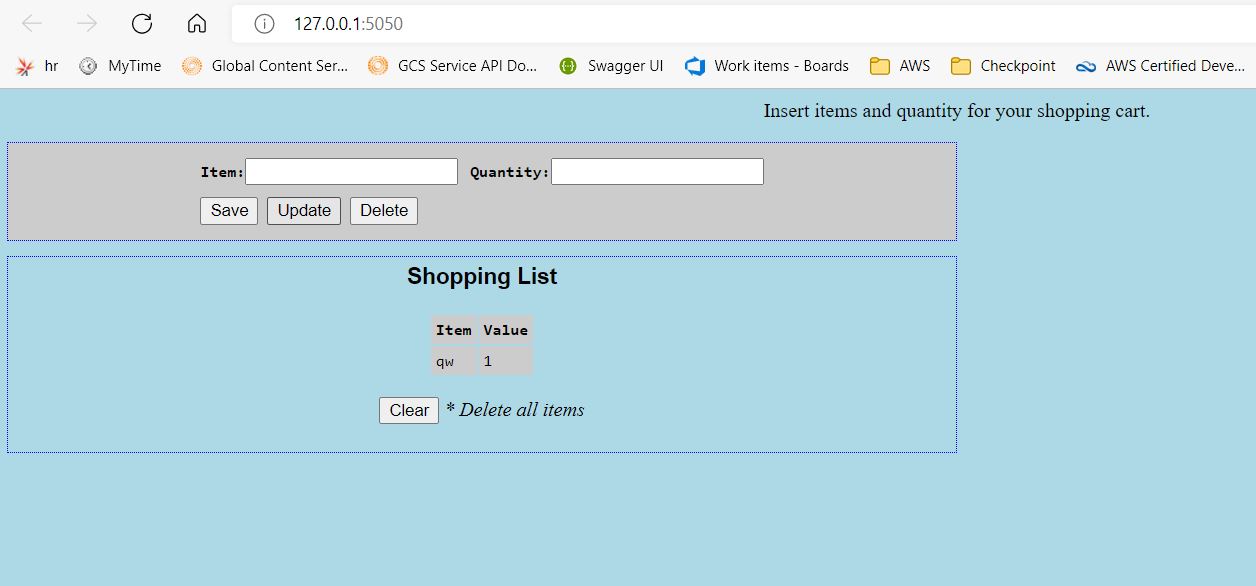
**Part 3: Create a Simple Web App**

Before we can launch an application in a Docker container, we first need to have the app. In this part, I am creating a very simple Python script that will display the order details of the client when the client visits the web page.

* 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.
* 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.
* Open the **web\_app.py** file located in the **/lambda** directory.

Finally, configure Flask to run the app locally at http://0.0.0.0:5050, which is also http://localhost:5050. Save your script and run it from the command line. Here we can see the following output which indicates that your “web\_app” server is running.





**Part 4: Create a Bash Script to Build and Run a Docker Container**

Add the commands to the **run-docker.sh** file to switch to the **docker** directory and build the Docker container. The **docker build** command **-t** option allows you to specify the name of the container.

docker build -t sampleapp .

Add the **docker run** command to the **run-docker.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.
* **--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.

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

**Part 5: Build, Run, and Verify the Docker Container**

In this part, I 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.

Output:

**bash ./run-docker.sh**

Sending build context to Docker daemon 6.144kB  
Step 1/7 : FROM python  
latest: Pulling from library/python

90fe46dd8199: Pulling fs layer

35a4f1977689: Pulling fs layer

bbc37f14aded: Pull complete

74e27dc593d4: Pull complete

4352dcff7819: Pull complete

deb569b08de6: Pull complete

98fd06fa8c53: Pull complete

7b9cc4fdefe6: Pull complete

512732f32795: Pull complete

Digest: sha256:ad7fb5bb4770e08bf10a895ef64a300b288696a1557a6d02c8b6fba98984b86a

Status: Downloaded newer image for python:latest

---> 4f7cd4269fa9

Step 2/7 : RUN pip install flask

---> Running in 32d28026afea

Collecting flask

Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)

Collecting click>=5.1

Downloading click-7.1.2-py2.py3-none-any.whl (82 kB)

Collecting Jinja2>=2.10.1

Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)

Collecting Werkzeug>=0.15

Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)

Collecting itsdangerous>=0.24

Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)

Collecting MarkupSafe>=0.23

Downloading MarkupSafe-1.1.1-cp38-cp38-manylinux1\_x86\_64.whl (32 kB)

Installing collected packages: click, MarkupSafe, Jinja2, Werkzeug, itsdangerous,

flask

Successfully installed Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2

flask-1.1.2 itsdangerous-1.1.0

Removing intermediate container 32d28026afea

---> 619aee23fd2a

Step 3/7 : COPY ./static /home/myapp/static/

---> 15fac1237eec

Step 4/7 : COPY ./templates /home/myapp/templates/

---> dc807b5cf615

Step 5/7 : COPY sample\_app.py /home/myapp/

---> d4035a63ae14

Step 6/7 : EXPOSE 8080

---> Running in 40c2d35aa29a

Removing intermediate container 40c2d35aa29a

---> eb789099a678

Step 7/7 : CMD python3 /home/myapp/web\_app.py

---> Running in 41982e2c6209

Removing intermediate container 41982e2c6209

---> a2588e9b0593

Successfully built a2588e9b0593

Successfully tagged sampleapp:latest

8953a95374ff8ebc203059897774465312acc8f0ed6abd98c4c2b04448a56ba5

CONTAINER ID

STATUS

IMAGE PORTS

COMMAND

CREATED

1 second ago

sampleapp

Up Less than a second 0.0.0.0:8080->8080/tcp samplerunning

**Access and explore the running container.**

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**.

**docker exec -it samplerunning /bin/bash**

root@:/#

Once you see all the copied items inside the docker container we can start the docker.

You can stop a container with the **docker stop** command.

Docker stop <container\_id>

You can remove a container once the code changes done.

Docker rm <container\_id>