**Containerizing Flask App with Docker**

**What You'll Do**

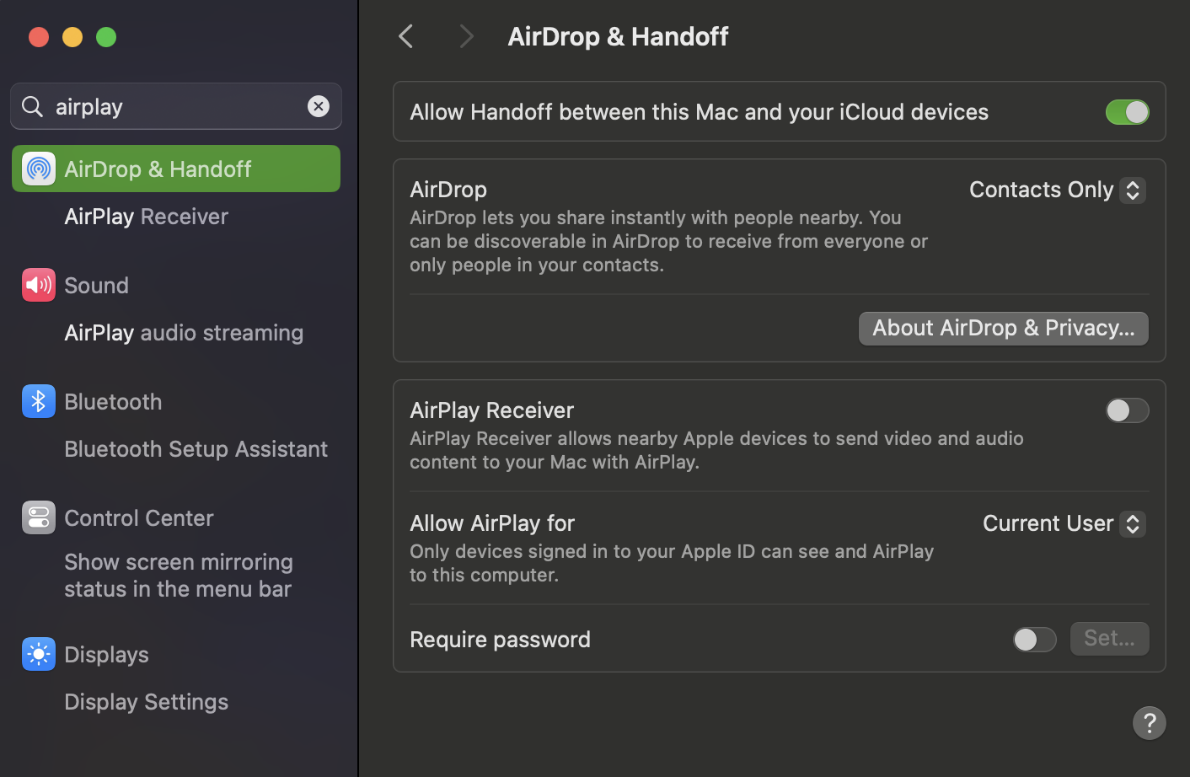
* This exercise will be similar to the one where we walked through containerizing a Django application with Docker.
* In this exercise, we will walk through an example of containerizing a Python application that has been built with the Flask microframework.
* Again, we will make use of the Docker VS Code tools to build the Docker image.
* You will also practice the Docker CLI commands you learned in the previous exercise.

**Instructions**

**Environment setup**

* Download this file:

[hello\_docker\_flask\_vscode.zip](https://learn.nucamp.co/pluginfile.php/88933/mod_book/chapter/5243/hello_docker_flask_vscode.zip?time=1699797363949)

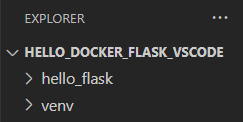
* Unzip this file. Once unzipped, you should have a folder named **hello\_docker\_flask\_vscode**.
* Depending on how you unzipped it, you may have a nested folder inside it, also named **hello\_docker\_flask\_vscode**. If so, this inner folder is the folder that you want to use for the next step, **not** the outer one.
* Take the **hello\_docker\_flask\_vscode/**folder and place it in your **3-DevOps/week1/**folder.
* Open VS Code to this **hello\_docker\_flask\_vscode/**folder.
  + This folder should contain a **hello\_flask/**folder, which is the project folder for an example project using the Flask microframework for Python.
  + There should only be two files in this folder: **app.py** and **requirements.txt** .
  + Notice the difference between a microframework and a framework - Flask requires far less to get started! (But it does less for you.)
* Make sure that **Docker Desktop** is running before you continue.
* If you are using a Mac, make sure that **AirPlay**is toggled off in System Preferences, as it uses the same port as Flask.
  + Do this under System Settings => AirDrop & Handoff => AirPlay Receiver.  
    

**Confirm working application in virtual environment**

**Create virtual environment**

* As we did with the Django app, we will confirm that the Flask app works correctly *without* Docker first.
* We will again confirm this in a virtual environment.
* Open the integrated terminal in VS Code. This should open automatically to the **hello\_docker\_flask\_vscode/**folder.
* In this folder, create a virtual environment for testing this project:

python -m venv venv



* You should now see a **venv/** folder inside your **hello\_docker\_flask\_vscode/** folder.

**Select Python interpreter**

* Open the VS Code Command Palette. You can do this in two ways:
  + From the **View** menu, select **Command Palette**.
  + Or, use the keyboard shortcut:**Ctrl-Shift-P** for Windows /**Cmd-Shift-P** for macOS.
* In the search field that appears, type in as much of the text **Python: Select Interpreter** as you need to until you see the option appear below it:
* Click on that option. You should next see a list of interpreters:
* Select the one that with the name ending in **('venv':venv)**. It will most likely be marked as "Recommended" as you see above.
* Close the currently open terminal by clicking the trash can icon:

**About the Flask app**

* In the **hello\_flask/** folder is a single Python file that composes our basic "Hello World" Flask app. This is named **app.py**.
* Let's take a look at this file. We will not be making any edits to it, only seeing what it contains.
* Go ahead and open it in VS Code. You should see the following contents:
* Recall what you learned in the previous course regarding the use of route decorators in implementing endpoints in Flask.
  + The route decorator **@app.route('/')**here binds the '/' route (that is, the root directory of the server) to the function **hello\_world()**, thus that the hello\_world() function is run whenever an end-user visits the server's root directory.
  + Note: The default HTTP method for the route decorator is **GET**if none are specified.
* The **if \_\_name\_\_ == '\_\_main\_\_':**statement is boilerplate code that's very often used in Python to invoke the main function of a file. It is especially important when working with Python files that are imported as modules. [This StackOverflow thread](https://stackoverflow.com/questions/419163/what-does-if-name-main-do) contains an excellent discussion on it, as does [this article](https://www.freecodecamp.org/news/if-name-main-python-example/) .
* The IP address **0.0.0.0** is commonly used in development. It is a way to have a server listen on all the IPv4 addresses on the local machine. It is similar to the **localhost** IP address. See: <https://en.wikipedia.org/wiki/0.0.0.0>
* Close the **app.py** file.

**Install requirements and launch Flask app**

* Open a new terminal. VS Code will automatically activate the virtual environment, and you should see**(venv)**in the command prompt:
* Navigate to the **hello\_flask/**folder, which is the project root:

cd hello\_flask

* A**requirements.txt**file has been provided in this folder along with the Flask project. This file lists the Python packages required by this project.
* First, update the pip installation in your virtual environment:

python -m pip install --upgrade pip==24.3.1

* Next, install the packages listed in the **requirements.txt** file with the following command:

python -m pip install -r requirements.txt

* Run the Flask application with the following command:

python app.py

* You should see the output below in the console:
* Recall that the server is listening on all local addresses (as per the setting **0.0.0.0** in**app.py**). The Flask server will run by default on port **5000**.
* Also recall that the**app.py** file sets its function **hello\_world()** to run when the root directory of the server is requested.
* Open a web browser and go to this address: <http://127.0.0.1:5000/>
* If all went well, you should see the following in the browser:
* This confirms that the Flask application is working.
* In your terminal, stop the server with **ctrl-c**.
* We can now move on to creating a Docker image for it.

**Dockerize the Flask application**

* We will again use the Docker extension for VS Code for the dockerization process.
* First, we need to have VS Code open to the project's root folder. This is the **hello\_flask**folder.
* Open the **hello\_docker\_flask\_vscode/hello\_flask/**folder in VS Code. Your VS Code Explorer should look like this:
* Next, we will generate the necessary Docker files.

**Add Docker files to workspace**

* Open the VS Code Command Palette (Ctrl/Cmd-Shift-P) and type in/select **Docker: Add Docker Files to Workspace…**:
* You will be asked to **Select Application Platform**. Choose **Python: Flask**:
* Then for the **entry point**, select **app.py**:
* For the **port**, VS Code will have automatically populated the field with **5000**, as it knows that this is Flask's default port. You only need to press **Enter**:  
  + **Mac users**: It may automatically populate with **5002**, change it to **5000**.
* Finally, you will be asked this question:
* Press **Enter** to select the default option of **No.**
* VS Code will then generate several files, including a file named **Dockerfile** and a file named **.dockerignore**. It will also have created a **.vscode/** folder:

**Confirm Dockerfile configuration for Gunicorn**

* Open the generated **Dockerfile**if it is not automatically opened.
* As in the previous exercise with Django, Gunicorn is also used as the default web server for the Flask application.
* A Docker command is added to bind Gunicorn to the entry point of the Flask web application, **app.py**.
* Confirm that the final line in the Dockerfile appears as follows:

CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]

* If it does not look exactly like this line, update it and save the file.
* You can now close the Dockerfile.

**Build Docker image**

* We will now build the Docker image for this application, using the VS Code integrated tools for Docker.
* Make sure that VS Code is open to the **hello\_flask** folder.
* Open the **Command Palette** (Ctrl/Cmd-Shift-P).
* Type in **Docker Images** to filter the Docker Image commands.
* Select **Docker Images: Build Image…**
* The Build command will execute, and a console for docker-build will open in your terminal panel.
  + You will see several "DONE …" statements in the console output as the build completes various steps.
  + This will take a few minutes to build all the layers in the Docker image.
  + Be sure to examine the output in the terminal for errors. All errors need to be resolved for a successful build.
* When you see output similar to this in the terminal...:
* ...the build is complete, and you can press any key to close the docker-build terminal.

**Run Docker image**

* Open the Command Palette (Ctrl/Cmd-Shift-P) and type in/select **Docker Images: Run**:
* You will be asked to **Select image group**. Select **helloflask.**
* You will then be asked to **Select image**. Select **latest**.
* Your terminal should then show this output:
* You can press any key to close the above terminal.
* Confirm that the image is running by going to**Docker Desktop**, and click **Containers/Apps** on the left side menu.
* You should see a container built from the **helloflask:latest** image running there, at port **5000**. The name will be randomized:
* If you check the **Images**panel of Docker Desktop, you will also see that the Flask image is labeled as **IN USE**, with the **latest** tag**:**
* In your web browser, confirm that the containerized Flask app can now be viewed at [http://localhost:5000](http://localhost:5000/).

**Practice Docker CLI commands**

* Let's practice some of the CLI commands you learned about in the last exercise, using this new Docker image.

**Docker logs**

* First, recall that Docker logs can display status information about the Docker container that is running.
* To view the logs, we will need the **Container ID**. Obtain it by running this command in a bash terminal:

docker ps

* The output should include a Container ID for the**helloflask:latest** image:
* Your Container ID will be different from the one above. Go ahead and copy it, then use it in the next command.
* To view the Docker logs, run the command:

docker logs <ContainerID>

* You should see output similar to this:
* Yours may have more or less of some of the later messages, but at least the first three log messages should be the same.

**Docker run, stop**

* Let's try using the Docker run and stop commands from the command line, instead of VS Code's Command Palette.
* Enter this command:

docker run --rm -d -p 5000:5000/tcp helloflask:latest

* Notice that this is the same command issued by using the VS Code Command Palette to run the Docker image.
* This should result in an error because we already have the image running on that port:
* To stop the Docker container, enter the command below, substituting in your running container's ID:

docker stop <ContainerId>

* Example output:
* Note that if the command was successful, then the Docker CLI will simply echo the Container ID back to you, as you can see in the above screenshot.
* You can use **docker ps** to confirm that the Flask container has stopped and is no longer in that list.
* Now, we can try the same **docker run** command we used previously:

docker run --rm -d -p 5000:5000/tcp helloflask:latest

* It should show output similar to this:
* Check **docker ps** once again to confirm that the Flask container is back up. Notice that it will have a *different*Container ID.
* You can also check**Docker Desktop** and confirm that the container exists in the **Containers/Apps** panel, and that in the **Images**panel, the Flask image has an **IN USE** label next to it.
* Feel free to experiment with this container, and with the Flask app as well.
* For example, can you figure out how to change the "**Hello, World!**" message to something else?
* When you are done with it, stop the app with the **docker stop <ContainerID>**command, or through Docker Desktop, before you continue on.
* Don't remove your Hello Flask Docker image though! You will use it in a later exercise.