**Docker Images and Containers**

Completion requirements

**Done: View**

**7. Exercise: Containerizing Django App with Docker Compose**

**What You'll Do**

* In this exercise, we will learn the basics of using the Docker Compose tool.
* Docker Compose is typically used for multiple containers. Here, we will use it for a single container, for initial exposure to how it works. We will return to this subject in greater detail later.
* Many development technologies are complex, with many layers to them. Using them, gaining exposure and practice, and learning about it a portion at a time is generally more productive than trying to understand everything about such technologies all at once.

**About Docker Compose**

* Recall from the[Introduction to Docker](https://learn.nucamp.co/mod/book/view.php?id=5105&chapterid=5240)lesson that Docker Compose is a tool used for running multi-container Docker applications. ​
* Let's be more specific about what this means: Docker Compose allows you to run multiple Docker containers representing different parts of an application, e.g., the application code in one Docker container, and the database in a separate Docker container.
* To use Docker Compose, a file called**docker-compose.yml** is used to configure services that comprise these application parts.
* Then, we can use the **docker compose** CLI tool to create, start, and stop these services as separate Docker containers.

**Environment setup**

* Download this file:

[hello\_django\_docker\_compose.zip](https://learn.nucamp.co/pluginfile.php/88933/mod_book/chapter/5245/hello_django_docker_compose.zip)

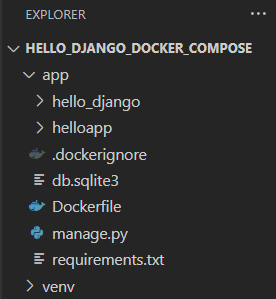
* Unzip this file. Once unzipped, you should have a folder named **hello\_django\_docker\_compose**.
* Depending on how you unzipped it, you may have a nested folder inside it, also named **hello\_django\_docker\_compose**. If so, this inner folder is the folder that you want to use for the next step, **not** the outer one.
* Take the **hello\_django\_docker\_compose**folder and place it in your **3-DevOps/week1/**folder.
* Open the **hello\_django\_docker\_compose/**folder in VS Code.
  + The contents of this folder will look nearly identical to the **hello\_docker\_django\_vscode/**folder you unzipped in the first exercise this week.
  + The main difference is that the outer folder is named **app**, and we have provided the initial **Dockerfile**and**.dockerignore** files for this exercise.
  + These files have been pre-configured for use with Docker Compose.
* Make sure that **Docker Desktop** is running before you continue.

**Confirm working application in virtual environment**

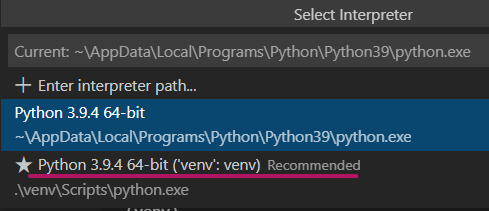
**Create virtual environment**

* We will once again confirm that this Django app works in a local environment, using venv, before attempting to containerize it with Docker.
* Open the integrated terminal in VS Code. This should open automatically to the **hello\_django\_docker\_compose/**folder.
* In this folder, create a virtual environment for testing:

python -m venv venv

* You should now see a **venv** folder in your **hello\_django\_docker\_compose/** folder:  
  

**Update the Python interpreter**

* Open the**Command Palette** with Ctrl/Cmd-Shift-P, then type/select **Python: Select Interpreter**.
* You should see an option listed with the**('venv': venv)** tag. Select this one.   
  
* Close the currently open terminal by clicking the trash can icon:  
  
* Then open a new integrated terminal and make sure that it automatically activates the virtual environment -- you should see **(venv)**in your command prompt.

**Install requirements and launch Django app**

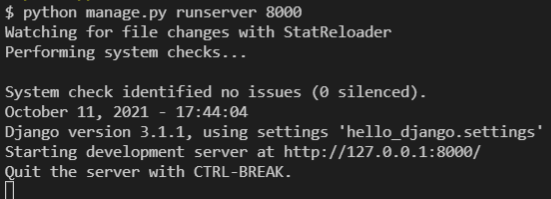
* In the VS Code integrated terminal (bash), navigate to the **app/** folder using the **cd** command.
* In the **app/** folder, update pip, then use it to install packages in the requirements file:

python -m pip install --upgrade pip==21.1.2

python -m pip install -r requirements.txt

* Enter the following command to run the Django app locally:

python manage.py runserver 8000

* You should see the following output:  
  
  + If you see an error message instead, double check to make sure that you are in the **app/** folder in your bash terminal.
* You should now be able to point your web browser to [http://127.0.0.1:8000](http://127.0.0.1:8000/) and see the following:  
  
* Back in your bash terminal where you started the server, use **ctrl-c** to close it.

**Create docker-compose.yml file**

* In the **hello\_django\_docker\_compose/**folder, add a new file named **docker-compose.yml.**
* **This file should *not*be in the app/ folder.**
* In this file, add the following content:

version: "3.8"

services:

  web:

    build: ./app

    command: python manage.py runserver 0.0.0.0:8000

    volumes:

      - ./app/:/usr/src/app/

    ports:

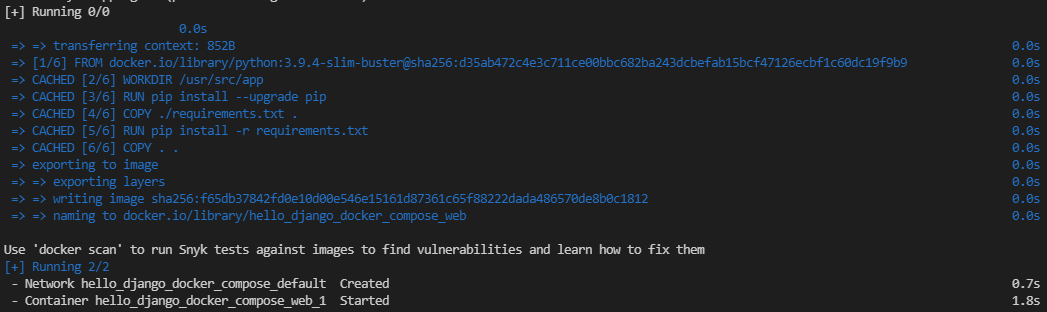
      - 8000:8000

* Save the file.
* Recall from the previous course that a file with a **.yml** extension is what's called a YAML file, a type of text file format commonly used for configuration files.
* The options here define a **web**service, with a Dockerfile for **build**ing an image located in the **app/** folder, the **command** for starting the web service, the **volumes** that will be used for storing persistent data, and the **ports** exposed for accessing web services. We will discuss the use of volumes and ports in more detail next week.
* Detailed information on the specifications for the**docker-compose.yml**configuration file can be found here:
  + <https://github.com/compose-spec/compose-spec/blob/master/spec.md>

**Build the Docker image and start/stop container from CLI**

* The next command must be entered in your bash terminal from the **hello\_django\_docker\_compose/**folder.
* If your bash terminal is open to the **app/** folder, enter **cd ..** in order to return to the **hello\_django\_docker\_compose/** folder.
* Enter this command to build the Docker image based on the **docker-compose.yml** file, and to run it as a detached (background) process:

docker compose up -d



* If this command does not work, confirm that you are executing the command from the **hello\_django\_docker\_compose/** folder and that the **docker-compose.yml** file exists in that folder.
* Note that if we had defined multiple services in the **docker-compose.yml**file, this command would build the images for *all*services defined, then create and start containers for *all*of them, not only one. You will see an example of this in this week's workshop. For this example, however, we are only using one image and one container.
* You can confirm that this container is running in several ways:
  + It should show up in both **docker ps** and **docker compose ps**.
    - The **docker ps** command will show any docker containers that are running on the system.
    - On the other hand, the **docker compose ps**command will only show containers created from images built from the **docker-compose.yml** file in the current directory, or any parent directory.
  + You should also be able to see both the image and container in VS Code's Docker panel, as well as in your Docker Desktop application.
* Feel free to experiment with this container, and with the Django app itself.
* For example, can you figure out how to change the "**Hello, World!**" text in the browser to another message? Hint: Look through the files in the **helloapp/** subfolder.

**Stopping and removing containers**

* When you are done, be sure to stop the container before you continue, using this command:

docker compose stop

* + Note that, unlike the **docker stop** command, no container ID is required as an argument for **docker compose stop**. It will stop all containers that were started with the **docker compose up** command.
  + Also, where **docker stop <containerID>**can be used from any directory, in order to use the **docker compose stop** command, you must be in the same directory as the **docker-compose.yml**file that was used to start the containers, or a child directory.
* You do not have to do this, but for reference, you can go one step further and remove the containers with the command **docker compose rm**. Composed containers must first be stopped before being able to be removed in this way.
* Another command, **docker compose down**, is a shortcut to both **docker compose stop** and **docker compose rm** in a single command.