**Docker Images and Containers**

Completion requirements

**Done: View**

**4. Exercise: Containerizing Django App with Docker**

**What You'll Do**

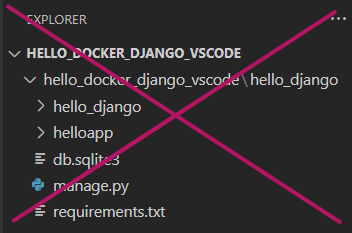
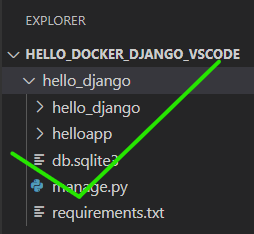
* In this exercise, we will walk through the steps of containerizing an application with Docker, using the Docker tools in VS Code.
* We will be using a basic Django "Hello World" application as the example app for running in a Docker container.
* The focus of this and the following exercises is to learn about using the Docker technology, rather than about Django or Flask. Docker is able to containerize many different kinds of applications. We will explore the Django framework in more depth later this week.

**Instructions**

**Environment setup**

* Download this file:

[hello\_docker\_django\_vscode.zip](https://learn.nucamp.co/pluginfile.php/88933/mod_book/chapter/5241/hello_docker_django_vscode.zip?time=1634104265032)

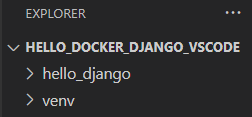
* Unzip this file. Once unzipped, you should have a folder named **hello\_docker\_django\_vscode/**.
* Depending on how you unzipped it, you may have a nested folder inside it, also named **hello\_docker\_django\_vscode/**. If so, this inner folder is the folder that you want to use for the next step, **not** the outer one.
* Take the **hello\_docker\_django\_vscode/** folder and place it in your **3-DevOps/week1/**folder.
* Open the **hello\_docker\_django\_vscode/**folder in VS Code. Make sure that you have viewed [this lesson on what it means to open a folder in VS Code](https://learn.nucamp.co/mod/page/view.php?id=5432).
* Inside the **hello\_docker\_django\_vscode/**folder should be a **hello\_django/** project folder, which contains the standard files and folders for an example project that has been generated by the Python framework called Django.
* There should not be two **hello\_docker\_django\_vscode/**folders, only one:  
   

**Confirm working application in virtual environment**

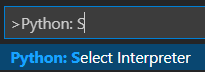
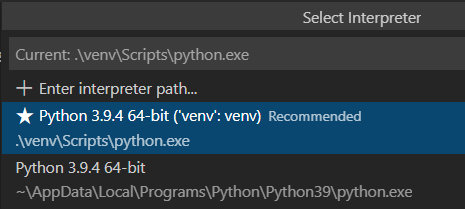
**Create virtual environment**

* Before using this application with Docker, we will first make sure that it works correctly *without*Docker.
* We will do this in a virtual environment, using **venv**.
* Open the integrated terminal in VS Code. This should open automatically in the **hello\_docker\_django\_vscode/** folder.
* In this folder, create a virtual environment for testing:

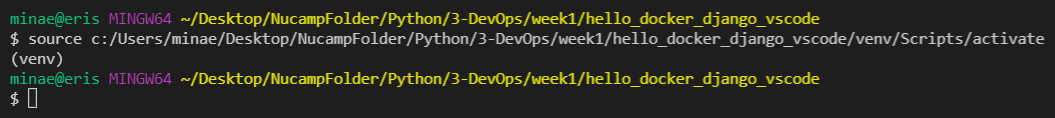
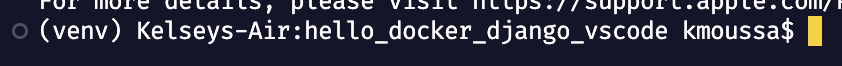
python -m venv venv

* You should now see a **venv/** folder inside your **hello\_docker\_django\_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 needed until you see that option appear below it:  
  
* Click on that option. You should next see a list of interpreters:  
  
* Select the one 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:  
  

**Install requirements and launch Django app**

* Open a new terminal. VS Code will automatically activate the virtual environment, and you should see**(venv)**in the command prompt:  
  
* MacOS:  
  
* Navigate to the **hello\_django/** folder using the **cd**command:

cd hello\_django

* A**requirements.txt**file has been provided in this folder along with the Django project.
* This file lists the Python packages required by this project, and we will use it to install those files.
* First, update the pip installation in your virtual environment:

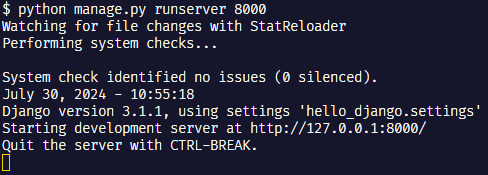
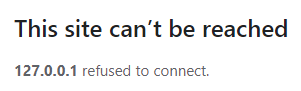
python -m pip install --upgrade pip==21.1.2

* + *\*Remember this command for the workshop*
* Next, install the packages in the **requirements.txt**file with the following command:

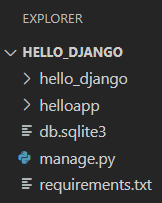
python -m pip install -r requirements.txt

* + \**Remember this command for the workshop*
* Then start the Django app by executing the code in **manage.py**:

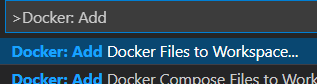
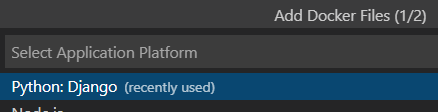
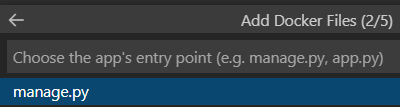
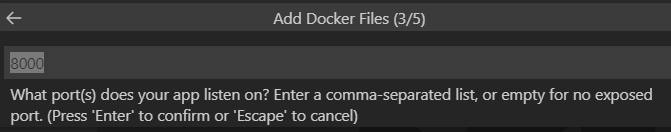
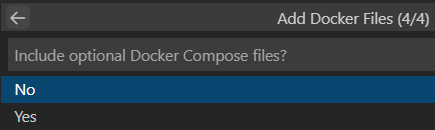
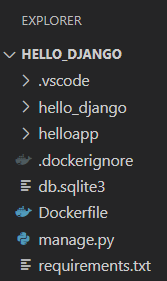
python manage.py runserver 8000

* This will launch the Django server locally on port 8000. You should see the following output in your terminal:  
  
* In a browser, open the Django app by going to this URL: [http://127.0.0.1:8000](http://127.0.0.1:8000/)
* You should see:  
  
* This confirms that the application is working in a local environment.
* Now, back in your bash terminal, shut down the Django app with the keyboard command **ctrl-c**.
* Your command prompt should appear again as the last line in your terminal.
* Return to the web browser and refresh the Django app webpage.
* You should no longer be able to connect, and an error message similar to this should be shown:  
  
* We can now proceed with creating a Docker image.

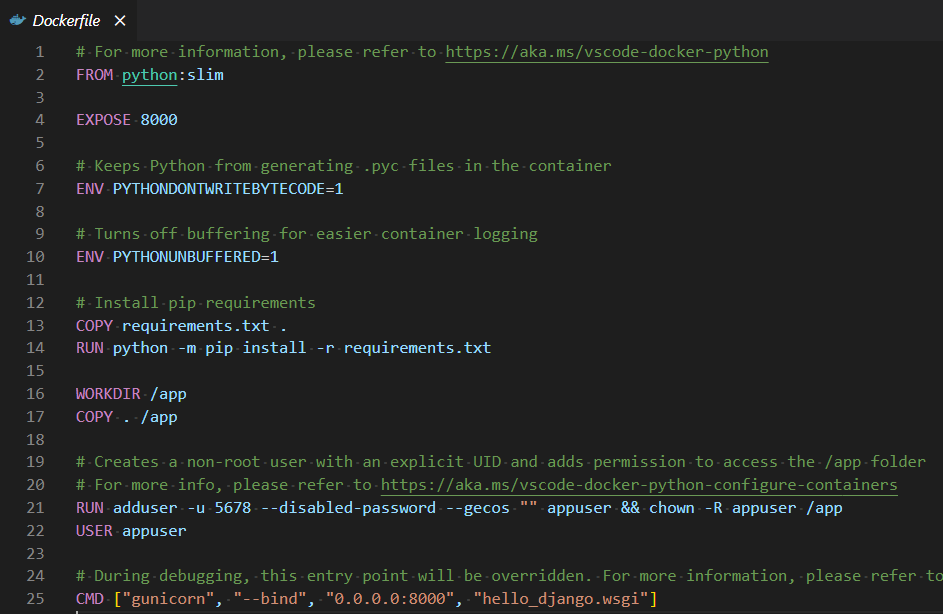
**Dockerize the Django web application**

* Dockerizing is an informal term that refers to building a Docker image of an application, then creating Docker containers from that image.
* There are multiple ways to do this.
* In this exercise, we will first look at how the Docker extension in VS Code can help us with the Dockerization process.
* In the following exercises, you will learn how to use the Docker CLI (command line interface) instead.
* First, we need to have VS Code open to the project's root folder, **hello\_django**.
* Open the **hello\_django/** folder in VS Code.
* This means you should go to **File > Open Folder**(Windows) or **File > Open** (macOS), select the **hello\_django/** folder (inside the **hello\_docker\_django\_vscode/** folder) by clicking once on it to highlight it, then click **Select Folder**.
* Once you have done this, your VS Code Explorer should look like this:  
  
* Make sure that you do *not*see the **hello\_docker\_django\_vscode/**folder in the VS Code Explorer.
* The very top-level folder there *must*be the **hello\_django/** folder as shown above. This is necessary so that VS Code creates the Docker files in the correct place.
* Next, we will generate the **Dockerfile**, a text file that contains commands Docker will use to build an image of our app.

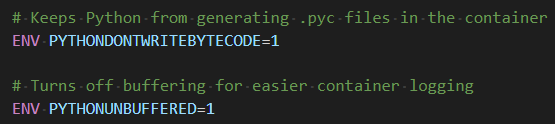
**Generate Dockerfile**

* The VS Code Docker extension can generate a basic **Dockerfile**for many different kinds of applications, including a Django application.
* Open the VS Code Command Palette (Ctrl/Cmd-Shift-P, or go to the View menu and choose Command Palette) and type in/select **Docker: Add Docker Files to Workspace…**:  
  
* You will then be asked to **Select Application Platform**. Choose **Python: Django**:  
  
* You will then be asked to **Choose the app's entry point (e.g. manage.py, app.py)**:  
  
  + This file is the point at which your application begins.
  + For Django apps, this is typically the **manage.py** file at the project's root folder.
  + Select this option. If you don't see it, scroll down the list of options, or simply type it in.
* Next, you will be asked what port(s) your app listens on.
* This will be automatically populated with the value **8000**, and all you need to do is press **Enter:**  
  ****
* 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**.
  + The **Dockerfile**contains instructions for building a Docker image.
  + The **.dockerignore**file, similar to the .gitignore file in Git, tells Docker what files and folders to ignore when building the image.
* It will also have created a **.vscode/** folder with some data used by the VS Code Docker extension:  
  

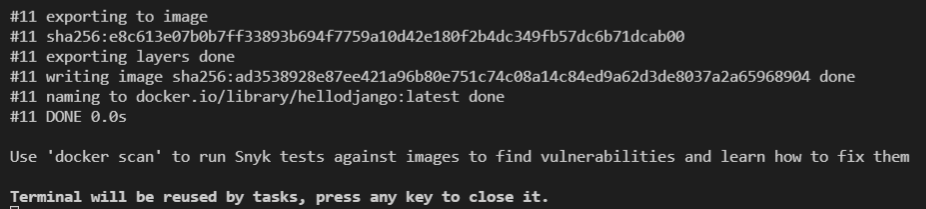
**Confirm Dockerfile configuration for Gunicorn**

* Open the generated **Dockerfile**. It should look like this:  
  
* The way that you have started the Django application thus far, using **python manage.py runserver** command, is only useful in development. It's not meant for anything other than testing.
* To run the Django application in a more production-ready way, a popular Python package called **Gunicorn**(Green Unicorn) is typically used as the web server that routes HTTP requests and responses for the Django app.
* The Gunicorn server is what's known as a **WSGI**server. WSGI stands for Web Server Gateway Interface, and you can find detailed information about this specification here: <https://wsgi.readthedocs.io/en/latest/learn.html>
* To use this server, we must provide certain arguments, including a host IP and port to bind, and a path to a configuration file.
* In our case, the Gunicorn configuration file already exists inside the **hello\_django/** folder and is named **wsgi.py**.
* The final line of the **Dockerfile**should already have this configured:

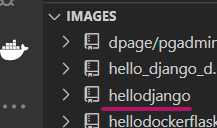
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "hello\_django.wsgi"]

* + Note that the final argument is not pointing to a *file* named "**hello\_django.wsgi"**! It is pointing to the **hello\_django/**folder, then to the **wsgi.py** file there.
  + If this line does not have "**hello\_django.wsgi**" as its path, update it, then save the changes.
* We will not need to update any of the other instructions in this Do**c**kerfile, but let's take a look at the contents.
* We'll also revisit the **CMD**instruction and what it means generally.
* The **Dockerfile**is made up of a series of instructions on how to build the image.   
  
  + The [**FROM**](https://docs.docker.com/engine/reference/builder/#from)instruction must be the first instruction in any **Dockerfile**. It sets a base Docker image from which to create the new image.
  + In this case, we are using a publicly available image called **python:slim**. This image will be pulled from a public Docker repository on the Docker Hub registry, which you will learn more about next week.
    - **The packages we are using for these exercises require python 3.12, make sure to change this line to** 
  + This image will set up a Python environment for us, which is required to run a Django application.
  + If we wanted to create an image for a Node.js application, we would use a Node image as a base image instead in the **FROM**command. 
  + The [**EXPOSE**](https://docs.docker.com/engine/reference/builder/#expose)instruction acts as documentation for the port that will be used by the Docker container.
  + In this case, the Docker container will need to use port 8000, because this is the default port for Django. This can be changed if needed.
  + You will learn more about managing ports in Docker next week as well.   
    
  + These are standard settings used by Docker. You do not need to be concerned with these, as they will rarely need to be changed.  
    
  + The Docker image has a separate filesystem from your computers, with its own set of folders and files.
  + The [**COPY**](https://docs.docker.com/engine/reference/builder/#copy)instruction  copies files from a source into the Docker image's filesystem.
    - In this case, the **requirements.txt** file that contains this application's Python package dependencies is being copied into the home folder of the image.
  + The [**RUN**](https://docs.docker.com/engine/reference/builder/#run)instruction will run a command in the Docker image, which automatically creates a new image layer.
  + In this case, the **RUN**instruction is being used to install the required packages from the **requirements.txt**file.  
    
  + The [**WORKDIR**](https://docs.docker.com/engine/reference/builder/#workdir)instruction sets the working directory within the image's filesystem for instructions that follow it, meaning those instructions will be executed from in that directory/folder.
  + This is similar to using a **cd** command in your terminal to change the working directory in your terminal.
  + If the provided working directory/folder (in this case, **app/**) does not already exist, Docker will create it.
  + You can use **WORKDIR**multiple times within a **Dockerfile**.
  + The [**COPY**](https://docs.docker.com/engine/reference/builder/#copy)instruction takes two arguments: a source to copy from, and a destination to copy to.
  + In this case, the dot after **COPY**stands for the current folder in your computer's filesystem where the **Dockerfile**exists.
  + The **Dockerfile**should exist in the **hello\_django/** folder, so that means it will copy everything *from* inside the **hello\_django/** folder.
  + Then it will copy all those files *to* the **/app** folder in the image. This is a crucial step!   
    
  + This **RUN**instruction creates a new user in the Docker image environment.
  + The [**USER**](https://docs.docker.com/engine/reference/builder/#user)instruction sets that newly created user as the one to use when the image is run.
* Finally, let's revisit the [**CMD**](https://docs.docker.com/engine/reference/builder/#cmd)instruction at the bottom of the file:  
  
  + As previously mentioned, this particular instruction is used to set up the Gunicorn web server for the Django app to use.
  + The **CMD** instruction in general is used to specify a command to run when an image is actually run, along with any arguments for it, provided in the form of a JSON array.
  + Unlike other instructions, this is not an instruction that occurs when the image is being built, but saved for when the image is run after being built.
  + In this example, the Gunicorn server is started when the image is run.
  + There can be only one **CMD** instruction for an image. If more than one is defined in the image's **Dockerfile**, only the final one will be used.
* You can find out more about **Dockerfile**instructions here:[https://docs.docker.com/engine/reference/builder/](https://docs.docker.com/engine/reference/builder/#user)
* Close the **Dockerfile**.

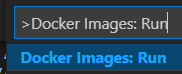
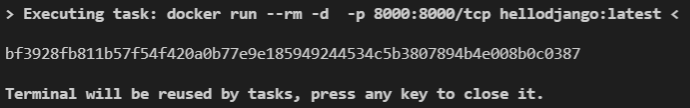
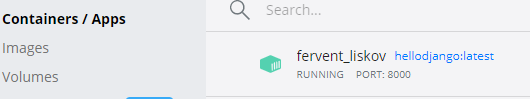
**Build Docker image**

* We will now build the Docker image for this application, using the VS Code integrated tools for Docker.
* Start up Docker Desktop if you don't have it running already.
* Make sure that VS Code is open to the **hello\_django/**folder, the one that has the **Dockerfile**in it.
* 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.

**Confirm Docker image creation**

* At this point, go to the **Docker** panel of VS Code by clicking on the Docker icon (the Docker whale ) on the left side of VS Code.
* You should now be able to see the new image in Docker Explorer, under **IMAGES**:  
  
  + The image name is automatically generated from the project name.
* You can also verify that the image has been built in the **Docker Desktop** app, in the **Images** panel.

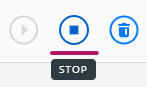
**Run container from Docker image**

* Open the Command Palette (Ctrl/Cmd-Shift-P) and type in/select **Docker Images: Run**:  
  
* You will be asked to **Select image group**. Select **hellodjango**.
* You will then be asked to **Select image**. Select **latest**.
* Your terminal should then show this output:  
  
* Confirm that the image is running by going to Docker Desktop, and click Containers/Apps on the left side menu.
* You should see the image running there at port 8000. The name will be randomized:  
  
* Go back to your web browser and confirm that the Django web app can now be viewed at [http://localhost:8000](http://localhost:8000/).
* Congratulations! You have just set up a Docker container for a Django app.

**So what? What does that even mean?**

* Let's reflect for a moment on what this means.
* Near the beginning of this exercise, you ran your Django app outside of a Docker container first, just to make sure that it works on its own.
* To do this, you had to use **pip** to install the required packages first, then run the app directly using Python.
* What if someone else wanted to run your app on their own computer, without Docker?
* They would need to get the code from you, then they would need to install Python, and install all the packages before they could run your app.
* If they had Docker installed, they could simply download the image of your app, then run your app without needing to install Python or any packages.
* In your SQL course, you similarly were able to run Postgres and pgAdmin on your computer by using Docker containers, without needing to actually install Postgres or pgAdmin directly to your computer and configure it to be ready for use.
* This avoided many potential configuration issues and bugs that can occur when installing an application directly to your system.

**Stop the hellodjango container**

* Back in Docker Desktop, **stop**the app from the Containers/Apps panel before you continue on to the next exercise.  
  
* In the next exercise, we'll look at using Docker CLI commands to interact with the Docker image and container we just created.