Creating and Using Docker Dev-Containers for Python Jupyter Notebooks in VS Code

Example GitHub Repository

Docker containers provide a great way to eliminate dependencies on the hosts software configuration. This ensures all your code runs the intended way and adds great value in giving ability to run programs that would otherwise not be configured for your devise.

In my example I was not able to install Esri's arcgis python module since I use apple silicon to compute, and it is unavailable for this configuration at this time. Docker proved a great workaround allowing me to effectively run Linux (a supported OS) within the container.

Running a Pre-Configured Image

Many pre-configured docker images are available on Docker Hub including the one for my example: https://hub.docker.com/repository/docker/ejturkon/mtc_geo_image/general

For context let's run through getting the image from my example up and running in a notebook in VS Code.

Following anywhere there is formatting with <some_text>, this represents a part of the example where you should replace this for your specific use case.

Dependencies for Image Usage in VS Code:

- VS Code
- Python 3.7 or newer
- Docker Desktop (ensure its up to date)

Ensure you are in the correct working directory and pull the Docker image.

Pull the Docker image.

```
$ docker pull <ejturkon/mtc_geo_image>
```

Ensure Docker cli recognizes the image. If you see the name of your image you are all set, i.e"ejturkon/mtc_geo_image".

```
$ docker images
```

Build the container. This line assumes all of your ports are still in the default config, port 8888 is the default.

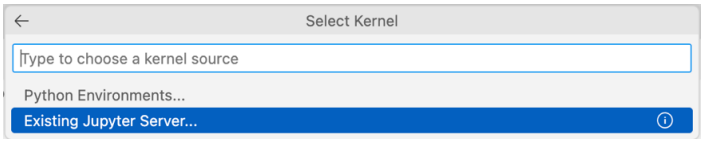
```
$ docker run –p 8888:8888 ––name <geo_container> <ejturkon/mtc_geo_image>
```

Once built you should see an output with "To access the server, open this file in a browser:" and it should then include 3 URLs after. Copy and store the last URL that looks most like the one seen below.

```
http://127.0.0.1:8888/lab?token=cf66495e0d3f4396392e24ea48545772d861442dfd9
f8b1f
```

Restart VS Code and open the .ipynb file of your choice. Click the Select Kernel button, or from the command palate "Notebook: Select Notebook Kernel"

Choose "Existing Jupyter Server". Paste the URL saved from above and hit enter twice.



Click Python 3 (ipykernel) and you should be good to go!

The following section is image specific and is not required by any means. Because docker runs within a container you're not able to access any local files. The example image we are walking through includes dvutils updated as of 01/10/24, I wanted to include some documentation on how to use the i/o functions for anyone this may apply to.

Using dvutils i/o Functionality Within the Container

```
import os
import sys
import json
import arcgis
import geopandas as gpd
DVUTILS_LOCAL_CLONE_PATH = '/usr/src/app/dvutils'
sys.path.insert(0, DVUTILS_LOCAL_CLONE_PATH)
from utils_io import *
```

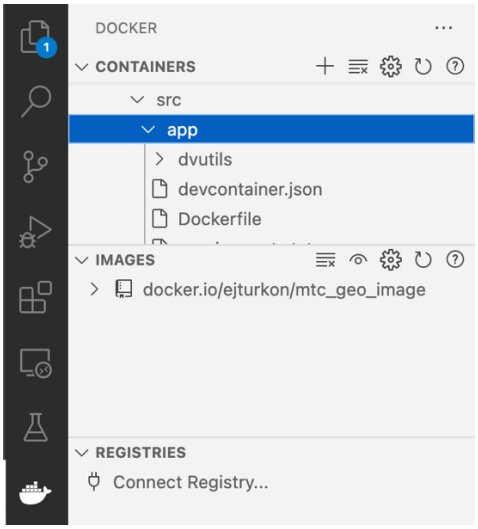
✓ 1.3s

```
Info: No local Box directory found
Info: User not found in teams.py
Info: No credentials found. Utility code functionality will be limited.
```

Assuming you have the Docker extension for VS Code you can view and edit the dvutils files as they are stored in "/usr/src/app". Here you can hard code your creds, just change your username key to 'root' seen below.

```
def get_socrata_creds():
    """Loads the user's Socrata creds.

    Returns:
        dict: A dictionary of the user's Socrata creds
    """
    s_creds = {'domain': 'data.bayareametro.gov',
               'root': {'username': 'ejturkon@dons.usfca.edu',
                        'password': ''
                        'app_token': ''}
                        }
    return s_creds
```



Creating a Container for Python

To create a docker container you will need 3 components a Dockerfile, image and container these components must be created in that order since each one is a dependency for the next.

Dependencies for Image Creation

- Python 3.7 or newer
- Docker Desktop (ensure its up to date)
- Ensure you are in the correct root folder where the project is contained

Dockerfile

A Dockerfile is a text document containing all the command line prompts necessary to assemble an image. For simplicity of the Dockerfile lets create a reqirements.txt file specifying the modules the code being run is dependent on.

```
$ touch requirements.txt
```

The format within this file should be <module>==<version> my example below.

```
numpy==1.26.3
pandas==2.1.4
arcgis==2.2.0
geopandas==0.14.2
```

Back in the command line, lets create and open the Dockerfile. I will walk through the essential parts of creating a Dockerfile, the full example Dockerfile can be seen here.

```
$ touch Dockerfile
$ vi Dockerfile
```

In my example I wanted to use python 3.11 and as mentioned above required a Linux OS. This call specifies the tag base image being used. The -slim-buster indicates a Debian Buster Linux distribution. Within the Dockerfile use FROM to specify what you wish to run the code on.

```
FROM <python:3.11-slim-buster>
```

To install dependencies without interaction we use.

```
ENV DEBIAN_FRONTEND=noninteractive
```

Install system dependencies.

```
RUN apt-get update && apt-get install -y \
    …
    && rm -rf /var/lib/apt/lists/*
```

We then create and activate a virtual environment within the container. Using a virtual environment removes the possibility of collisions with host OS modules and makes the environment easier to control.

```
RUN python -m venv .venv
ENV VIRTUAL_ENV=/app/.venv
```

```
ENV PATH="$VIRTUAL_ENV/bin:$PATH"
```

Copy your current directory into image. Here is where you can include dvutils or any other files you wish to use within your docker container.

```
COPY . /usr/src/app
WORKDIR /us/src/app
```

Install the requirements in requirements.txt.

```
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt
```

Expose the port so that you can access the container while its running (8888 is default).

```
EXPOSE 8888
```

Change the working directory to within the container.

```
WORKDIR /usr/src/app
```