Joseph Stewart
jstew7@illinois.edu
CS 410 Text Information Systems
Course Project Documentation

Purpose

The purpose of this project was to develop a way to run Python applications which make use of the MeTA Toolkit metapy Python bindings in a consistent environment regardless of the host operating system and installed versions of various tools.

An overview presentation can be found at the following link:
https://drive.google.com/file/d/1Ru5L1xznvhgG1XZzRRiSrkfNPk_kuoRK/view

Motivation

Many students enroll in Text Information Systems with a great deal of knowledge and experience and have very few problems utilizing the required tools for the course. Some, however, have very limited experience and, for a variety of reasons, far less time than others to spend on troubleshooting configuration issues in addition to the time it takes to complete the work.

Also, some students may be in a situation where they do not have a dedicated, quality development environment in which they are free to configure how they see fit. Additionally, some may simply want a one-and-done solution to a smaller assignment or short-term project where they don't want to keep files and configurations around once completed. This project attempts to address these by providing a way to run small Python applications that make use of the metapy library without having to alter the configuration of their host environment in a consistent and configurable temporary environment.

Dependencies

This project requires that Docker Desktop be installed and running before getting started.

Docker Desktop can be downloaded free of charge from the link below:
https://www.docker.com/products/docker-desktop

The system also needs to have Python installed so that Python source files can be run via the command line. Python 3 is preferred so that an optional feature can be used, but it will work with Python 2.

Python can be downloaded free of charge from the link below:
https://www.python.org/downloads/

Finally, the source file for this project needs to be downloaded before use. It can be found and downloaded free of charge from the link below:
https://github.com/jstew7-410/CourseProject

Implementation

The implementation of my project is fairly simple after getting past the learning curve involved with Docker. Essentially a Python application will be run in a Docker container which contains all of the dependencies needed for that specific application.

The project involves a single Python script called docker_driver.py which can be invoked from the command line on the host machine. The script first reads in the command line arguments and validates the input. There are 4 command line arguments with two being required and two being optional.

The required arguments are the -s(--src) flag which has the path to the directory containing the source code that is to be run in the Docker container and the -f(--file) flag which has the name of the Python file to be run in the Docker container – this Python file needs to be in the -s source directory.

An optional -a(--args) flag is used to specify the command line arguments that should be used when running the main Python file that was passed in with the -f flag.

The other optional flag is the -r flag which can be used to automatically create the requirements.txt file used for building the Docker image. Python 3 must be installed and used to run the docker_driver.py script with the -r option because this functionality relies on the stdlib_list module which is only available for Python 3.

When the -r option is used, the script deletes the requirements.txt file if it already exists and creates a new one. It then reads in the Python file passed in with the -f (--file) argument and parses it for the modules on which the script is dependent. If a module is not part of the standard Python 3 libraries, then it is added to the requirements.txt file.

If the -r option is not used, the script checks for an existing requirements.txt file and prints and error and exits if one is not found.

The script then creates the Dockerfile used to build the Docker image; it will delete this file if it already exists and create a new one like the requirements.txt file. This file contains all of the instructions needed to build the image.

The script then has all of the information it needs so it then builds the docker image with a generic image name.

The application is then run in a Docker container. The Docker container and image are removed from the system after the application finishes leaving behind as small a footprint as possible.

Usage

There are a couple of dependencies as mentioned above which are required in order to run the script. Docker Desktop must be installed and running. The system must be capable of running Python applications from the command line.

The project directory structure should be set up as follows:

projectDirectory
- app src directory
  - Python file to run in Docker container
- docker_driver.py

Here is an example use case for running our MP2.4 via this project's script:

project/
- MP2.4/
  - search_eval.py
  - etc
- docker_driver.py

I modified search_eval.py to require that *-f config.toml* be used to specify the configuration file so that the -a optional argument can be demonstrated.

To run search_eval.py on the command line, for example, one would enter:
python3 search_eval.py -f config.toml

Running MP2.4 using this project would require the following steps:

- Open a terminal and change directory to project/
- Enter the following command:
  - *python3 docker_driver.py -s MP2.4 -f search_eval.py -a " -f" config.toml -r*
    - Notice that the -f argument is in double quotes with a leading space
    - This is required because Python's argParse reads the variable number of arguments until it reaches the next flag
    - The quotes are needed due t the space and the space is needed to avoid -f as being interpreted as a separate argument to the docker_driver.py script when it is actually part of the -a argument since it is to be passed in to the search_eval.py process
  - For Python 2, run: *python docker_driver.py -s MP2.4 -f search_eval.py -a " -f" config.toml*

This will perform the following:
- Invoke the docker_driver.py script
- Parse the command line arguments
- Parse the search_eval.py file for module dependencies
- Build the Dockerfile and requirements.txt files
- Build the Docker image
- Run the search_eval.py file in a Docker container and pass in *-f config.toml* as an argument

The project directory will look like the following after the application finishes:

project/

- MP2.4/
  - search_eval.py
  - etc.
- docker_driver.py
- Dockerfile
- requirements.txt

Potential Improvements

There is much room for improvement for this project. Some ideas include:

- Capturing the application's output to a file may be useful
- Checking if Docker is running and starting it if it is not
- Differentiating between user-defined modules and those to be installed in the image
- Automatically copying and deleting project files to root directory for convenience and to avoid having multiple copies in use
- Having the Docker container continue running at a terminal when the Python application exits