

How to run Python scripts using Docker

Running Python scripts is one of the most common tasks in automation. However, managing dependencies across multiple systems can be a challenge. That's where Docker comes in.

Running a Python script is one of the most common tasks in automation. However, managing dependencies across different systems can be a challenge. That's where Docker comes in. Docker allows you to package a Python script along with all the necessary dependencies into a container, ensuring it runs the same way on every machine. Today's article will walk you through the process of creating an actual Python script and running it inside a Docker container.

Write Python script

Create a project directory to store your Python script and Dockerfile. Once created, navigate to this directory using the `cd` command :

```
mkdir docker_file_organizer cd docker_file_organizer
```

Create a script called 'organize_files.py' to scan a directory and group files into directories based on their extensions:

```
nano organize_files.py
```

Paste the following code into the 'organize_file.py' file. Here, we use two built-in Python modules, named **os** and **shutil** , to handle files and create directories dynamically:

```
import os import shutil SOURCE_DIR = "/files" def organize_by_extension(directory, ext): try: os.mkdir(directory + {ext}/" ) except Exception as e: print(f"Error organizing files: {e}") if __name__ == "__main__":
```

In this script, we sort the files in a given directory based on their extensions. We use the **os** module to list the files, check if each entry is a file, extract the file extensions, and create directories named after those extensions (if they don't already exist). Then, we use the **shutil** module to move each file into its corresponding directory. For each move, a message showing the new location of the file is output.

Create Dockerfile

Now, let's create a Dockerfile to define the environment in which the script will run:

```
FROM python:latest LABEL maintainer="you@example.com" WORKDIR /usr/src/app COPY
```

We use this Dockerfile to create a container in Python , add a script to it, and make sure the script runs automatically when the container starts:

```
GNU nano 7.2 Dockerfile *
FROM python:latest
LABEL maintainer="anees@mte.com"

WORKDIR /usr/src/app
COPY organize_files.py .

CMD ["python", "./organize_files.py"]
```

Build Docker image

Before you can build a Docker image, you first need to install Docker. Then, run the following command to package everything into a Docker image:

```
sudo docker build -t file-organizer .
```

It reads the Dockerfile, combines the Python setup and scripts so they are ready to run in a single container image:

```
anees@linuxuser:~/docker_file_organizer$ sudo docker build -t file-organizer .
[+] Building 174.6s (4/7) docker:default
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 177B 0.0s
=> [internal] load metadata for docker.io/library/python:latest 3.8s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/3] FROM docker.io/library/python:latest@sha256:a6af772cf98267c48c145928c 170.7s
=> => resolve docker.io/library/python:latest@sha256:a6af772cf98267c48c145928cbe 0.1s
=> => sha256:a6af772cf98267c48c145928cbeb35bd8e89b610acd70f93e3e 9.72kB / 9.72kB 0.0s
=> => sha256:1c1dd43eba265a3a394d7e086c38e8d5178da79845a2b359c28 6.32kB / 6.32kB 0.0s
=> => sha256:c1995213564325caf7e52ecd95fe4435c70b03eb94c674ac 48.49MB / 48.49MB 82.6s
=> => sha256:238379aacf40f83bfec1aa261924a463a91564b85fbbb97c9a9 2.32kB / 2.32kB 0.0s
=> => sha256:7bbf972c6c2f5b7313ae3cb74e63888ab70931bcd9aefd96 24.02MB / 24.02MB 75.3s
=> => sha256:900e2c02f17f686733f4f957ddfb07b3342d1957d87b562 64.40MB / 64.40MB 141.4s
=> => sha256:abe9c1abe6f3b8ca9fc6abe710405f830f95262f1d356e 63.96MB / 211.37MB 170.6s
=> => sha256:562e9f67c041256c29786a8c683feb6476a163a988ae50af68 6.16MB / 6.16MB 92.2s
=> => extracting sha256:c1995213564325caf7e52ecd95fe4435c70b03eb94c674ac15706733 8.0s
[+] Building 174.8s (4/7) docker:default
=> [internal] load build definition from Dockerfile 0.1s
```

Create a sample folder with the files

To see the script in action, create a test directory called 'sample_files' with a few files of different types. Create these files just to clutter the directory a bit and see how the Python script handles it:

```
mkdir ~/sample_files touch ~/sample_files/test.txt touch ~/sample_files/image.jpg
```

Run scripts inside Docker

Finally, run the Docker container and mount the sample directory into it. The `-v` flag mounts the local `~/sample_files` directory into the `/files` directory in the container, allowing the Python script to read and sort the files on your host:

```
docker run --rm -v ~/sample_files:/files file-organizer
```

Here, the article uses the `--rm` option to automatically delete the container after it finishes running, helping to save disk space:

```
anees@linuxuser:~$ sudo docker run --rm -v ~/sample_files:/files file-organizer
Moved: image.jpg -> jpg/
Moved: test.txt -> txt/
Moved: data.csv -> csv/
anees@linuxuser:~$
```

Finally, use the `tree` command to check if the files are sorted into directories based on their extensions:

```
tree sample_files
```

```
anees@linuxuser:~$ tree sample_files
sample_files
├── csv
│   └── data.csv
├── jpg
│   └── image.jpg
└── txt
    └── test.txt

4 directories, 3 files
anees@linuxuser:~$
```

Note : The `tree` command does not come pre-installed on most systems. You can easily install it using a package manager like `apt` on Ubuntu , `brew` on macOS, etc.

You finished reading the article "**How to run Python scripts using Docker**" edited by the [TipsMake](#) team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.