ARCathon

Python Docker Tutorial

About the Tutorial

The following will describe how you create a simple Dockerfile that allows you to create a Docker image containing your python code. Generally, a Dockerfile is a set of instructions that specifies the environment in which a code should run and the needed commands.

Disclaimer

This is a basic tutorial on the use of Docker. The purpose of this tutorial is to provide a practical guide to the rules for submitting an ARCathon solution and to provide an entry point for those who have never used Docker before.

Our Python script

To explain the submission rules, we will "dockerize" the Python script which you can find in the same zip folder of this PDF (named: tutorial_script.py). The script reads the evaluation data set in the data folder. It makes a meaningless prediction about the output by taking random grids and filling them with the color of the test input with the highest numeric value. Its goal is to explain how a submission should run in the sense that:

- It takes the evaluation data from outside the container as input.
- It outputs text to indicate progression.
- It outputs the solutions to a JSON file with a structure described in the submission rules.

In the end, it outputs text which indicates it has finished.

Installing Docker

If you have not yet installed Docker, you can download it here. You will need it for building and running the Docker image.

Creating a Dockerfile

As a first step, you start with an empty directory (preferably not your root directory) which you can name docker_tutorial. In the end, this directory should contain everything needed to build the Docker image. In the case at hand, put the script tutorial_script.py in the folder tutorial_code/ and the folder containing the publicly available tasks in secret_data/. You could also choose different names for these folders. However, when you submit your solution, we will put our secret task set in the folder secret_data/evaluation/, so you can test how this works. We will explain to you how you include this data in your image and make it available to your code below when we discuss how to execute an image.

Now open any text editor at hand and create within the directory a new file called Dockerfile without extension (like .txt). In the end, this file contains all specifications to build the Docker image later. As a launching point, use the official Python 3 image provided by the Docker community and choose the basic one with Python version 3.10.6 tagged 3-slim (all Python 3 versions work for this). For this, add the following line to your Dockerfile:

```
FROM python:3-slim
```

The goal is to run the python script called tutorial_script.py. For this, the script needs to be added to the Dockerfile. Above, you put the script in the folder tutorial_code/, and you can include it by adding the following line:

```
ADD tutorial_code/tutorial_script.py /
```

Suppose that the script needs some libraries. In our example, we need the library requires *NumPy*. All dependencies need to be installed before running the image. To install *NumPy,* add the following line to your Dockerfile:

```
RUN pip install numpy
```

Now you are ready to include the command to run the script in your Dockerfile. This can be achieved by adding the following line:

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

Everything put together; your Dockerfile should look as follows (note that you can add comments by starting your line with a #):

```
# Use basic Python 3 image as launching point
FROM python:3-slim
# Add script
ADD tutorial_code/tutorial_script.py /
# Install dependencies
RUN pip install numpy
# Execute the script
CMD ["python", "./tutorial_script.py"]
```

Summary:

- FROM specifies which image you want to base your image on (e.g., Ubuntu or here Python 3).
- ADD is used to include files like code you want to execute or anything else that should be available when running the image.
- RUN specifies which commands should be executed when building your image.
- CMD will execute the command specified in the brackets when the image is run.

Building an image

You are now ready to build an image from your Dockerfile. For this, open a terminal inside your docker_tutorial folder and run the following command:

```
docker build -t docker_python_tutorial .
```

The dot indicates that Docker should look for your file called Dockerfile in the current directory. The -t is used to name your image (here, docker_python_tutorial).

Running an image

Now you are ready to run the previously built docker image as a container. You can first run the command docker images in your terminal. If the previous build was successful, you should see an entry "docker_python_tutorial".

As explained in the "Submission Rules & Instructions", we want to include the evaluation tasks outside our container. Here we will take the publicly available ones, which you can download directly with this link: ARC (800 tasks). Then put the data folder into the directory where we have our Dockerfile and rename the data folder to secret_data/ (that's where we will put the secret set in the folder evaluation/ after you submitted your solution).

Now you are ready to run our newly created Docker image by passing it the path to our task data:

```
docker run --mount type=bind,source="$(pwd)"/secret_data,target=/data
docker_python_tutorial
```

This command (or a similar one) is what needs to be provided for submission together with your image. Using the mount command, you included the folder containing the tasks downloaded before in your docker container, located in your current directory in the data folder. Finally, you use the name of the previously created image to tell which image you want to use.

The terminal output should now look something like this:

```
(base) ➜  Docker_python_tutorial docker run --mount
type=bind,source="$(pwd)"/secret_data,target=/data docker_python_tutorial
Generated solution for 50 of 400 test examples
Generated solution for 100 of 400 test examples
Generated solution for 150 of 400 test examples
Generated solution for 200 of 400 test examples
Generated solution for 250 of 400 test examples
Generated solution for 300 of 400 test examples
Generated solution for 350 of 400 test examples
Generated solution for 400 of 400 test examples
Program has finished!
```

And you should find a file solution_lab42.json in the secret_data/solution/ folder.

Further questions

Please refer to arcathon@lab42.global in case of any technical or other questions on ARCathon.