Docker Programming

EECS 348 Lab 11 — 4/17/2025

Harlan Williams

Electrical Engineering and Computer Science
University of Kansas

Docker

Docker provides lightweight virtual machines called *containers* that run your application in a consistent environment

- Runs an operating system with a specified version
- Very few other applications will be running at the same time in the container, ensuring efficiency

These containers are *instantiations* of what Docker calls *images*, templates for how to create a container

- Created using a script which makes instantiation very reproducible
- Can be based off of other images
- Can easily be shared or deployed to servers

Workflow

In an existing or new project, you create a file named `Dockerfile` and in it specify commands to set up images for your application

`docker build -t image-name .` will use the `Dockerfile` in the current directory to build an image in Docker

`docker run image-name` will instantiate a container from the image and run it

On the Docker hub website you can create a repository and run `docker tag image-name repository-name` then `docker push repository-name` to publish your image

Example web app

Below is a very basic ”Hello world” web application written using the Flask library for Python:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return "<html><body>Hello, world!</body></html>"
```

The app is instantiated in the second line, and then the function `index` defines what is returned when a user navigates to `"/"`

Example Dockerfile

Below is a Dockerfile that sets up a lightweight environment to run the web app from the previous slide:

```
FROM python:3.13-alpine3.21

RUN python3 -m pip install flask

WORKDIR /app
COPY hello.py .

ENV FLASK_APP=hello
EXPOSE 80

CMD ["flask", "run", "--host", "0.0.0.0", "--port", "80"]
```

Example Dockerfile *contd.*

The first line with the `FROM` keyword specifies what image to base our image off of. In this case, a lightweight Linux distro with Python 3.13 already installed

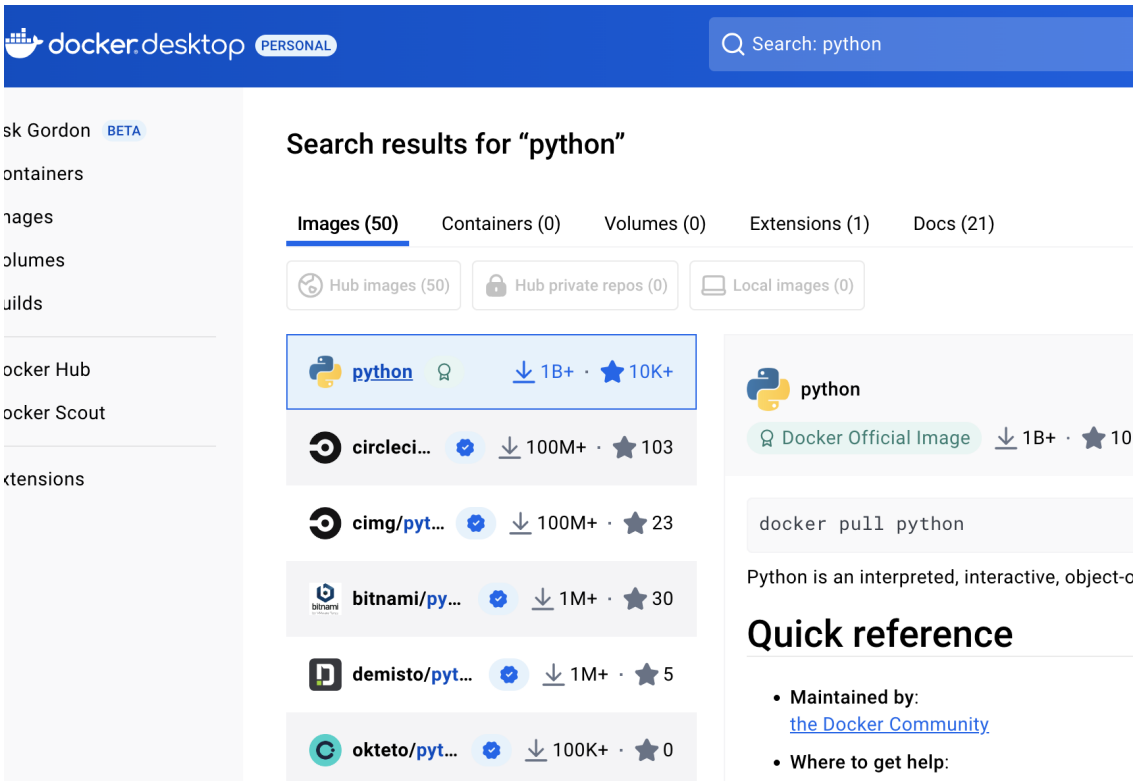
Next `WORKDIR` makes a new directory and `COPY` moves our Python source file into that directory (`.` is an abbreviation for current directory)

`ENV` sets an environment variable pointing `flask` to our source file and `EXPOSE 80` opens port 80 (the port HTML uses)

Lastly `CMD` runs our flask app whenever we create a new container from this image

Finding images

You can find images to use with the `FROM` keyword by searching in Docker desktop. If you can find an image that has most of what you need, it can save you a lot of time writing configuration files



Lost? See...

- https://docs.docker.com/build/concepts/dockerfile/
- Auth errors? Try running `docker logout` and then `docker login` again