Docker and Docker Compose

Teacher Version

Software Architecture

March 6, 2023

Evan Hughes & Brae Webb

Figure 1: Docker containers by MidJourney with Prompt ""

1 This Week

This week our goal is to:

- Learn docker basics.
- Dockerise our Todo application from last week.
- Move to a postgresql database from the SQLite database we used last week.
- Testing and Packaging our application on commit.

2 Getting Started with Docker

In the lectures we have delved into the world of containers and how they can be used to create a consistent environment for our applications. Depending on the courses you have taken you may have been exposed to them as either a user or a consumer, for instance if you have submitted assessment to gradescope in CSSE1001 or CSSE2002 the submission system uses docker to run your code.

Notice

This practical will assume you have not used docker before but we will be skipping over some aspects since its not needed for our purposes. Though since docker is a tool that many courses can use we have created a mini-course on it. You can find it here: `https://extend.uq.edu.au`. We recommend checking out the mini-course if you have any difficulties with the practical.

For the teacher

Sections 2.1 and 2.2 are presentation based where you should be covering the fundementals of docker images and containers. Where we are using container to describe the implemented image running in an environment.

2.1 Docker Images

Images are the building block of the principals on containerisation. They are a self contained environment that can be run that contains the depenedencies and code for an application. A common way to build an image is to use a Dockerfile. Though Docker was not the first to make this concept, it is very popular and relativly easy to use.

2.1.1 Dockerfile

The Dockerfile is a plain text file with its own markup language that is used to describe the image. It is a series of instructions that are run in order to build the image. The instructions are run in a layered fashion, each instruction is run on top of the previous instruction. This means that if you change a line in the Dockerfile you only need to rebuild the layers that have changed. This is a huge time saver when developing your application.

A reference of all possible Dockerfile instructions can be found here: `https://docs.docker.com/engine/reference/builder/`. For the purposes of the course we are just gonna cover some of the basics and some common recipes.

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
```

When building a Dockerfile you need to start with a base image. This is the image that you will build on top of. In this case we are using the latest version of ubuntu. This ubutu is a very minimal image that is used intended to be used as a starting point and is itself an image.

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest

# Installing dependencies for running a python application
RUN apt-get update && apt-get install -y python3 python3-pip
```

Now that we have a base image we can start to add our own dependencies. In this case we are installing python3 and pip. The RUN command is used to run a command in the container. Since we are using ubuntu we can use apt-get to install our dependencies and we are chaining the commands so that they are run as one block. You will notice that we are running `apt-get update` before we install our dependencies. This is because the base image probably has no package caches as that would take up unnesscary space. So we need to update the package cache before we can install anything.

TODO: continue fleshing this out

2.1.2 Dockerfile Layers

In the previous sections we have been using the RUN, ADD and COPY to build up our image to run an example application. When the container is built each of these instructions is run on a layer of the image. This means that if you change a line in the Dockerfile you only need to rebuild the layers that have changed. This is a huge time saver when developing your application.

Lets have a look at an example of the layers that are created when we build our Dockerfile from the previous section.

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest

# Installing dependencies for running a python application
RUN apt-get update && apt-get install -y python3 python3-pip

# Copying our application into the container
COPY . /app

# Installing our application dependencies
RUN pip3 install -r /app/requirements.txt

# Setting the working directory
WORKDIR /app

# Running our application
CMD ["python3", "src/app.py"]
```

We see that we are copying in the entire application directory before we install our requirements. We can improve this so that we only copy in the requirements file and then install the requirements. This means that if we change our application code we don't need to reinstall the requirements.

```dockerfile
# syntax=docker/dockerfile:1
FROM ubuntu:latest

# Installing dependencies for running a python application
RUN apt-get update && apt-get install -y python3 python3-pip

# Copying our application into the container
COPY requirements.txt /app/requirements.txt

# Installing our application dependencies
RUN pip3 install -r /app/requirements.txt

# Copying our application into the container
COPY src/ /app

# Setting the working directory
WORKDIR /app

# Running our application
CMD ["python3", "src/app.py"]
```

For the containers we will be building this may not be a large issue, but once your project becomes larger and you have more dependencies it can become a large time saver.

To see the layers that are created when you build your Dockerfile you can use the following command:

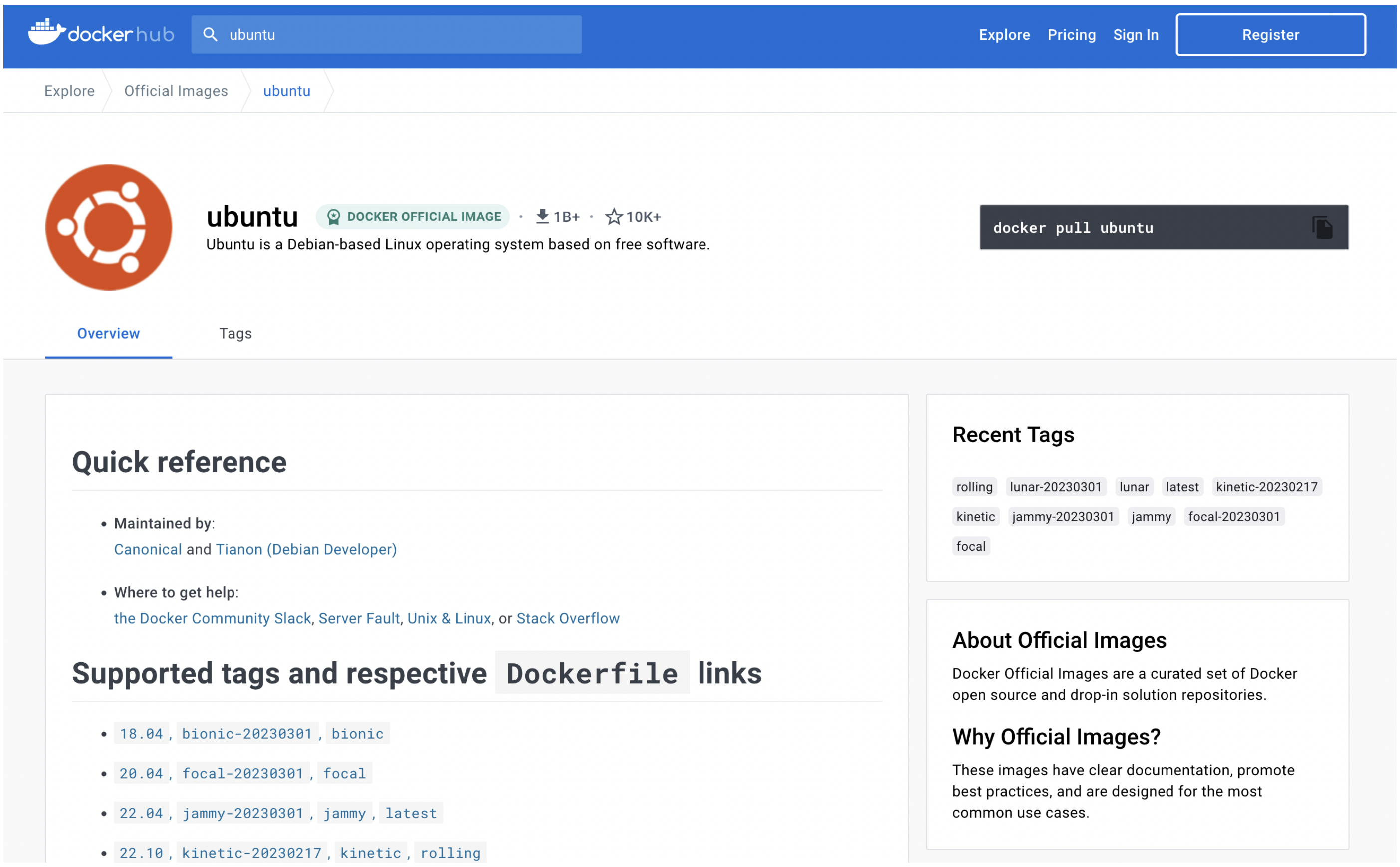
```
docker history <image name>

# Example
docker history ubuntu:latest
```

2.1.3 Image Registries

When we introduced the Dockerfile we glossed over the details of where `ubuntu:latest` is being sourced from. This image is coming from a registry. There are a few different registries that you can use. The default registry is one hosted by Docker themselves. This is the registry that is used when you don't specify a registry and is available at https://hub.docker.com/. There are also registries hosted by other companies such as Google and Amazon. These are available at https://cloud.google.com/container-registry and https://aws.amazon.com/ecr/ respectively.

As an example the image that we have been using is available at https://hub.docker.com/_/ubuntu. When browsing this page we can also see differnt tags that are available for this image, such as `latest` and `20.04`. These tags are used to specify different versions of the image so you can pin your image instead of using the `latest` tag which will always point to the latest version of the image. Note that not all images will have different tags or even a `latest` tag.



ubuntu DOCKER OFFICIAL IMAGE · 1B+ · 10K+

Ubuntu is a Debian-based Linux operating system based on free software.

```
docker pull ubuntu
```

Quick reference

- Maintained by:
  Canonical and Tianon (Debian Developer)
- Where to get help:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `18.04`, `bionic-20230301`, `bionic`
- `20.04`, `focal-20230301`, `focal`
- `22.04`, `jammy-20230301`, `jammy`, `latest`
- `22.10`, `kinetic-20230217`, `kinetic`, `rolling`

Recent Tags

rolling lunar-20230301 lunar latest kinetic-20230217 kinetic jammy-20230301 jammy focal-20230301 focal

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

Figure 2: Ubuntu image on Docker Hub

2.2 Docker Containers

TODO: continue fleshing this out

2.2.1 Docker Networking

TODO: continue fleshing this out

2.2.2 Docker Volumes

TODO: continue fleshing this out

2.3 Installing Docker

To install docker on your machine you can follow the instructions on the docker website: `https://docs.docker.com/get-docker/`.

2.4 Running a Docker Container

3 Dockerising our Todo Application

For the teacher

For this section let the students work through the practical on their own. The content above should be enough to guide them through the process with the provided instructions.

3.1 Creating a Dockerfile

3.2 Building our Docker Image

3.3 Running our Docker Container

3.4 Docker Compose

3.5 Moving to a Postgresql Database

4 Testing and Packaging our Application on Commit

For the teacher

Here we will describe the basics of CI and how we can use it to test and package our application. Run through this section and let the students use the provided completed file to get their repos running the tests and packaging on commit.

4.1 Intro to Continuous Integration

4.2 Intro to GitHub Actions

4.3 Testing on commit

4.4 Packaging on commit

References