Basics of Docker - Files, Images, & Containers

Docker

Let's start off with a common question, what exactly is Docker? In textbook terms, Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. That probably means nothing to you so let's simplify it and break it down for you. In simple terms Docker allows you to recreate an operating system or the software environment your program runs in. That means you no longer have to worry about a program not working on another machine when it works natively on yours. It's also this environment recreation that gives the most headaches as you'll find out in later chapters.

Basic Concepts

Docker files are the first step in many to using Docker. In essence, docker files are like the instructions for building the perfect software environment for your program to run in. It tells Docker what programs, dependencies, and/or repositories should be in a Docker container.

Docker images are basically the bare-bones OS your program needs to run. This could be Ubuntu Linx or something entirely different, a good rule of thumb to go with is to use whatever OS (image) your program natively runs in. You can find all official support images (OS) here.

Containers are what the docker files make after you run the build command on a docker file (more on that in another page). This is the environment your program is going to be in and is often almost completely isolated from the rest of a host system (more on that later).

Docker Prerequisites

As I've alluded to earlier Docker allows you to replicate the environment your program runs in, so that begs the question, how exactly am I supposed to do that? First things first you'll need Docker to run Docker, if you're on Windows or Mac you'll be basically forced to use Docker Desktop, if you're on Linux I recommend you install Lazy Docker. If its native documentation is confusing you can watch and follow this guide starting at 18:09 or email me at caicheng_li@yahoo.com. Other

than that you'll need code editing software such as Visual Studio Code (VS Code). Be sure to install the Docker extensions if you're going with VS Code.

Basic Info, Format, & Usage Examples

The default filename to use for a Dockerfile is `Dockerfile`, without a file extension. Using the default name allows you to run the `docker build` command without having to specify additional command flags. You can create one by starting a new file in your programming software and saving it as "Dockerfile" with no '.' or any other extensions.

That's it for the prerequisites let's get into how to actually create a docker file. The basic format of any docker file consists of the following functions: FROM, WORKDIR, RUN, COPY, and CMD. There are other more specialized functions for docker files such as ARG and ENV, however, for the purpose of this documentation, we won't be going over that. For the full list and explanation of docker files in its entirety go here. Other than that here's the simplified rundown of these functions:

`FROM <image>`	Defines a base for your image.
`RUN <command>`	Executes any commands in a new layer on top of the current image and commits the result. `RUN` also has a shell form for running commands.
`WORKDIR <directory>`	Sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY`, and `ADD` instructions that follow it in the Dockerfile.
`COPY <src> <dest>`	Copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>`.
`CMD <command>`	Lets you define the default program that is run once you start the container based on this image. Each Dockerfile only has one `CMD`, and only the last `CMD` instance is respected when multiple exist.

That's great but how are they set up?

FROM <Image>

Images are what come after "FROM" and define the base OS for your program. For example, if you've written a program that works in Ubuntu you can do the following:

```
FROM ubuntu:22.04
```

The `FROM` instruction sets your base image to the 22.04 release of Ubuntu. All instructions that follow are executed in this base image: an Ubuntu environment. The notation `ubuntu:22.04`, follows the `name:tag` standard for naming Docker images. When you build images, you use this notation to name your images. There are many public images you can leverage in your projects, by importing them into your build steps using the Dockerfile `FROM` instruction.

Docker Hubopen_in_new contains a large set of official images that you can use for this purpose.

RUN <Command>

The following line executes a build command inside the base image.

```
# install app dependencies
RUN apt-get update && apt-get install -y python3 python3-pip
```

This `RUN` instruction executes a shell in Ubuntu that updates the APT package index and installs Python tools in the container. Basically, this function acts as you typing in a terminal in your host system. Whatever program, repositories, or dependencies you install using commands like pip install or apt-get install is run using this function.

WORKDIR <directory>

This function simply acts as the cd command in so many terminals. It tells the docker where to install or copy programs you ask of it. Below is a common example of the WORKDIR command.

```
WORKDIR /app
```

Copy <src> <dest>

The `COPY` instruction copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>`. By default Copy copies the `<src>` files into whatever WORKDIR you're currently in, however, you can specify an existing directory or `<dest>` to use.

```
COPY tm-voicebox-v.3.7.4.py /app/tm-voicebox-v.3.7.4.py
```

In the above context, we're copying a local file titled "tm-voicebox-v.3.7.4.py" to the directory "/app/" set previously.

CMD [<command>]

Like RUN this function runs a terminal command, however, this is where the similarities end. CMD is usually used at the end of a docker file to specify how the user is going to interact with the program. For example:

```
CMD ["python", "/app/tm-voicebox-v.3.7.4.py"]
```

The above represents the execution of a Python script located at "/app/tm-voicebox-v.3.7.4.py" right when the docker file is run, more on that on another page. But you can also access the actual terminal within the docker container by ending the docker file with:

```
CMD ["bin/bash/"]
```

This allows the user access to the docker contain and can be useful for troubleshooting and development purposes.

Revision #8
Created 18 November 2023 04:28:22 by Caicheng Li
Updated 24 January 2024 00:50:01 by Daniel Odi