Docker

- Basics of Docker - Files, Images, & Containers
- Building, Running, & Packaging Dockerfiles
- Uploading to Tape Measure
- Example Program

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.

Building, Running, & Packaging Dockerfiles

This page is on what to do after you've created your docker file and how to run and package it to be uploaded.

> This page assumes your knowledge and understanding of Dockerfiles, how to create them properly, and general command line knowledge.

Building Docker Files

To build a docker file you need to utilize docker build commands. Before anything you have to make sure you're within the directory of your docker file. To do this simply right-click on the folder containing your docker file and "open in terminal" or in a command terminal cd into the directory.

Build Commands

To build a container image using the Dockerfile example from the previous section, you can use the `docker build` command:

```
sudo docker build -t test .
```

This is an example of the docker build command in Linux requiring the "sudo" function, if you're on Windows/Mac you might not need it. The `-t test` option specifies the name of the image.

The single dot (`.`) at the end of the command sets the build context to the current directory. This means that the build expects to find the Dockerfile and the `tm-voicebox-v.3.7.4.py` file in the directory where the command is invoked. If those files aren't there, the build fails.

There are also custom-build commands used for Tape Measure but we'll get to that later on. If you want to explore every build command option outside of this basic example go here.

Running Docker Containers

Once you successfully build your docker container you need to access or run it. This is where the run command comes in. Again, you'll have to be within the directory of your docker file for all of the

following directions.

Run Commands

To run the built container from the previous example you can use the `docker run` command.

```
sudo docker run -it test
```

In this instance "test" is the name of the container and, again, sudo might be required if you're on Linux. If you want to explore everything docker run has to offer outside of this basic example go here.

Packaging Docker Containers

To package a docker container and upload it into TM we need the docker image container to be compressed into a physical file instead of a container image. In order to do this we need to compile the docker container into a .tgz file. Below are the instructions to do so for Tape Measure.

Dockerfile to .tgz

The following terminal code segments were run in a Linux system, if you're on Windows/Mac you might need to adjust some of the terminal commands, if you run into a problem message @Cai or @dwight_2 on the discord server.

```
sudo docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
```

This command runs a throwaway container with full privileges, ensuring it has the latest qemu emulation binaries via the multiarch/qemu-user-static image. This is often used to run containers designed for another architecture (like ARM) on an x86 host which is what TM is built on.

```
sudo docker build -t tm-voicebox-v.10.6.8 --platform linux/arm64 --build-arg
BOSDYN_CLIENT_USERNAME=admin --build-arg BOSDYN_CLIENT_PASSWORD=<Insert Password> .
```

This command builds a Docker image for arm64 called tm-voicebox-v.10.6.8, passing credentials via build arguments. The Dockerfile to build the image is expected to be in the current working directory. To get the password please contact @dwight_2 on the RCCF discord server.

```
sudo docker save tm-voicebox-v.10.6.8 > tm-voicebox-v.10.6.8.tgz
```

This command saves the Docker image to a portable compressed archive file for transfer or backup. The tar archive contains all the layers and metadata of the image. The archive file can then be loaded into Docker again using `docker load` on TM but more on that on the next page.

Uploading to Tape Measure

This page showcases two ways you can upload your .tgz file into TM's core IO and how to load the container onto him.

Wireless Uploading

TBC

Physical USB

Frist things first make sure you have a USB big enough to store your .tgz file created previously after inserting it into your computer simply copy or move the .tgz file onto the USB. Using one of the USB slots in his core IO plug the USB into Tape Measure. Next, we'll SSH into him by running the command below:
*Note: replace "|" with @

```
ssh -p 20022 spot|192.168.80.3
```

This is the command if TM is on his own wifi however if he's on another wifi network the IP address will need to be changed. With that being said, it'll prompt you to insert a password which, again, you'll have to ask @dwight_2 for either on Discord or in person.

Next what you'll want to do is mount your USB. Since the core IO runs on Linux the following commands should just work when pasted in.

```
sudo fdisk -l
```

This command line lists all available USB and disk devices on TM. Once you spot your device name copy it. USB device name can look like: sda1, sda, etc.

```
sudo mount /dev/sda1 /data/usbin
```

This command line mounts the USB device sda1 to the directory /data/usbin. If you want to use a different directory make sure you specify it. Otherwise, you can create a directory by running:

```
sudo mkdir /data/usb-drive
```

This creates a new directory within data called USB-drive. Other than that just make sure you're within the directory of your USB device when running the following command line.

```
sudo docker load -i tm-voicebox-v.10.6.8.tgz
```

This loads your .tgz file as a container image on TM. That's it! Other than using the run command you've successfully uploaded the container image onto TM. Simply run your docker run command in order to access your imported container.

```
sudo docker run -it --device=dev/snd:/dev/snd tm-voicebox-v.10.6.8
```

Example Program

This is a basic example of how custom commands are created, dockerized, packaged, and then finally loaded onto Tape Measure.

The Python Script

Dockerizing

Packaging

Loading Via USB