How to Create and Run Docker Images

Become Familiar with Dockerfile Syntax

More information is available in the official Docker documentation: `https://docs.docker.com/engine/reference/builder/#from`.
A "cheatsheet":

`FROM` Create a new build stage and set the base image

- e.g., `FROM ubuntu:20.04`
- You will usually have one of these up at the top of your Dockerfile

`RUN` Run a command in the container

- e.g., `RUN apt-get update`
- You will use this for a majority of your instructions. Basically the same as running something in your terminal or in a shell script.

`COPY` Copy files/directories from the host machine to the container

- e.g., `COPY ./my_code.cpp /project/src/my_code.cpp`
- Be careful not to copy too many large items. Your image size may become very large!

Create a Dockerfile

The first step is to create a Dockerfile. This is where you'll put instructions that will tell Docker how to build your image.
In your text editor of choice or in your terminal, navigate to your project directory (this can be wherever you want it to be, but I recommend storing all your code and project files together). Create a new file with the name `Dockerfile` (note the capital "D" and lowercase letters following, lack of space, and lack of extension).

Open and Edit the Dockerfile

Open the `Dockerfile` in your editor. It should be blank, but we'll start adding things soon. Then, add a `FROM` instruction to the top of your Dockerfile. This will be the "base image" we use to build the rest of our environment. You can use any Docker image, but for now we'll use `ubuntu:20.04`. Your Dockerfile should now look like this:

```
FROM ubuntu:20.04
```

Add Instructions to the Dockerfile

Most software we containerize will need dependencies of some kind. Luckily, Ubuntu include a package manager that makes installing these very easy! We can use the `RUN` instructions along with the `apt-get` command in our Dockerfile to install packages. (Quick note: In a script or container, we need to include the `-y` flag when we use `apt-get`.)
Important: If you see a command that includes `sudo`, you will need to remove the `sudo` text from the command. Docker containers run as root by default, so `sudo` often isn't included in base images. If you forget to do this, you'll most likely see a "command not found" error.
Note that gRPC—the software we are going to create a Docker image for—requires the following dependencies:

- `wget`
- `build-essential`
- `libssl-dev`
- `git`

Your Dockerfile should now look something like this: (depending on what packages you want to install)

```
FROM ubuntu:20.04

RUN apt-get update  # Update the package index

RUN apt-get install -y wget build-essential libssl-dev git  # Install the dependencies
```

Look Up gRPC Documentation

In a web browser, go to https://grpc.io/docs/languages/cpp/quickstart/. Here you will find instructions for installing gRPC on Linux. Our task will be to convert the instructions given in the gRPC into instructions we can provide to Docker.
For example, one of the first instructions in the gRPC documentation is to set an environment variable called `MY_INSTALL_DIR`. If we were installing directly using a CLI or a shell script, we would type in

```
export MY_INSTALL_DIR=$HOME/.local # NOTE: DON'T RUN THIS!
```

We can convert it to a Dockerfile-compatible instruction:

```
ENV MY_INSTALL_DIR=$HOME/.local
```

So, our Dockerfile should now look like this:

```
FROM ubuntu:20.04

RUN apt-get update

RUN apt-get install -y wget build-essential libssl-dev git

ENV MY_INSTALL_DIR=$HOME/.local
```

The next step listed in the gRPC docs is to make sure the install directory exists. We can convert it by inserting a `RUN` instruction before the `mkdir` command. So, with this addition, the `Dockerfile` should look like:

```
FROM ubuntu:20.04

RUN apt-get update

RUN apt-get install -y wget build-essential libssl-dev git

ENV MY_INSTALL_DIR=$HOME/.local

RUN mkdir -p $MY_INSTALL_DIR
```

Testing Our Work

While we don't yet have a working gRPC Docker image yet, we can test what we've done so far. In your command line, navigate to the project directory where your `Dockerfile` resides.

```
cd ~/path/to/my/project/directory
```

Then, run the Docker build command:

```
docker build --file Dockerfile -t cse168grpc
```

Let's quickly break this command down:

- `docker` = Tell the OS that we would like to run a Docker command
- `build` = Specify that we would like to build a Docker image
- `--file Dockerfile` = Tell Docker which file to look for instructions include
- `-t cse168grpc` = Give our container a tag so we can use it later

Your Task

Your task will be to continue to follow the gRPC docs to create a working gRPC Docker image. You should be able to automate the installation process using your Dockerfile. If you do this successfully, it will make compiling and running gRPC code much easier! Hint: Lab 1 is all about gRPC!