Dockerfiles and Making Images

- Choosing Your Base Image
- Building Your Image

Choosing Your Base Image

Base images are, in a sense, the preexisting foundation that you will most likely build your containers upon. Often, it's a stripped down operating system like Alpine or Chiseled Ubuntu, but it can be something much more lightweight if needed, or you can even make your container completely from scratch by, well, adding `FROM scratch` to your Dockerfile. In this case, "scratch" is your base image, essentially meaning that nothing extra is added. This is most useful when trying to make your image as small and secure as possible. You will have to add every single dependency into your container by hand, however, so it can often be significantly more work to maintain.

Alpine

If you want to use a base like Alpine (a very lightweight Linux distro designed for containerization) for your image, you must be aware of the limitations and advantages that your base gives you. For instance, while Alpine is very small and minimal, it has it's own unique package manager invoked with `pkg`. Packages are installed by adding `pkg install [package_name]` to your Dockerfile while in a RUN instruction (this might look like `RUN pkg install ffmpeg` for instance), thus executing the package installation on the base image when the docker image is being built. Another potential drawback of Alpine is that it uses the incredibly lightweight and secure `musl libc` as its standard C library rather than `glibc` (the GNU C Library), which can affect compatibility with a number of pre-compiled Linux binaries. These will have to be recompiled for `musl libc` to work properly on Alpine, otherwise you may get assorted errors, such as these executable file not being found, even when present on the file system.

Building Your Image

To build your Docker image, make sure you are building within the same directory as your Dockerfile.

```
docker build -t myusername/myimage:tag .
```

Multi-platform builds, with multiple tags, and auto-push:

```
docker buildx build --push \
--platform linux/arm/v7,linux/arm64/v8,linux/amd64 \
--tag your-username/packagename:versionNum \
--tag your-username/packagename:latest .
```

If you just need to re-build the project due to updated dependencies, and don't want Docker to use the pre-cached steps of the build process, use `--no-cache` in the build command.