Docker Best Practices (TA1)

Michael P. Gerlek

mgerlek@inferlink.com

v1.1 / 3 Feb 2024

Docker is a tool used to provide a complete, reproducible, and portable Linux runtime environment for your applications. You can think of it as similar in spirit to pip: where pip uses a requirements.txt file to capture all your python dependencies, docker uses a Dockerfile file to capture all your system-level dependencies by allowing you to run any linux command, including pip, apt-get, and conda. When you start up your docker container, you can share (mount) directories (volumes) with your host OS, share (bind) ports to the host, specify which commands to run, and more. You can also store pre-built docker images on the web, to make sharing even easier.

From a software engineering perspective, docker allows a developer to specify exactly what applications, libraries, and services needs to be installed to run their applications, which means that someone else can fully reproduce the developer’s intended execution environment.

# Best Practices

1. **Use a Dockerfile:** Always build your docker image from a Dockerfile. Yes, you can use a prebuilt image and save (“commit”) changes to it, but that’s like manually editing a binary executable. You need to instead modify the source code, so that all the steps that went into defining your execution environment are human-readable (and, more importantly, auditable or modifiable!).
2. **FROM:** Your base image should be a well-known image from a trusted source, specified with the Dockerfile FROM command. The ubuntu or alpine general linux images are preferred. Sometimes the setup requirements are painful enough that more customized images like pytorch are better (such as when needing GPU support).
3. **CMD and ENTRYPOINT:** If your docker image is intended to only be used to run a specific tool, consider using the CMD and ENTRYPOINT commands in your Dockerfile. (A user can always override these when starting the container to get an interactive shell, for debugging purposes.)
4. **RUN:** Your Dockerfile should use the RUN command to execute package installers like pip, apt-get, and conda. Packages should be specified using exact versions (“pinned”) when possible. Also, when possible, use the most recent versions.
5. **RUN coalescing:** Minimize the number of RUN commands in your Dockerfile to minimize the size of the built docker image. Instead of two commands (e.g. RUN pip install foo and then RUN apt install bar), use one command (e.g. RUN pip install foo && apt install bar).
6. **VOLUME:** When you want to share a directory on the host machine with your running container, you will use the -v /host/path:/container/path option to the docker run command. If your image will always require a mounted directory, use the VOLUME /container/path command in your Dockerfile.
7. **COPY:** If you need to share a host directory with your container, but the contents of that directory are set a build time, use COPY to replicate the directory within the image instead of sharing it via -v and VOLUME.
8. **Tags:** Use tags to indicate the versions of your images. For example, the name interlink/tablex:v1.2 is much preferred to the name interlink/tablex-v1.2. Also, use the :latest tag convention to point to the most recent version.
9. **USER:** By default, your docker container will run as root. For local development this is usually fine, but when you start sharing an image or using it on exposed hosts, you should use a non-root user, e.g. with USER demo.