Teamwork

Build Docker Images

Thomas Domingues

Summary

- Introduction
- Dockerfile
  - Config example
  - FROM
  - COPY
  - CMD / ENTRYPOINT
  - Others: RUN / ARG / EXPOSE
- Build
- Volumes
- Practical work

Introduction

In the Docker basics, we learn to use Docker images, but despites being able to find a very large collection on Docker Hub, they occasionally meet exactly our needs.

The goal of this course is to learn how to build our own Docker images via Dockerfiles

Dockerfile - Config example

From hello-world image source code:

- First line: Defines the Base Image with FROM
- Second line and more: Instructions to build the image
- Last line: Default command to run

```
FROM scratch

COPY hello /

CMD ["/hello"]
```

Dockerfile - FROM

“The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction”

`stratch` is native from the Docker Engine, so for optimized Dockerfiles, it’s always better to start with it, but you can (and usually) start from any built image

https://docs.docker.com/engine/reference/builder/#from

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

Dockerfile - COPY

Syntax: `COPY [--chown=<user>:<group>] <src> <dest>`

Here, a simple file COPY is made from the source (hello file in current folder) to the docker image (at the root)

See source code of the folder

```
FROM scratch

COPY hello /

CMD ["/hello"]
```

https://docs.docker.com/engine/reference/builder/#copy

Dockerfile - CMD

Default command when running the image in a container. Can be override by specifying a command at the docker run after the image name

Here, the hello file is simply executed at the run and the container will stop once the hello script execution is done

```
FROM scratch

COPY hello /

CMD ["/hello"]
```

https://docs.docker.com/engine/reference/builder/#cmd

Dockerfile - ENTRYPOINT

Allows you to configure how a container will run a specific command (default or custom). Example for Dockerfile source code on the right and a built image named myphp:

```
FROM php

[...]

ENTRYPOINT ['/usr/local/bin/php']

CMD ["index.php"]
```

The command `docker run myphp` will run `/usr/local/bin/php index.php`, while `docker run myphp test.php` will run `/usr/local/bin/php test.php`.

If the entrypoint command was defined in the CMD alone (and not the ENTRYPOINT), the whole command would be replaced and php would not be called on a custom run.

https://docs.docker.com/engine/reference/builder/#cmd

Dockerfile - RUN

Execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

Example:

```
RUN apt install -y curl

RUN curl -sL https://deb.nodesource.com/setup_15.x | bash - && \
    apt-get install -y nodejs
```

https://docs.docker.com/engine/reference/builder/#run

Dockerfile - ARG

The ARG instruction defines a variable that users can pass at build-time to the builder with the docker build command using the `--build-arg <varname>=<value>` flag

```
ARG arg1=defaultValue
```

- For `docker build --build-arg arg1=customValue`, arg1 would equal `customValue`
- For `docker build`, arg1 would equal `defaultValue`

https://docs.docker.com/engine/reference/builder/#arg

Dockerfile - EXPOSE

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.

- Syntax: `EXPOSE <port>[/<protocol>]` (default: TCP)
- Example: `EXPOSE 80` for web applications
- Can be override at run: `docker run -p 443:443/tcp`

https://docs.docker.com/engine/reference/builder/#expose

Docker - build

Build an image from a Dockerfile

- Basic syntax: `docker build` (default: check for a Dockerfile in current directory and defines this last as the build context
- Advanced syntax: `docker build [-f <path_to_dockerfile>] [-t <image_tag>] [--build-arg <arg1>=<value1> …] <build_context_path>`

https://docs.docker.com/engine/reference/commandline/build

Docker - volumes

Persist data generated by and used by Docker containers.

- Volumes need to be created first before use: `docker volume create webapp-vol`
- At run, specify the volume: `docker run -v webapp-vol:/app mywebapp:latest`

Aims to replace the COPY in the Dockerfile for more flexibility and image size reduction

Volumes are very powerful, documentation reading is recommended:

https://docs.docker.com/storage/volumes

Practical work

- Create a Dockerfile at the root of your previous Git Project (Course 3)
- Write all instructions to make your project run, starting from an alpine image
  - Step 1: COPY all files from source to image as a Dockerfile instruction
  - Step 2: Use docker volume
- Build the docker image and run it. Successfully access to your web application at localhost
- Name your docker container “mywebapp” (find a way to do it in the Docker documentation)