Exercise 3

Dockerfile

A dockerfile gives instructions to docker how to build the image.

Dockerfile links

- Best practices for writing Dockerfiles
- Docker development best practices
- Use multi-stage builds

EXERCISE Docker images size

Depending on how you build your image, the size will differ. If you follow best practices and use the correct base image, your deploys will be faster and you will consume less disk space.

1. You will use the Python official images
2. Run the different official images for Python:

```
docker run --rm python:3.11-bullseye
docker run --rm python:3.11-buster
docker run --rm python:3.11-slim
docker run --rm python:3.11-alpine

# then check the image sizes with
docker images python

# Check the shared sizes
docker system df --verbose
```

Linux Base images

In this exercise you will learn how to write your own script run by docker. In this assignment we will use different linux distributions, debian and alpine. You will notice that available programs differs depending on which image you use.

All Dockerfile commands and examples can be found in Dockerfile reference. Also check the LINKS file for hints how to write your Dockerfile.

`EXERCISE` Dockerfile with a custom script

In this exercise you will copy a bash script into your images. To handle multiple Dockerfiles in the same folder, you can name your them something else than the default `Dockerfile` e.g `Dockerfile.alpine`

```
# A dockerfile named Dockerfile.alpine
docker build -t your_image_name -f Dockerfile.alpine .
```

1. You should create Dockerfiles that uses Python official images so first read their docs page. Create three Dockerfiles based on the following images:
    - python:3.11-bullseye
    - python:3.11-slim
    - python:3.11-alpine
2. Create a shell script that execute:
    - apt list --installed (on debian)
    - apk list --installed | sort (on alpine)
    - cat /etc/*-release
    - python --version

    e.g

    ```
    #!/bin/sh

    apt list --installed
    echo "----------"
    cat /etc/*-release
    echo "----------"
    python --version
    ```
3. In your `Dockerfile`, add the instruction `COPY` so it copy your scripts to the destination folder `/usr/src/app`
4. Add your script as a `CMD`
5. Build your images `Dockerfile.alpine`, `Dockerfile.slim`, `Dockerfile.bullseye`

```
# Don't forget the . at the end, this tells docker to look for the
`Dockerfile` in your current directory
docker build -f your_dockerfile_name -t your_image_name[s] .
```

6. Run your images and save the output in a screenshot, you should also check the image sizes.

```
docker run --rm -it your_image_name[s]

docker images
```

EXTRA Dockerfile multistage

The goal in this assignment is to create a multistage build, to learn more about layers, image size and build time.

1. First build a python package from source without the multistage, numpy. You should use the base image `python:3.11-alpine` and install with `RUN pip install --user numpy`.
2. Building from source can be tricky and you must read the `Prerequisites`. In this case we need `python3-dev`, `build-base` to be able to install. You can install packages on alpine linux with `RUN apk add --no-cache --update package-name`.
3. Add a test a default command to your Dockerfile that will test your numpy installation `CMD ["python", "-c", "import numpy; print(numpy.__version__)"]`
4. Run your newly built container! `docker run your-image-name`
5. Investigate the image size and build time, how does it differs from:
    1. a empty `python:3.11-alpine`
    2. a installation of numpy on `python:3.11-slim` you create another Dockerfile for this step
    3. Try to build the slim image many times
    4. Try to build the slim image with `--no-cache`

```
# List images with size
docker images

# You can compare the history.
docker history image_name
```

6. We should go back to our alpine version of the Dockerfile. Creating a multistage build:
    1. First we need to give the first stage a name "build": `FROM python:3.11-alpine AS build`.
    2. After the apk and pip step, add another `FROM python:3.11-alpine` that will become the final image.
    3. Add a copy of the built files `COPY --from=build /root/.local /root/.local`.
    4. Move the `CMD`command from step 3.
7. When you build the multi stage build, give the image a another name. How does the size differ from your previous builds? Investigate with docker history image-name