Dockerfile tuning

Dockerfile best practices

- **Separation of concern**: Ensure each Dockerfile is, as much as possible, focused on one goal. This will make it so much easier to reuse in multiple applications.
- **Avoid unnecessary installations**: This will reduce complexity and make the image and container compact enough.
- **Reuse already built images**: There are several built and versioned images on Docker Hub; thus, instead of implementing an already existing image, it's highly advisable to reuse by importing.
- **Have a limited number of layers**: A minimal number of layers will allow one to have a compact or smaller build. Memory is a key factor to consider when building images and containers, because this also affects the consumers of the image, or the clients.

Multi stage builds

Multi-stage builds are useful to anyone who has struggled to optimize Dockerfiles while keeping them easy to read and maintain. With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction can use a different base, and each of them begins a new stage of the build. You can selectively copy artifacts from one stage to another, leaving behind everything you don't want in the final image.

Permissions

The USER instruction sets the user name (or UID) and optionally the user group (or GID) to use as the default user and group for the remainder of the current stage. The specified user is used for RUN instructions and at runtime, runs the relevant ENTRYPOINT and CMD commands.

```
FROM ubuntu:latest
RUN apt-get -y update
RUN groupadd -r user && useradd -r -g user user
USER user
```

Multi stage builds

```
FROM node:10.15.2-alpine AS appbuild
WORKDIR /usr/src/app
COPY package.json ./
COPY .babelrc ./
RUN npm install
COPY ./src ./src
RUN npm run build
# Build Stage 2
# This build takes the production build from staging build
#
FROM node:10.15.2-alpine as runtime
WORKDIR /usr/src/app
COPY package.json ./
COPY .babelrc ./
RUN npm install
COPY --from=appbuild /usr/src/app/dist ./dist
EXPOSE 4002
CMD npm start
```

Build just a specific stage via:

docker build --target runtime -t hello .

Dockerfile tuning

1. always use **versioning** on images:

es. use

- FROM python:slim-bullseye instead of
- FROM python

2. **reduce size** of image

docker image ls python:latest → 921MB

docker image ls python:slim-bullseye → 126MB

docker image ls python:3.10-alpine → 48.7MB

Dockerfile tuning

3. run executables as **unprivileged users**:

define a USER in the Dockerfile with limited privileges

4. use **2 stages build** to separate between build stage and execution stage: this results in lightweight containers