Dockerfile

A **Dockerfile** is a text document containing a series of instructions on how to build a Docker image. Docker images are the basis for running containers, and Dockerfiles automate the creation of these images, allowing for reproducibility and consistency across environments. A Dockerfile defines all the steps needed to set up an environment for your application to run.

1. FROM

The FROM instruction specifies the base image from which the Docker image will be built. This is typically the first line in a Dockerfile, and it pulls an image from Docker Hub or another registry.

Example: Dockerfile
FROM ubuntu:20.04

In this case, the base image is ubuntu:20.04

2. RUN

The RUN instruction executes commands inside the container during the build process. It is typically used to install dependencies or configure the environment

Example: Dockerfile
RUN apt-get update && apt-get install -y python3

This command updates the package index and installs Python3 in the container

RUN can be used in two forms:

1. **Shell form**:
   RUN echo "Hello World"
2. **Exec form**:
   RUN ["echo", "Hello World"]

3. COPY and ADD

The COPY instruction is used to copy files or directories from the host machine into the Docker image

Example: Dockerfile
COPY ./src /app

This copies the contents of the ./src directory from the host into the /app directory in the container.

The ADD instruction is similar to COPY, but it has additional features such as automatically extracting tar archives and supporting URLs. However, it's generally recommended to use COPY unless these extra features are needed.

Example: Dockerfile
ADD ./archive.tar.gz /app

4. WORKDIR

The WORKDIR instruction sets the working directory for subsequent instructions in the Dockerfile. This means that any command after this instruction will run relative to this directory.

Example: Dockerfile
WORKDIR /app

5. ENV

The ENV instruction sets environment variables inside the Docker container. These variables can be accessed later by commands or the running application.

Example: Dockerfile
ENV APP_ENV=production

6. EXPOSE

The EXPOSE instruction informs Docker that the container will listen on the specified network ports at runtime. This is mostly used for documentation purposes but can also be used with the -P or -p flags when running the container to map the ports.

Example: Dockerfile
EXPOSE 8080

7. CMD

The CMD instruction provides the default command to run when the container starts. If the container is run without specifying a command, this command is executed. It can be overridden by passing an alternative command when running the container.

Example: Dockerfile
CMD ["python3", "app.py"]

There are three forms of CMD:
CMD ["executable", "param1", "param2"] (exec form, preferred)
CMD ["param1", "param2"] (default parameters for ENTRYPOINT)
CMD command (shell form, e.g., CMD echo "Hello World")

8. ENTRYPOINT

The ENTRYPOINT instruction sets the command that will always run when the container starts. It's similar to CMD, but ENTRYPOINT cannot be overridden by arguments passed to docker run, while CMD can be overridden. Often used in combination with CMD to provide default arguments.

Example: Dockerfile
ENTRYPOINT ["python3", "app.py"]

9. VOLUME

The VOLUME instruction creates a mount point for external storage, enabling data persistence. This can be used to mount a directory from the host machine into the container, ensuring that the data persists even after the container stops.

Example: Dockerfile
VOLUME ["/data"]

10. USER

The USER instruction sets the user to use when running the container. By default, containers run as the root user, but it's a good practice to run containers as a non-privileged user for security reasons.

Example: Dockerfile
USER appuser

11. ARG

The ARG instruction defines build-time variables, which can be used to parameterize the build process. These values are passed in when building the image and can be used in the Dockerfile.

Example: Dockerfile
ARG VERSION=1.0

You can pass values to ARG during the build using the --build-arg flag:
docker build --build-arg VERSION=2.0 .

12. LABEL

The LABEL instruction adds metadata to the image. This metadata can include the image's version, description, authorship, or any other relevant information.

Example: Dockerfile
LABEL version="1.0"
LABEL description="This is a custom application"

13. SHELL

The SHELL instruction allows you to change the default shell used by the Dockerfile for RUN instructions. By default, Docker uses /bin/sh -c, but you can change it to something like bash if needed.

Example: Dockerfile
SHELL ["/bin/bash", "-c"]

ARG vs ENV

`ARG` is for build-time variables, meaning they are only available during the build process.

`ENV` is for run-time variables, meaning they can be accessed when the container is running.

Example of a Simple Dockerfile

Here’s an example of a basic Dockerfile for a Python application:

Dockerfile

```
# Step 1: Specify the base image

FROM python:3.9-slim

# Step 2: Set environment variables

ENV APP_HOME=/app

WORKDIR $APP_HOME

# Step 3: Copy the application files into the container

COPY . .

# Step 4: Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Step 5: Expose the application port

EXPOSE 5000

# Step 6: Set the default command to run the app

CMD ["python", "app.py"]
```