AJDEVOPSSOLUTIONS

DOCKER PRACTICALS-TRAINEDY BY AJAY BONGANI

A Dockerfile is a script comprised of various instructions, each serving a specific purpose in the process of building a Docker image

FROM

*- **Purpose:** Specifies the base image from which you are building.*
*- **Example**: `FROM ubuntu:18.04`*
*- **Explanation:** This instruction initializes a new build stage and sets the Base Image for subsequent instructions. For instance, using `ubuntu:18.04` as a base provides a minimal environment with Ubuntu 18.04.*

WORKDIR

*- **Purpose**: Sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY`, and `ADD` instructions that follow it.*
*- **Example**: `WORKDIR /app`*
*- **Explanation**: It's used to define the working directory (or the 'context') of your Docker container. If the directory does not exist, it will be created. This avoids using full paths in subsequent instructions.*

COPY

*- **Purpose**: Copies **new files or directories** from the source and adds them to the filesystem of the container at the path specified.*
*- **Example**: `COPY . /app`*
*- **Explanation**: Typically used to copy application source code into the container. Unlike `ADD`, `COPY` does not handle **local tar files or URL sources.***

ADD

*- **Purpose**: Similar to `COPY`, but can handle remote URLs and unpack local tar files.*
*- **Example**: `ADD https://example.com/big.tar.xz /usr/src/things/`*
*- **Explanation**: While it's more versatile than `COPY`, it's recommended to use `COPY` for copying local files as the intent is clearer.*

RUN

*- **Purpose**: Executes commands on the top layer of the Docker image and commits the results.*
*- **Example**: `RUN **apt-get update && apt-get install -y git**`*
*- **Explanation**: Used for installing software packages within the container. Each `RUN` instruction creates a new layer in the image, so combining commands helps reduce the number of layers created.*

CMD

*- **Purpose**: Provides defaults for an executing container. There can only be one `CMD` instruction in a Dockerfile.*
*- **Example**: `CMD ["python", "./app.py"]`*
*- **Explanation**: Specifies the command to run when the container starts. If multiple `CMD` instructions are listed**, only the last `CMD` will take effect.***

ENTRYPOINT

- ***Purpose***: *Configures a container that will run as an executable.*
- ***Example***: *`ENTRYPOINT ["python", "./app.py"]`*
- ***Explanation***: *Unlike `CMD`, it does not get overridden when Docker runs with command-line arguments. This is useful for containers that should always run as an executable or for when you want to pass arguments to the entry point.*

ENV

- ***Purpose***: *Sets the environment variable.*
- ***Example***: *`ENV MY_NAME="John Doe"`*
- ***Explanation***: *`ENV` can be used to provide dynamic configuration to the application running in the container, making your application environment agnostic.*

EXPOSE

- ***Purpose***: *Informs Docker that the container listens on specific network ports at runtime.*
- ***Example***: *`EXPOSE 8080`*
- ***Explanation***: *It's a way of documenting which ports are intended to be published. However, `EXPOSE` does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container.*

VOLUME

- ***Purpose***: *Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.*
- ***Example***: *`VOLUME /data`*
- ***Explanation***: *Useful for when you want to store data outside the container, ensuring data persists even after the container is destroyed.*

USER

- ***Purpose***: *Sets the username or UID to use when running the image and for any `RUN`, `CMD`, and `ENTRYPOINT` instructions that follow it.*
- ***Example***: *`USER nobody`*
- ***Explanation***: *Enhances security by allowing you to run applications as a non-root user even within the container.*

HEALTHCHECK

- ***Purpose***: *Tells Docker how to test a container to check that it is still working.*
- ***Example***: *`HEALTHCHECK CMD curl --fail http://localhost:8080/ || exit 1`*
- ***Explanation***: *This instruction helps Docker determine the health status of the container by running a command inside it. If the command exits with a zero status, the container is considered healthy.*

ONBUILD

- ***Purpose***: *Adds a trigger instruction to be executed at a later time, when the image is used as the base for another build.*
- ***Example***: *`ONBUILD RUN echo "This will run when the image is used as a base."`*
- ***Explanation***: *Useful for images intended to be used as base images, allowing you to defer some configuration until the image is extended.*

The details of each instruction, highlighting their differences and appropriate use cases.

FROM vs. WORKDIR

- FROM

- Purpose: Initiates a build stage by setting the base image for subsequent instructions. It's the foundation upon which your image is built.

- Use Case: Essential for all Dockerfiles as it defines the starting point, such as `FROM ubuntu:20.04` for an Ubuntu-based image.

- WORKDIR

- Purpose: Sets the working directory for any `RUN`, `CMD`, `COPY`, and `ENTRYPOINT` instructions that follow in the Dockerfile.

- Use Case: To define a working directory (e.g., `WORKDIR /app`) for the application within the container, simplifying subsequent paths.

COPY vs. ADD

- COPY

- Purpose: Copies files and directories from the build context into the Docker image.

- Use Case: Ideal for copying local files into the container, such as source code.

- ADD

- Purpose: Similar to `COPY` but with the added capability of handling remote URLs and auto-extracting compressed files.

- Use Case: Suitable for situations where you need to add files from a URL or extract tar files directly into the image, though its use for local files is generally discouraged in favor of `COPY`.

RUN vs. CMD vs. ENTRYPOINT

- RUN

- Purpose: Executes commands during the image build process and commits the results, creating a new image layer.

- Use Case: Installing software packages, building software, or performing setup tasks within the image.

- CMD

- Purpose: Provides default execution command for an image, which can be overridden by command-line arguments when the container starts.

- Use Case: Setting a default application to run when the container starts, such as `CMD ["nginx", "-g", "daemon off;"]`. It's meant to provide defaults for an executing container.

- ENTRYPOINT

- Purpose: Configures a container to run as an executable; command-line arguments passed to `docker run` append to the defined entry point.

- Use Case: When building executable containers that should always run with certain commands or parameters, e.g., `ENTRYPOINT ["python", "/usr/src/app.py"]`, allowing additional arguments to be passed at runtime.

ENV vs. ARG

- ENV
- Purpose: Sets environment variables within the Docker image.
- Use Case: Configuring software within the container to use certain settings, paths, or external services, e.g., `ENV PATH="/app/bin:${PATH}"`.
- ARG
*- Purpose: Allows **you to pass variables at build time** to be used in Dockerfile instructions.*
- Use Case: To dynamically set values during the build, such as version numbers or paths that may change between builds, e.g., `ARG VERSION=1.0`.

EXPOSE vs. VOLUME

- EXPOSE
- Purpose: Indicates that the container listens on specified network ports at runtime. It's a form of documentation and does not actually publish the port.
- Use Case: Informing Docker and users of the image about which ports the container should make available, e.g., `EXPOSE 80`.
- VOLUME
- Purpose: Creates a mount point in the container and marks it to hold externally mounted volumes from the host or other containers.
- Use Case: To persist data generated by and used by Docker containers, e.g., `VOLUME ["/data"]`.

USER vs. HEALTHCHECK

- USER
- Purpose: Sets the username or UID to use when running the image and for any `RUN`, `CMD`, or `ENTRYPOINT` instructions that follow it.
- Use Case: Enhancing security by ensuring services run under a non-root user, e.g., `USER nobody`.
- HEALTHCHECK
- Purpose: Tells Docker how to test a container to check that it is still working as expected.
- Use Case: To automatically check the health of applications running within a container, e.g., `HEALTHCHECK CMD curl --fail http://localhost:8080/health || exit 1`.

ONBUILD

- Purpose: Adds a trigger instruction to be executed later, when the image is used as the base for another build.
- Use Case: Preparing base images that require additional commands to be run at build time of a child image, e.g., `ONBUILD ADD . /app/`.

NOTE:

Each of these Dockerfile instructions serves a unique role in the lifecycle of a Docker image, from building and configuring to running and managing the application within containers.