Dockerfile Tutorial

A Dockerfile is a script with a set of instructions on how to build a Docker image. Each instruction in a Dockerfile creates a layer in the image, making it easier to update images efficiently. Below is a comprehensive guide on how to write a Dockerfile with explanations for each important line.

#### **Basic Structure**

1. FROM
2. RUN
3. COPY/ADD
4. WORKDIR
5. ENV
6. EXPOSE
7. CMD
8. ENTRYPOINT
9. VOLUME
10. ARG
11. LABEL

### **Detailed Explanation with Examples**

#### **1. FROM**

Specifies the base image to use for the Docker image. This is usually the first line in a Dockerfile.

# Use an official Python runtime as a parent image

FROM python:3.9-slim

#### **2. RUN**

Executes commands in the image during the build process. Often used to install software packages.

# Update and install any necessary dependencies

RUN apt-get update && apt-get install -y \ build-essential \ libssl-dev \ && rm -rf /var/lib/apt/lists/\*

#### **3. COPY and ADD**

Both instructions add files from your host machine to the Docker image. COPY is preferred for simple file copying. ADD can also extract local tar archives and download files from URLs.

# Copy the current directory contents into the container at /app

COPY . /app # Add a tar file into the image ADD my\_archive.tar.gz /app

#### **4. WORKDIR**

Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it.

# Set the working directory to /app

WORKDIR /app

#### **5. ENV**

Sets environment variables.

# Set environment variables

ENV NAME World

#### **6. EXPOSE**

Informs Docker that the container listens on the specified network ports at runtime. It doesn't actually publish the port; you need to use docker run -p to do that.

# Make port 80 available to the world outside this container

EXPOSE 80

#### **7. CMD**

Specifies the command to run within the container. Unlike RUN, it is not executed during the build, but when a container is instantiated from the image.

# Run app.py when the container launches

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

#### **8. ENTRYPOINT**

Sets the command that will run as the main process of the container. It can be combined with CMD to provide default arguments.

# Set the entry point to the Python interpreter

ENTRYPOINT ["python"]

# Provide default arguments to the entry point CMD ["app.py"]

#### **9. VOLUME**

Creates a mount point with the specified path and marks it as holding externally mounted volumes from the host or other containers.

# Create a mount point with the specified path

VOLUME /data

#### **10. ARG**

Defines a variable that users can pass at build-time to the builder.

# Define a build-time variable ARG VERSION=1.0 # Use the ARG variable RUN echo "Version: $VERSION"

#### **11. LABEL**

Adds metadata to an image. Labels are a way to add key-value metadata to your Docker images.

# Add metadata to an image LABEL version="1.0" LABEL description="This is an example image" LABEL maintainer="you@example.com"

### **Complete Example**

Here is a complete example combining several of the above instructions to create a Docker image for a simple Python application.

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set environment variables

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

# Set the working directory

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any necessary dependencies

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

# Make port 80 available to the world outside this container

EXPOSE 80

# Define the default command to run when starting the container

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

### **Building and Running the Docker Image**

1. Build the Docker image:

docker build -t my-python-app .

1. Run the Docker container:

docker run -p 4000:80 my-python-app

By following this guide, you should have a solid understanding of how to create and utilize a Dockerfile to automate the building of Docker images.