## **Dockerfile Overview**

A **Dockerfile** is a text file containing a series of instructions that Docker uses to create an image. Each instruction represents a step for configuring the image, such as specifying the base image, adding files, installing packages, or defining the commands to run when the container starts.

### **Dockerfile Basics**

Here are some commonly used instructions in a Dockerfile:

* **FROM**: Specifies the base image to use. Every Dockerfile must begin with this instruction.
* **RUN**: Executes commands in the container, such as installing software or configuring settings.
* **COPY** or **ADD**: Copies files or directories from the host system into the container.
* **WORKDIR**: Sets the working directory for subsequent instructions.
* **CMD**: Specifies the default command to execute when the container starts.
* **EXPOSE**: Indicates the network ports the container will use.
* **ENV**: Sets environment variables inside the container.

### **Example 1: Basic Dockerfile Using Ubuntu**

#### **Dockerfile:**

# Use the latest Ubuntu image as the base image

FROM ubuntu:latest

# Update package manager and install curl

RUN apt-get update && apt-get install -y curl

# Copy a script into the container

COPY hello.sh /hello.sh

# Make the script executable

RUN chmod +x /hello.sh

# Define the default command to run the script

CMD ["/hello.sh"]

#### **hello.sh:**

#!/bin/bash

echo "Hello from Docker container!"

tail -f /dev/null

#### **Steps to Build and Run:**

Create the files in a directory (e.g., mydockerapp/).

**Build the image:**

docker build -t my-ubuntu-app .

**Run the container:**

docker run my-ubuntu-app

**Output:**

Hello from Docker container!

### **Example 2: Dockerfile for a Node.js Application**

#### **Dockerfile:**

# Use an official Node.js runtime as the base image

FROM node:14

# Set the working directory inside the container

WORKDIR /usr/src/app

# Copy package files and install dependencies

COPY package\*.json ./

RUN npm install

# Copy application files

COPY . .

# Expose the application port

EXPOSE 3000

# Define the command to run the app

CMD ["node", "app.js"]

#### **app.js:**

const http = require('http');

const hostname = '0.0.0.0';

const port = 3000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello from Node.js Docker container!\n');

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

#### **Steps to Build and Run:**

Create the files in a directory (e.g., node-docker-app/).

**Build the image:**

docker build -t my-node-app .

**Run the container with port mapping:** docker run -d -p 3000:3000 my-node-app

1. **Access the application:** Visit http://<your-server-ip>:3000.

### 

### 

### **Example 3: Dockerfile for a Python Flask Application**

#### **Project Structure:**

/my-flask-app

|-- Dockerfile

|-- app.py

|-- requirements.txt

#### **Dockerfile:**

# Use the official Python image with version 3.8

FROM python:3.8-slim

# Set the working directory

WORKDIR /usr/src/app

# Copy application files

COPY . .

# Install dependencies

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

# Expose the Flask app port

EXPOSE 5000

# Define the command to run the Flask app

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

#### **app.py:**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return "Hello from Flask!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

#### **requirements.txt:**

Flask==2.0.3

#### **Steps to Build and Run:**

1. Create the files in a directory (e.g., my-flask-app/).

**Build the image:**

docker build -t my-flask-web-app .

**Run the container:**

docker run -p 5000:5000 my-flask-web-app

**Access the application:** Visit http://<your-server-ip>:5000.

### **Key Notes:**

1. **0.0.0.0 Hostname:**
   * This binds the server to all available network interfaces, making the application accessible from external devices.
2. **Port Mapping:**
   * Use the -p flag to map container ports to host machine ports (docker run -p <host-port>:<container-port>).
3. **Building Docker Images:**
   * Use docker build -t <image-name> . to build the image. The . specifies the current directory as the build context.
4. **Running Docker Containers:**
   * Use docker run <image-name> to start a container.