

Dockerfile examples

Dockerfile for simple Apache Server

```
1 FROM ubuntu
2 RUN apt update
3 RUN apt install -y apache2
4 RUN apt install -y apache2-utils
5 RUN apt clean
6 EXPOSE 80
7 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Dockerfile to install Apache and PHP

```
1 FROM ubuntu
2 RUN apt update
3 RUN apt install -y apache2
4 RUN apt install -y apache2-utils
5 RUN apt install -y php
6 RUN apt clean
7 EXPOSE 80
8 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

```
ot@devops-raghav:~# docker build -t apaphp .
[+] Building 29.3s (7/9)
> [internal] load build definition from Dockerfile
> => transferring dockerfile: 212B
> [internal] load .dockerignore
> => transferring context: 2B
> [internal] load metadata for docker.io/library/ubuntu:latest
> CACHED [1/6] FROM docker.io/library/ubuntu@sha256:aabed3296a3d45cedelcdc866a24476c4d7e093aa806263c27ddaadbdc3c1054
> [2/6] RUN apt update
> [3/6] RUN apt install -y apache2
> [4/6] RUN apt install -y apache2-utils
> [5/6] RUN apt install -y php
> # Please select the geographic area in which you live. Subsequent configuration
> # questions will narrow this down by presenting a list of cities, representing
> # the time zones in which they are located.
> # 1. Africa 3. Antarctica 5. Arctic 7. Atlantic 9. Indian 11. US
> # 2. America 4. Australia 6. Asia 8. Europe 10. Pacific 12. Etc
> # Geographic area:
```

Installation halts, waiting for your input. But docker build is not an interactive command so it finally fails

Modified Dockerfile to install Apache and PHP

```
1 FROM ubuntu
2 ENV TZ=Asia/Kolkata
3 RUN apt-get update
4 RUN apt-get install -y tzdata
5 RUN apt-get install -y apache2
6 RUN apt-get install -y apache2-utils
7 RUN apt-get install git -y
8 RUN apt update
9 RUN apt install -y libz-dev libssl-dev libcurl4-gnutls-dev libexpat1-dev gettext cmake gcc
10 RUN apt-get install -y nano && \
11     apt-get install -y wget && \
12     rm -fr /var/lib/apt/lists/*
13 RUN apt-get clean
```

```
14 EXPOSE 80
15 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Dockerfile for an Ubuntu-based LAMP (Linux, Apache, MySQL, PHP) stack

```
1 # Use the official Ubuntu image as the base
2 FROM ubuntu:20.04
3
4 # Set environment variables to avoid interactive prompts during installation
5 ENV DEBIAN_FRONTEND=noninteractive
6
7 # Install required packages for LAMP stack
8 RUN apt-get update && apt-get install -y \
9     apache2 \
10    mysql-server \
11    php \
12    php-mysql \
13    php-apcu \
14    libapache2-mod-php \
15    && apt-get clean \
16    && rm -rf /var/lib/apt/lists/*
17
18 # Configure Apache web server
19 RUN a2enmod rewrite
20 RUN echo 'ServerName localhost' >> /etc/apache2/apache2.conf
21
22 # Start services
23 CMD ["apache2ctl", "-D", "FOREGROUND"]
24
25 # Expose ports
26 EXPOSE 80 3306
```

In this Dockerfile:

1. We start with the official Ubuntu 20.04 image as the base.
2. We set an environment variable to avoid interactive prompts during package installation.
3. We update the package repository and install the necessary components for a LAMP stack, including Apache, MySQL, and PHP.
4. We enable the Apache `mod_rewrite` module for URL rewriting.
5. We configure Apache to set the "ServerName" to "localhost."
6. We specify that Apache should run in the foreground as the main process.
7. We expose ports 80 (HTTP) and 3306 (MySQL) for external access.

Dockerfile for Ubuntu-based LEMP(Linux, Nginx, MySQL (or MariaDB), and PHP)

```
1 # Use the official Ubuntu image as the base
2 FROM ubuntu:20.04
3
4 # Set environment variables to avoid interactive prompts during installation
5 ENV DEBIAN_FRONTEND=noninteractive
6
7 # Update the package repository and install necessary packages
8 RUN apt-get update && apt-get install -y \
9     nginx \
10    mysql-server \
11    php-fpm \
12    php-mysql \
```

```

13     php-common \
14     php-gd \
15     php-cli \
16     php-curl \
17     php-json \
18     php-zip \
19     php-mbstring \
20     php-xmlrpc \
21     php-xml \
22     php-bcmath \
23     && apt-get clean \
24     && rm -rf /var/lib/apt/lists/*
25
26 # Configure PHP-FPM
27 RUN sed -i 's/;cgi.fix_pathinfo=1/cgi.fix_pathinfo=0/' /etc/php/7.4/fpm/php.ini
28
29 # Start services
30 CMD service mysql start && service php7.4-fpm start && nginx -g 'daemon off;'
31
32 # Expose ports
33 EXPOSE 80
34
35 # Create a directory for the Nginx site configuration
36 RUN mkdir -p /var/www/html
37
38 # Copy a sample Nginx site configuration file
39 COPY default.conf /etc/nginx/sites-available/default
40
41 # Create a symbolic link to enable the site configuration
42 RUN ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/
43
44 # Remove the default Nginx index.html file
45 RUN rm /var/www/html/index.nginx-debian.html

```

In this Dockerfile:

1. We start with the official Ubuntu 20.04 image as the base.
2. We set an environment variable to avoid interactive prompts during package installation.
3. We update the package repository and install Nginx, MySQL, and PHP packages.
4. We configure PHP-FPM to disable `cgi.fix_pathinfo` for security reasons.
5. We start MySQL, PHP-FPM, and Nginx as the main processes.
6. We expose port 80 for HTTP traffic.
7. We create a directory for the Nginx site configuration and copy a sample Nginx site configuration file.
8. We create a symbolic link to enable the site configuration.
9. We remove the default Nginx index.html file.

You should customize the Nginx site configuration file (`default.conf`) and add your PHP application code to the `/var/www/html` directory.

Dockerfile for a basic Node.js application:

```

1 # Use an official Node.js runtime as the base image
2 FROM node:14
3
4 # Set the working directory in the container
5 WORKDIR /app

```

```
6
7 # Copy package.json and package-lock.json to the working directory
8 COPY package*.json ./
9
10 # Install application dependencies
11 RUN npm install
12
13 # Copy the rest of the application source code to the working directory
14 COPY . .
15
16 # Expose a port that the application will listen on
17 EXPOSE 3000
18
19 # Define a command to run the application
20 CMD ["node", "app.js"]
```

In this example:

1. We start with the official Node.js Docker image version 14 as our base image.
2. We set the working directory to `/app` inside the container.
3. We copy `package.json` and `package-lock.json` into the container to install dependencies.
4. We run `npm install` to install the application dependencies.
5. We copy the rest of the application source code into the container.
6. We expose port 3000 for the application to listen on.
7. We specify the command to run when the container starts, which is `node app.js`.