1.) Create a instance

2.) Download docker

3.) Vi Dockerfile D capital letter

4.) Insert i

5.) FROM openjdk:17
MAINTAINER Pankaj
RUN echo 'run-1'
RUN echo 'run-2
CMD echo 'cmd-1'
CMD echo 'cmd-2'

6.) Esc —> :wq

docker build -t pankajsiracademy/image1 .

It real project first we have to clone it then we have to use build by going in to project and use . to check in present directory

Dockerfile is in present directory only so no need for going in to some thing

Multiple cmd only last will execute rest over ride so we use entry point

How to create docker file? Create Docker Image? push that to docker hub?

In order to build a docker file as a developer you should know when to use the following keywords.Dockerfile instructions (FROM, CMD, ADD, etc.) are case-sensitive and must be written in uppercase.

1. FROM

_____

->The FROM instruction in a Dockerfile specifies the base image for your container or it will be used to specify what softwares should be downloaded to run our app. It is always the first instruction in a Dockerfile because it defines the environment where your application will run

FROM <image>:<tag>

<image> → Name of the base image (e.g., ubuntu, node, python)
<tag> → (Optional) Specifies the image version

Example:

FROM python:3.9
FROM openjdk:17
FROM tomcat:9.0

2. MAINTAINER

_____

-> The MAINTAINER instruction was used in older Docker versions to specify the author of the Dockerfile.
Example of old version - MAINTAINER Your Name <your-email@example.com> (This is deprecated)

Example for new version - LABEL maintainer="Your Name <your-email@example.com>"

3. RUN

_____

-> The RUN instruction in a Dockerfile is used to execute commands during the image build process.

Example :

RUN 'git clone <repo-url>'

RUN 'mvn clean package'

Note: If you specify multiple RUN instructions in Dockerfile, then those will execute in sequential manner.

4. CMD

_____

-> The CMD instruction in a Dockerfile specifies the default command that runs when the container starts

CMD "java -jar myapp.jar"

Note: when we write multiple CMD instructions in dockerfile, docker will execute only last CMD instruction only.

Let us create our first docker file:

_____

Step 1: vi dockerfile

```
FROM openjdk:17
MAINTAINER Pankaj
RUN echo 'run-1'
RUN echo 'run-2
CMD echo 'cmd-1'
CMD echo 'cmd-2'
```

Step 2: create docker image using the command
-> docker build -t pankajsiracademy/image1 .

we have used dot (.) in above command to mention that dockerfile in present in same working directory

Step 3: To check image created use the command
--> docker image

Step 4: to delete the image
--> docker rmi image-id

Step 5: You can again create the image
--> -> docker build -t pankajsiracademy/image1 .

Step 6: Run your docker image
--> docker run [image-id]

Note:
1. Only last CMD instruction will run
2. docker run [image-id] echo 'hello world', you will notice now hello world will execute and not the last CMD of docker file

To over come the above said instruction we can use

5. ENTRYPOINT

_____

-> Alternative command for CMD but the advantage is we cannot override this command

Example-1 : CMD "java -jar app.jar"

Example-2 : ENTRYPOINT ["java", "-jar", "app.jar"]

6. COPY

_____

--> COPY instruction will copy the files from source to destination.
--> It is used to copy application code from host machine to container machine.
--> Here Source means "HOST Machine" and Destination means "Container machine"

Example : COPY target/your-app.jar /usr/app/

7. ADD

_____

--> ADD instruction will copy the files from source to destination same as COPY, But in addition it can extract your compressed tar file
--> Here Source can be host machine

--> ADD cannot download from HTTP/S3 URLs—this is a misconception. (Be carefull in interviews)

Example:
ADD target/app.jar /usr/app/

ADD <http-url> /usr/app/ (Wrong)

8. WORKDIR

———————————

--> The WORKDIR instruction sets the working directory (Like cd command)
--> If the directory does not exist, Docker will automatically create it

Example: WORKDIR /path/to/directory

COPY target/your-app.jar   /usr/app/
WORKDIR /usr/app/
CMD "java -jar your-app.jar"

9. EXPOSE

———————————————

-> EXPOSE command is used to specify application is running on which PORT number. Using this you cannot change the port number of application. You are just mentioning that our application is running on the port number 9090.

-> If our application is running on port number 8081 you are mentioning expose 9090 then it is wrong.

-> It is optional to use

Example : EXPOSE 9090

Now time for practicals guys, Let us do Dockerizing Spring boot application

———————————————————————————————————————

Sure! Let's break down each line of your `Dockerfile`:

dockerfile
CopyEdit
```
FROM openjdk:17
```

- This sets the base image for your Docker container.
- `openjdk:17` is a pre-built image with Java 17 installed.
- This is needed to run your Spring Boot `.jar` file, which is a Java application.

---

dockerfile
CopyEdit
```
COPY target/demo-app.jar /usr/app/
```

- This copies the compiled Spring Boot JAR file (`demo-app.jar`) from your local machine's `target` folder (created after Maven/Gradle build) into the container's directory `/usr/app/`.

---

dockerfile
CopyEdit
```
WORKDIR /usr/app/
```

- Sets the working directory inside the container.
- All subsequent commands (like `ENTRYPOINT`) will be run from this directory.
- So now the container is "located" at `/usr/app`.

---

dockerfile
CopyEdit

```
EXPOSE 8080
```

- **Tells Docker that this container expects to serve traffic on port 8080.**
- **This is the default port Spring Boot uses.**

It doesn't actually publish the port — that's done when you run:

bash
CopyEdit

```
docker run -p 9090:8080 your-image-name
```

-

---

dockerfile
CopyEdit

```
ENTRYPOINT ["java", "-jar", "demo-app.jar"]
```

- **This tells Docker what command to run when the container starts.**

It runs:

bash
CopyEdit

```
java -jar demo-app.jar
```

-
- **This starts your Spring Boot application inside the container.**

Summary:

This `Dockerfile` creates a container that:

1. **Uses Java 17 to run your app.**
2. **Copies your Spring Boot JAR to the container.**
3. **Starts the app using `java -jar`.**

Would you like me to help you build and run this image step-by-step?