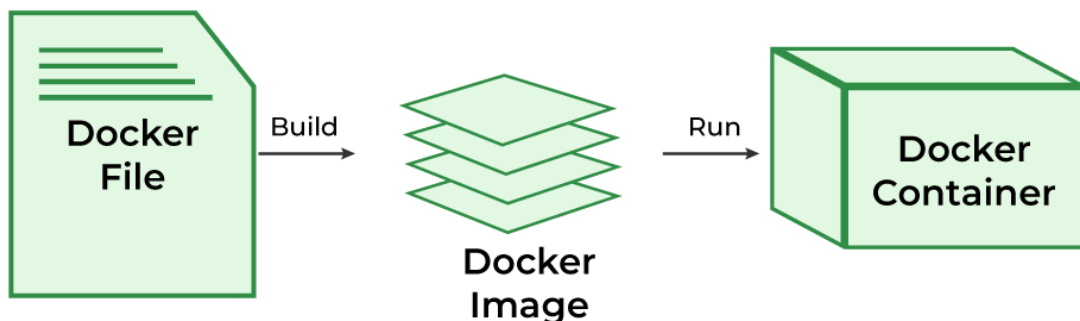


How to Write Dockerfile?

What is Dockerfile?

The operating system (OS) libraries and dependencies required to run the application source code which is not reliant on the underlying operating system (OS) included in the Dockerfile, which is a standardized, executable component. Programmers may design, distribute, launch, run, upgrade, and manage containers using the open-source platform Docker. Enterprise Edition (EE) and Community Edition (CE) of Docker are both available. The Enterprise Version is for businesses and IT teams working on mission-critical production applications, while the Community Edition is suitable for small teams just learning Docker.



The Dockerfile uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image. While creating your application, you should create a Dockerfile in order since the Docker daemon runs all of the instructions from top to bottom.

Dockerfile is the source code of the image

What is Docker Image?

An artifact with several layers and a lightweight, compact stand-alone executable package that contains all of the components required to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files is called a Docker image.

What is Docker Container?

A container is a runtime instance of an image. Containers make development and deployment more efficient since they contain all the dependencies and parameters needed for the application it runs completely isolated from the host environment.

Dockerfile commands/Instructions

1. FROM

- Represents the base image (OS), which is the command that is executed first before any other commands.

Syntax

```
FROM <ImageName>
```

Example: The base image will be ubuntu:19.04 Operating System.

```
FROM ubuntu:19.04
```

2. COPY

- The copy command is used to copy the file/folders to the image while building the image.

Syntax:

```
COPY <Source> <Destination>
```

Example: Copying the .war file to the Tomcat webapps directory

```
COPY target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war
```

3. ADD

- While creating the image, we can download files from distant HTTP/HTTPS destinations using the ADD command.

Syntax

```
ADD <URL>
```

Example: Try to download Jenkins using ADD command

```
ADD https://get.jenkins.io/war/2.397/jenkins.war
```

4. RUN

- Scripts and commands are run with the RUN instruction. The execution of RUN commands or instructions will take place while you create an image on top of the prior layers (Image).

Syntax

```
RUN < Command + ARGS>
```

Example

```
RUN touch file
```

5. CMD

- The main purpose of the CMD command is to start the process inside the container and it can be overridden.

Syntax

```
CMD [command + args]
```

Example: Starting [Jenkins](#)

```
CMD ["java","-jar", "Jenkins.war"]
```

6. ENTRYPOINT

- A container that will function as an executable is configured by ENTRYPOINT. When you start the Docker container, a command or script called ENTRYPOINT is executed.
- It can't be overridden. The only difference between [CMD and ENTRYPOINT](#) is CMD can be overridden and ENTRYPOINT can't.

Syntax

```
ENTRYPOINT [command + args]
```

Example: Executing the **echo command**.

```
ENTRYPOINT ["echo","Welcome to GFG"]
```

7. MAINTAINER

- By using the MAINTAINER command we can identify the author/owner of the Dockerfile and we can set our own author/owner for the image.

Syntax:

```
MAINTAINER <NAME>
```

Example: Setting the author for the image as a GFG author.

```
MAINTAINER GFG author
```

- To know more the syntax of Dockerfile refer to the [Syntax of Dockerfile](#).

Stages of Creating Docker Image from Dockerfile

The following are the stages of creating docker image form Dockerfile:

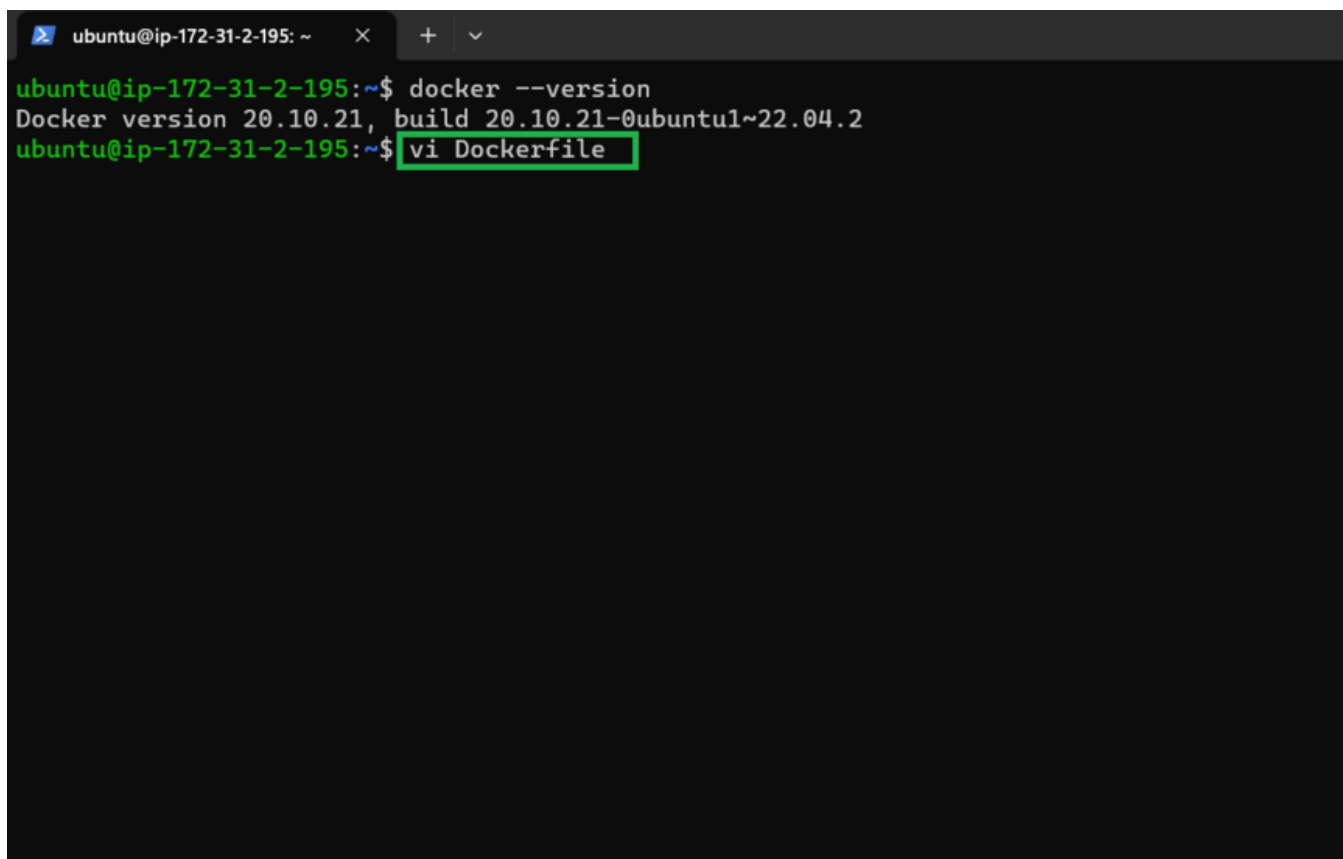
1. Create a file named Dockerfile.
2. Add instructions in Dockerfile.
3. Build Dockerfile to create an image.
4. Run the image to create a container.

Example 1: Steps To Create Dockerfile With Example (Jenkins)

In this example, we will write the Dockerfile for Jenkins and build an image by using Dockerfile which has been written for Jenkins and we will run it as a container.

Step 1: Open Docker and create a file with the name **Dockerfile**.

Step 2: Open the Dockerfile by using the vi editor and start writing the command that is required to build the Jenkins image.

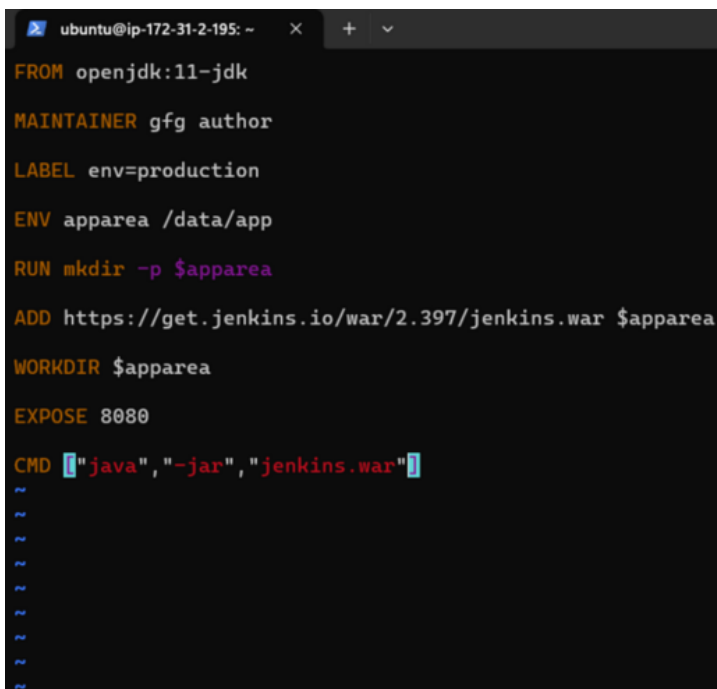
A terminal window with a dark background. The title bar shows 'ubuntu@ip-172-31-2-195: ~' and window controls. The terminal text shows the command 'docker --version' being executed, followed by the output 'Docker version 20.10.21, build 20.10.21-0ubuntu1~22.04.2'. Then, the command 'vi Dockerfile' is entered, and the text 'vi Dockerfile' is highlighted with a green rectangular box.

```
ubuntu@ip-172-31-2-195: ~  
ubuntu@ip-172-31-2-195:~$ docker --version  
Docker version 20.10.21, build 20.10.21-0ubuntu1~22.04.2  
ubuntu@ip-172-31-2-195:~$ vi Dockerfile
```

Dockerfile for Jenkins image

We used JDK as a base image because Jenkins's pre-requisite is JDK after that we added a command called **MAINTAINER** which indicates the author or owner of the docker file and we added the **ENV** variable where we set the path for the Jenkins and by using **RUN** command we are creating the path and by using **ADD** we are downloading the Jenkins and starting the **.war** file with the help of **CMD** command.

```
FROM openjdk:11-jdk
MAINTAINER GFG author
LABEL env=production
ENV apparea /data/app
RUN mkdir -p $apparea
ADD https://get.jenkins.io/war/2.397/jenkins.war $apparea
WORKDIR $apparea
EXPOSE 8080
CMD ["java","-jar","jenkins.war"]
```

A terminal window with a dark background and light-colored text. The terminal shows the same Dockerfile content as the first block. The prompt is 'ubuntu@ip-172-31-2-195: ~'. The text is color-coded: 'FROM' is orange, 'MAINTAINER' is orange, 'LABEL' is orange, 'ENV' is orange, 'RUN' is orange, 'ADD' is orange, 'WORKDIR' is orange, 'EXPOSE' is orange, and 'CMD' is orange. The values are in various colors: 'openjdk:11-jdk' is white, 'gfg author' is white, 'env=production' is white, '/data/app' is white, '-p \$apparea' is white, 'https://get.jenkins.io/war/2.397/jenkins.war \$apparea' is white, '\$apparea' is white, '8080' is white, and the array elements in 'CMD' are white, red, and white respectively. There are several tilde characters at the bottom of the terminal.

```
ubuntu@ip-172-31-2-195: ~  
FROM openjdk:11-jdk  
MAINTAINER gfg author  
LABEL env=production  
ENV apparea /data/app  
RUN mkdir -p $apparea  
ADD https://get.jenkins.io/war/2.397/jenkins.war $apparea  
WORKDIR $apparea  
EXPOSE 8080  
CMD ["java","-jar","jenkins.war"]  
~  
~  
~  
~  
~  
~  
~
```

Step 3: Build the image by using the below command with the help of Dockerfile and give the necessary tags. and the dot(.) represents the current directory which is a path for Dockerfile.

```
docker build -t jenkins:1 .
```

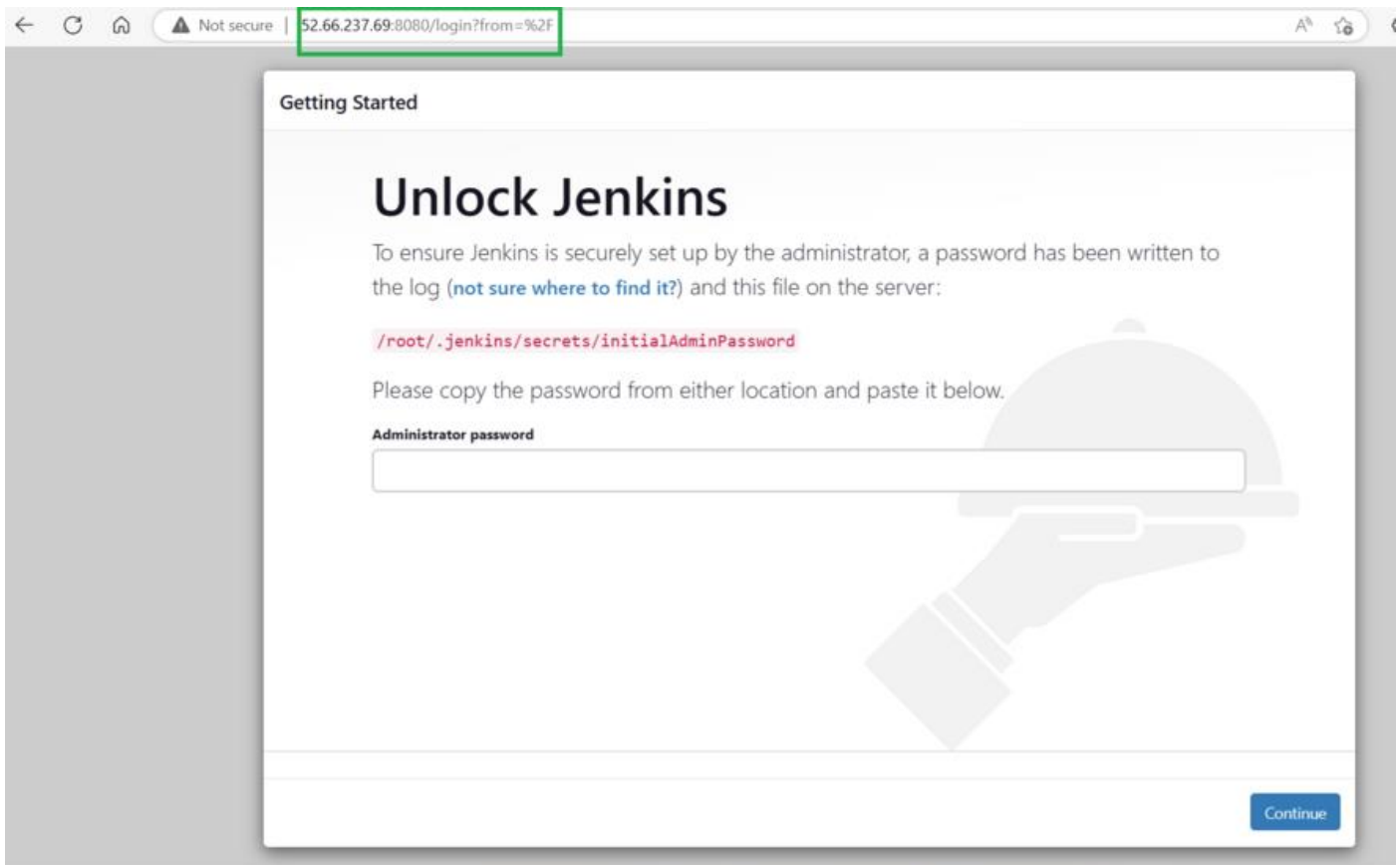
```
ubuntu@ip-172-31-2-195: ~$ docker build -t jenkins:1 .
Sending build context to Docker daemon 14.85MB
Step 1/9 : FROM openjdk:11-jdk
11-jdk: Pulling from library/openjdk
001c52e26ad5: Pull complete
d9d4b9b6e964: Pull complete
2068746827ec: Pull complete
9daef329d350: Pull complete
d85151f15b66: Pull complete
66223a710990: Pull complete
db38d58ec8ab: Pull complete
Digest: sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9c9db7c21ab
Status: Downloaded newer image for openjdk:11-jdk
--> 47a932d998b7
Step 2/9 : MAINTAINER gfg author
--> Running in 3697993426b5
Removing intermediate container 3697993426b5
--> cdbf26250cc3
Step 3/9 : LABEL env=production
--> Running in c2ca5c0ea41f
Removing intermediate container c2ca5c0ea41f
--> 82d694e061f6
Step 4/9 : ENV apparea /data/app
--> Running in 2a05388b20ee
Removing intermediate container 2a05388b20ee
--> 52c0895b482e
Step 5/9 : RUN mkdir -p $apparea
--> Running in d769bdf794fd
Removing intermediate container d769bdf794fd
--> 2eec8f5023dd
Step 6/9 : ADD https://get.jenkins.io/war/2.397/jenkins.war $apparea
Downloading [=====] 98.37MB/98.37MB
--> e97dc8e3bfe4
Step 7/9 : WORKDIR $apparea
--> Running in 4460d4247b7f
Removing intermediate container 4460d4247b7f
--> 8220175c3192
Step 8/9 : EXPOSE 8080
--> Running in 1b258383e8e9
Removing intermediate container 1b258383e8e9
--> 153a8ffe5032
Step 9/9 : CMD ["java","-jar","jenkins.war"]
--> Running in 337bc640ee17
Removing intermediate container 337bc640ee17
--> 72d820d68004
Successfully built 72d820d68004
Successfully tagged jenkins:1
```

Step 4: Run the container with the help image ID or tag of the image by using the below command.

docker run -d -p 8080:8080 <Imagetag/ID>

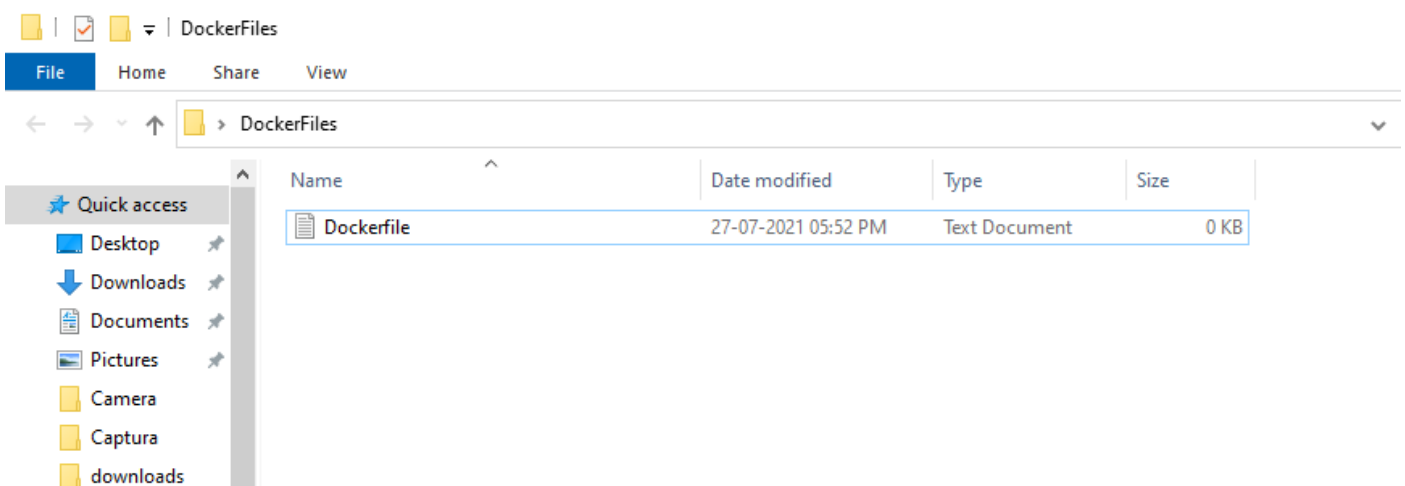
```
ubuntu@ip-172-31-2-195: ~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
jenkins        1         72d820d68004   46 seconds ago 753MB
openjdk        11-jdk    47a932d998b7   8 months ago  654MB
ubuntu@ip-172-31-2-195: ~$ docker run -d -p 8080:8080 72d820d68004
1ad19fb97038644c20e77980a5449aab35c96cf54a5f703ce3556fe732431c65
ubuntu@ip-172-31-2-195: ~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
1ad19fb97038   72d820d68004   "java -jar jenkins.w..." About a minute ago Up About a minute 0.0.0.0:8080->8080/tcp, :::80
80->8080/tcp    nifty_cerf
```

Step 5: Accesses the application ([Jenkins](#)) from the internet with the help of host port and hostIP (HostIP: Port)



Example 2: Steps To Create Dockerfile

Step 1: Create a file name called “**Dockerfile**”. By default when you run the docker build commands docker searches for a file named Dockerfile. However, it is not compulsory, you can also give some different names, and then you can tell the docker that this particular file is local but for now we will go with the Dockerfile.



Step 2: The very first instruction that a docker file starts with is FROM. Here you have to give a base image. So for example, if you want to get a base image from Ubuntu we will use FROM Ubuntu.

FROM ubuntu

- Then the other instruction is you must give a **MAINTAINER**. This is optional but it's a best practice that you give the maintainer of this image so that it is very easy to find out who is the maintainer, and you can give your name and email as well.
- And if you want you can just give the email as well without giving the name. But here we are giving the entire thing.

MAINTAINER YOUR_NAME <YOUR_EMAIL_ID>

- Next, we want to run something so we will say run any command we can use **RUN** and add the command that you need to run.

RUN apt-get update

- And if you want to run something on the command line during container creation you can give **CMD** and inside square brackets, and we add the command. Here it is as shown below:

CMD ["echo", "Hello People!"]

- At this point the file will have the following commands:

FROM ubuntu

MAINTAINER YOUR_NAME <YOUR_EMAIL_ID>

RUN apt-get update

CMD ["echo", "Hello People!"]

Step 3: Now we have to build the image so here are the commands you can use:

docker build /<FILE_LOCATION>

(Or)

docker build . -f Dockerfile.txt

- It says docker build and you have to give the location of your docker file. This will start building the image.

```
C:\Users\Geeks\Desktop\DockerFiles>docker build . -f Dockerfile.txt
[+] Building 19.9s (4/6)
=> [internal] load build definition from Dockerfile.txt
=> => transferring dockerfile: 140B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [auth] library/ubuntu:pull token for registry-1.docker.io
=> [1/2] FROM docker.io/library/ubuntu@sha256:82becede498899ec668628e7cb0ad87b6e1c371cb8a1e597d83a47fac21d6af3
=> => resolve docker.io/library/ubuntu@sha256:82becede498899ec668628e7cb0ad87b6e1c371cb8a1e597d83a47fac21d6af3
=> => sha256:82becede498899ec668628e7cb0ad87b6e1c371cb8a1e597d83a47fac21d6af3 1.42kB / 1.42kB
=> => sha256:1e48201ccc2ab83afc435394b3bf70af0fa0055215c1e26a5da9b50a1ae367c9 529B / 529B
=> => sha256:1318b700e415001198d1bf66d260b07f67ca8a552b61b0da02b3832c778f221b 1.46kB / 1.46kB
=> => sha256:16ec32c2132b43494832a05f2b02f7a822479f8250c173d0ab27b3de78b2f058 28.57MB / 28.57MB
```

- Command to list the images

docker image ls / docker images

```
C:\Users\Geeks\Desktop\DockerFiles>docker images
REPOSITORY          TAG          IMAGE ID      CREATED        SIZE
<none>              <none>      837a3fba860d  3 minutes ago 102MB
geeksforgeeks/docker101tutorial  latest      0b4e2b7426bd  3 hours ago   28.2MB
docker101tutorial    latest      0b4e2b7426bd  3 hours ago   28.2MB
alpine/git           latest      b8f176fa3f0d  2 months ago  25.1MB

C:\Users\Geeks\Desktop\DockerFiles>
```

Benefits of Dockerfile

The following are the benefits of Dockerfile:

- **Consistency and Reproducibility:** Dockerfile ensures that environment setups and dependencies are consistently facilitated across different setups minimizing the host environment dependent issues.
- **Version Control:** Docker files can be used for versioning along with your source code, It helps in tracking the changes and rollbacks.
- **Automation:** It provides the automation with the process of building, configuring, and deploying the applications with reducing the manual intervention and errors.

Best Practices for writing Dockerfile

The following are the best practices for writing Dockerfile:

- **Use official base Images:** Try on using the official base images for ensuring reliability, security and compatibility.
- **Minimize Layers:** Try on minimizing the layers by combining the common commands using && option and with using multistage builds.
- **Leverage Caching:** By ordering the instructions from least to most frequently changing one we can maximize the cache layering and speed up the builds.
- **Keep it Clean:** Ensure to remove the unnecessary files and use the .dockerignore to exclude the files and directories that are not needed in the image.

Trouble Shooting of Dockerfile Issues

The following are the some of the trouble shooting of Dockerfile Issues:

- **Check Build Logs:** Review the build logs for identifying the error through error messages and with log details. It helps in with providing the valuable information.

- **Validate Syntax and Instructions:** It helps in ensuring the docker file syntax and the instructions are in the proper order, It helps in addressing the common issues including the missing commands or in correct parameters.
- **Optimize Layer Caching:** Try to verify whether the caching is using effectively. Through reordering the instructions, we can reduce the changes in frequently modified layers to speed up the build process.
- **Dependency Management:** Through ensuring all the dependencies correctly placing as accessible, we can avoid the build failures.