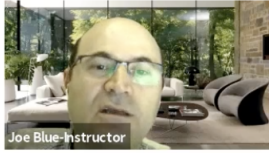
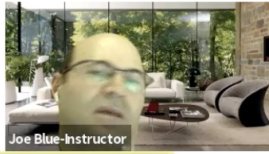


Docker Image

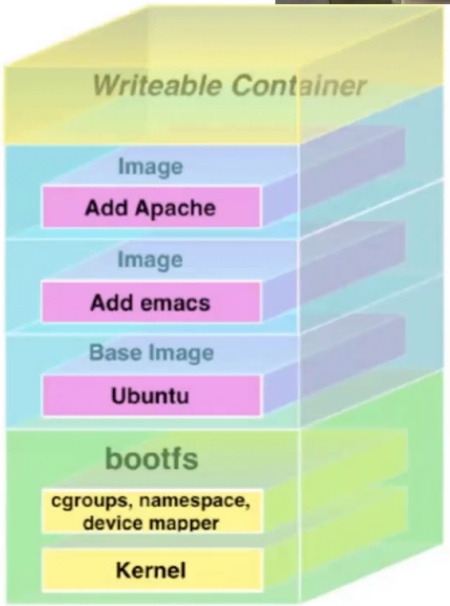
Table of Contents

- What is a Docker image?
- Container filesystem
- Dockerfile
- Dockerfile Instructions
- docker image Commands

What is a Docker image?



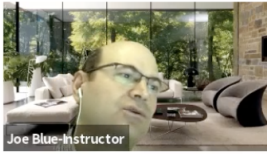
- An image is a read-only template with instructions for creating a Docker container.
- Often, an image is based on another image, with some additional customization.
- For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.



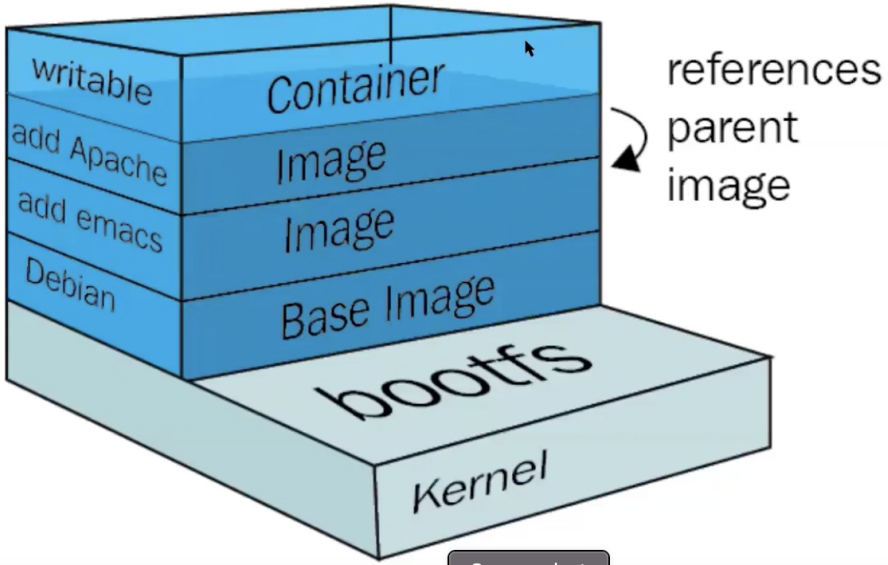
Container filesystem

- The **container filesystem**, used for every Docker image, is represented as a list of **read-only layers** stacked on top of each other.
- These layers eventually form a base root filesystem for a container. In order to make it happen, different storage drivers are being used.
- All the changes to the filesystem of a running container are done to the top level image layer of a container. This layer is called a **Container layer**.

Container filesystem

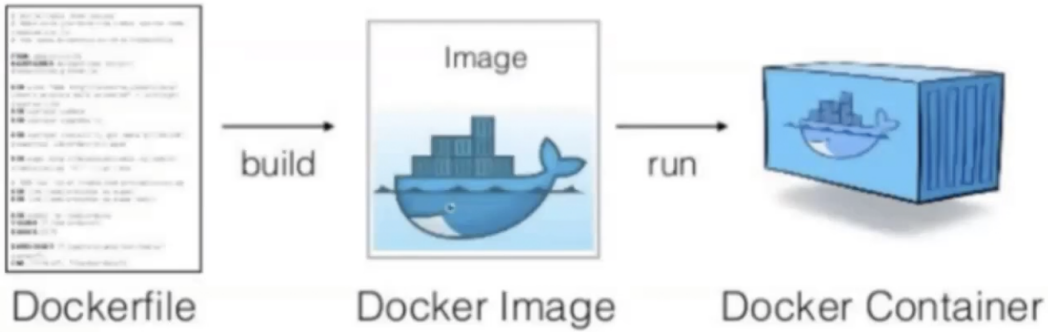


- Several containers may share access to the same underlying level of a Docker image, but write the changes locally and uniquely to each other.



Dockerfile

- A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.
- Using **docker build** users can create an automated build that executes several command-line instructions in succession.



Dockerfile

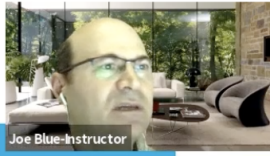

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code
ENTRYPOINT FLASK_APP=/opt/source-code/app.py
flask run
```

1. Start from a base OS or another image
2. Install all dependencies
3. Copy source code
4. Specify Entrypoint

Dockerfile Instructions

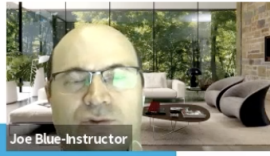
Instruction	Description
FROM	The FROM instruction initializes a new build stage and sets the *Base Image* for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.
RUN	The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.
CMD	The main purpose of a CMD is to provide defaults for an executing container. There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.
EXPOSE	The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

Dockerfile Instructions



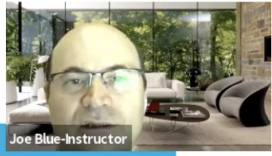
Instruction	Description
ENV	The ENV instruction sets the environment variable <key> to the value <value>.
ADD	The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.
COPY	The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
ENTRYPOINT	An ENTRYPOINT allows you to configure a container that will run as an executable.
VOLUME	The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

Dockerfile Instructions



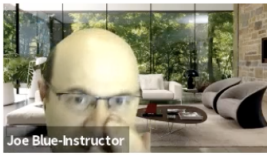
Instruction	Description
WORKDIR	The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile. If the WORKDIR doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.
ARG	The ARG instruction defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag. If a user specifies a build argument that was not defined in the Dockerfile, the build outputs a warning.
HEALTHCHECK	The HEALTHCHECK instruction tells Docker how to test a container to check that it is still working. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

Dockerfile Instructions



Instruction	Description
SHELL	The SHELL instruction allows the default shell used for the *shell* form of commands to be overridden. The default shell on Linux is ["/bin/sh", "-c"], and on Windows is ["cmd", "/S", "/C"]. The SHELL instruction *must* be written in JSON form in a Dockerfile.
ONBUILD	The ONBUILD instruction adds to the image a *trigger* instruction to be executed at a later time, when the image is used as the base for another build. The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile. Any build instruction can be registered as a trigger.
USER	The USER instruction sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.

Dockerfile Instructions



CMD

The `CMD` instruction has three forms:

- `CMD ["executable","param1","param2"]` (*exec* form, this is the preferred form)
- `CMD ["param1","param2"]` (as *default parameters to ENTRYPOINT*)
- `CMD command param1 param2` (*shell* form)

There can only be one `CMD` instruction in a `Dockerfile`. If you list more than one `CMD` then only the last `CMD` will take effect.

The main purpose of a `CMD` is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an `ENTRYPOINT` instruction as well.

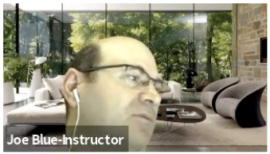
If `CMD` is used to provide default arguments for the `ENTRYPOINT` instruction, both the `CMD` and `ENTRYPOINT` instructions should be specified with the JSON array format.

```
FROM ubuntu
CMD echo "This is a test." | wc
```

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

Screenshot

Dockerfile Instructions



ENTRYPOINT

ENTRYPOINT has two forms:

The *exec* form, which is the preferred form:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

The *shell* form:

```
ENTRYPOINT command param1 param2
```

An `ENTRYPOINT` allows you to configure a container that will run as an executable.

Exec form ENTRYPOINT example

You can use the *exec* form of `ENTRYPOINT` to set fairly stable default commands and arguments and then use either form of `CMD` to set additional defaults that are more likely to be changed.

```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
CMD ["-c"]
```

docker image Commands

Command	Description
docker image build	Build an image from a Dockerfile
docker image history	Show the history of an image
docker image inspect	Display detailed information on one or more images
docker image ls	List images
docker image prune	Remove unused images
docker image pull	Pull an image or a repository from a registry
docker image push	Push an image or a repository to a registry
docker image rm	Remove one or more images
docker image tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE