# Materials to learn from

* High level overview - [www.youtube.com](http://www.youtube.com/)
* More detailed tutorials:
  + [www.youtube.com](https://www.youtube.com/watch?v=pg19Z8LL06w)
  + [docs.docker.com](https://docs.docker.com/get-started/get-docker/)

# Image

A Docker image is a template used for running containers. An image contains:

* OS layer - only OS files, not kernel. For example Ubuntu or Alpine.
* Files
* Dependencies (installed different tools / packages)
* Environment variables
* Instructions for starting processes (like a bash script in the CMD or ENTRYPOINT inscructions in a Dockerfile)

# Container

A container is a running instance of an image. In a container we have running processes specified in an image (started for example by a bash script specified in the CMD or ENTRYPOINT inscructions in a Dockerfile).

We can have multiple containers running using the same image.

Container adds additional writable layer to an image. If container modifies files it only affects the writable layer, not the image.

What a container and a host machine have separate:

* Filesystem (including Operating System files)
* Processes

What they have common:

* Kernel

# Layers

A layer contains files which represents filesystem changes. A layer is a set of instructions for Docker about how to change an image’s filesystem.

Each instruction in a Dockerfile creates a layer.

If we want to add or modify a file in an image, then that file will appear in a layer.

If we want to remove a file from an image, then in a layer we will have a special whiteout file:

* .wh.filename

That tells Docker that this file needs to be removed from the filesystem in an image.

Those layers are applied one be one to create a final image. Each layer applies changes in the filesystem.

Each layer is saved on a computer.

For example if we have a Dockerfile like this:

A black background with white text

AI-generated content may be incorrect.

Then:

* The first layer will contain only a file1 (which we copied to the image from the host)
* The second layer will contain a whiteout file .wh.file1 which tells Docker that this file should be removed from the filesystem
* The third layer will contain only the file2

After building an image using this Dockerfile, Docker will apply each layer one by one and that will cause:

* Adding the file1 to the image
* Removing the file1 from the image
* Adding the file2 to the image

## Layers caching

Layers are cached. That means that if we build an image multiple times using the same Dockerfile, then we are gonna use saved layers from previous builds, if we didn’t change an instruction in the Dockerfile.

## Layers sharing

Layers can be shared across images. If we build mutliple images using the same base layer, for example containing Ubuntu, then we are not duplicating that layer. The same layer will be used for every image.

# Dockerfile

Dockerfile is a file which contains instructions which create an image.

## Instructions

### RUN

The RUN instruction is used in order to execute a command in container’s terminal.

### USER

We can use the USER instruction in order to change a user which will be executing a command used in the RUN instruction.

### SHELL

By default Docker run commands (from instructions like RUN, CMD, ENTRYPOINT) using sh (/bin/sh). Using the SHELL instruction we can change it. We can specify there which shell to use, for example this changes shell into bash:

SHELL [“/bin/bash”, “-c”]

### COPY

Copy a file from a host to the created image. If has the following format:

* COPY local\_path container\_path

Where:

* **Local\_path** – Is a path on a local host where is located a file to copy. This path needs to be relative to the build context directory.
* **Container\_path –** Is a path in a container where to save a file

### CMD

Specify what bash command will be executed when starting a container.

### ENTRYPOINT

Specify what bash script will be executed when starting a container.

# Build context directory

Build context directory is a directory which contains the Dockerfile and all the other files which will be used in the Dockerfile (which we will copy into an image using the COPY or ADD instructions).

We provide a path to this directory in the ‘docker build’ command.

What path we will need to specify in the COPY instruction depends on this path.

# Bind Mounting

We can mount a folder on host to a folder in a container. That means that all the files which we create in the host folder will be reflected in the container one and vice versa.

This is useful if we want to upload files to a container and have the same files even after restarting a container.

**Folder permissions:**

If we run an app in a container, and that app wants to perform action (create, delete) on a file on the host which is mounted into the container, then user from the container which is running this app will be verified for permission.

The best practice is to create a user with a specific UID and GID in a container and grant permissions to the folder in container and on host for that UID and GID (just using a username is not enough, we need to use UID and GID when defining permissions).

If we create a user in a container without specifying UID and GID then those values are generated automatically. It is better to specify those values in that case so we know what are those values.