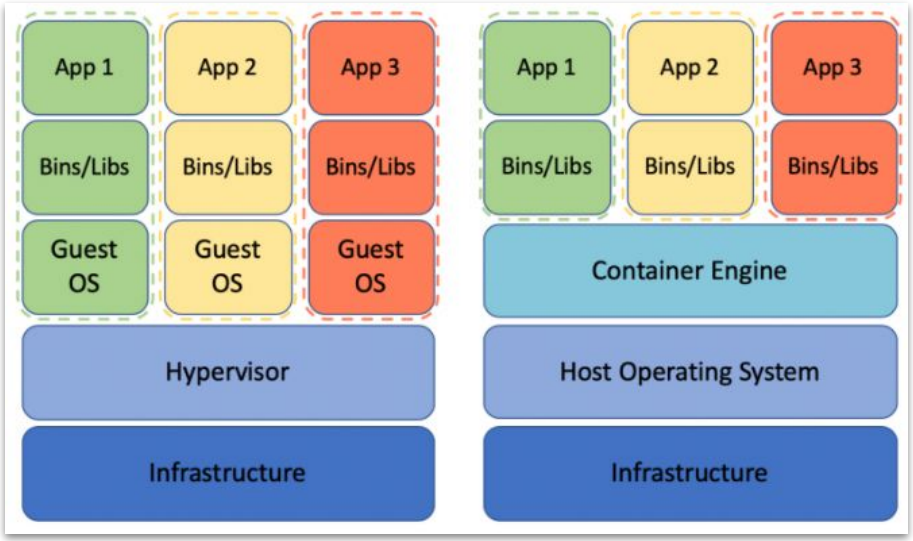
System and Network Engineering - Lecture 12

$ Containerization - Docker

Main Concept

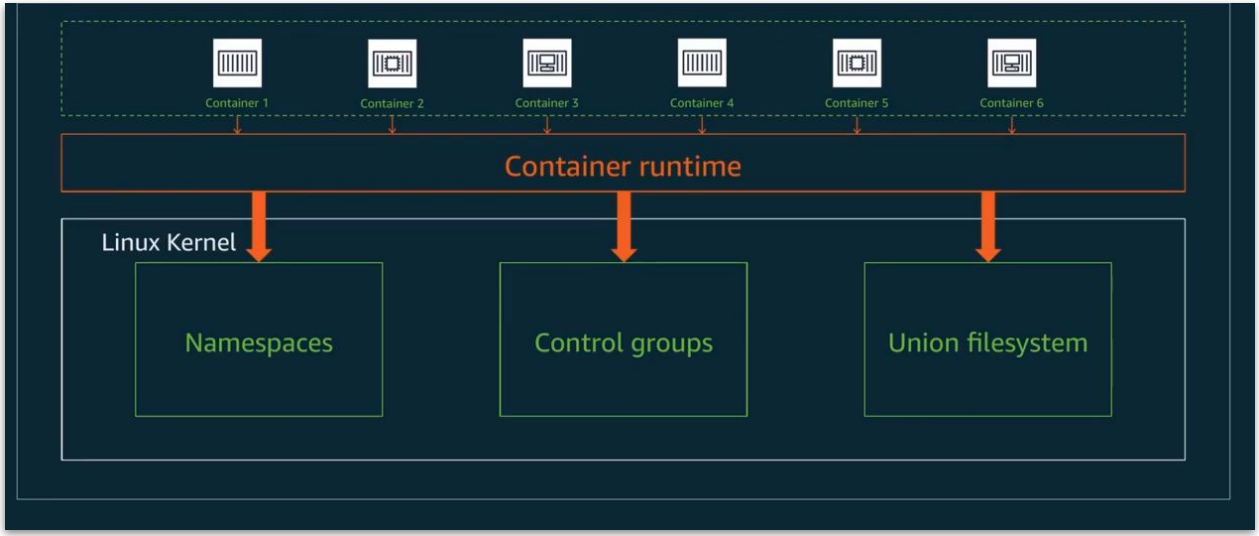
- Containerization is an opportunity to run isolated processes in a protected environment.
- A container is a sandboxed application with all of its dependencies included, so the application runs quickly and reliably from one computing environment to another.



Virtualization vs Containerization

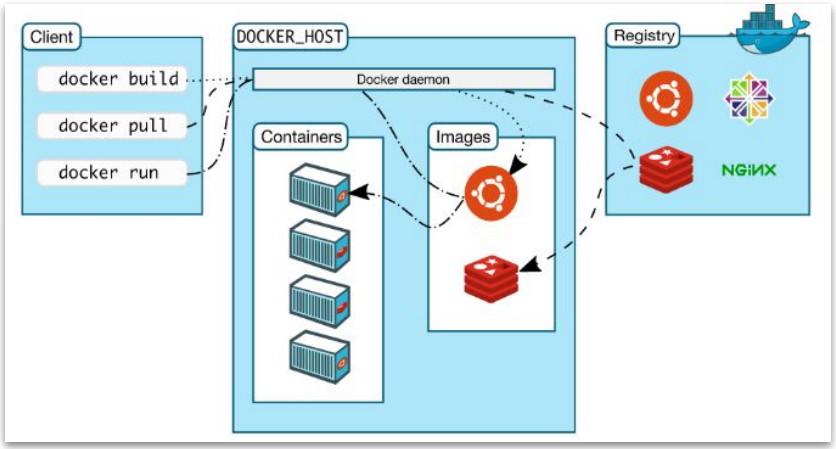
Virtualization	Containerization
Hardware level virtualization	Operating system virtualization
Abstracts OS from hardware	Abstracts Application from OS
Heavyweight	Lightweight
Minutes to startup	Few seconds to startup
More secure and isolated	Less secure and isolated

Containerisation

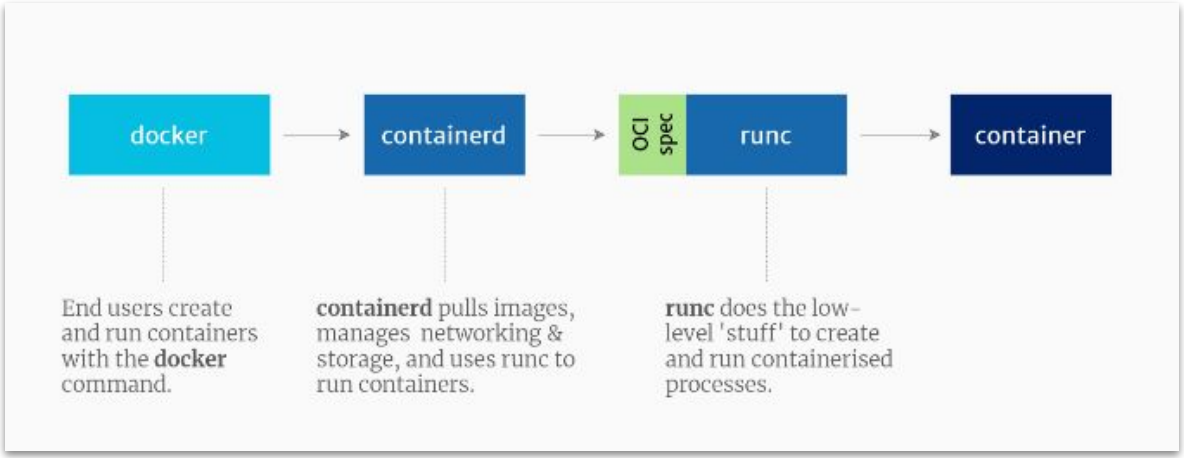


Docker components

- **Docker daemon (dockerd)** - listens for docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.
- **Docker client (docker)** - is the primary way when users interact with Docker.
- **Docker registries** - stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on DockerHub by default.
- **Images** - 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.
- **Containers** - is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.

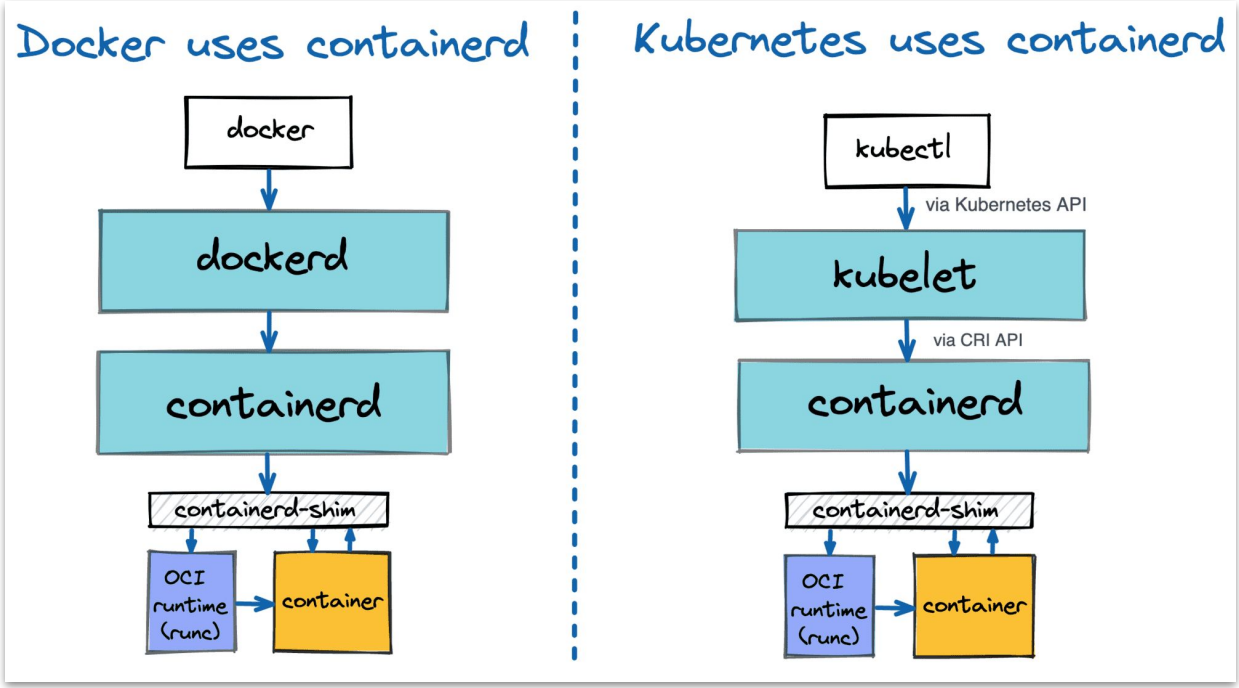


Docker to container



- **Containerd** - is a high-level container runtime that came from Docker, and implements the CRI spec. It pulls images from registries, manages them and then hands over to a lower-level runtime, which actually creates and runs the container processes (to runc).
- **Runc** - is an OCI-compatible container runtime. It implements the OCI specification and runs the container processes.

Docker components



Container

Container - runnable instance of an image

- ❑ `$docker run -it ubuntu /bin/bash` # run container interactively; `-i` - *keep open stdin*, `-t` - *allocate a pseudo-TTY*
- ❑ `$docker run -d nginx` # run container as an application in the background
- ❑ `$docker ps [-a]` # list running [or all] containers including the stopped ones
- ❑ `$docker inspect` # get info about container

manipulate container lifecycle

- ❑ `$docker stop/start/restart/kill <ID>/<name>`

execute command [interactively]

- ❑ `$docker exec [-it] <ID>/<name> <command>`

```
saltanov@UbuntuPC:~$ docker ps -a
CONTAINER ID   IMAGE             COMMAND       CREATED              STATUS
   PORTS     NAMES
6aa6de39f32a   python:3.9-slim   "python3"     About a minute ago   Exited (1) 44 seconds ago
             awesome_engelbart
c450bc840880   python:3.9-slim   "python3"     2 minutes ago        Exited (1) About a minute ago
             fervent_satoshi
a1240d1c726f   python:3.9-slim   "python3"     2 minutes ago        Exited (1) 2 minutes ago
             cranky_solomon
172c17b2ec46   python:3.9-slim   "/bin/bash"   3 minutes ago        Exited (130) 3 minutes ago
             mystifying_gagarin
31c63ec2d431   python:3.9-slim   "python3"     3 minutes ago        Exited (0) 3 minutes ago
             practical_lehmann
b17b6378e37d   python:3.9-slim   "python3"     4 minutes ago        Exited (0) 4 minutes ago
             silly_swirles
dc0ba1304f5e   hello-world       "/bin/sh"     7 minutes ago        Created
             exciting_bouman
550ea856dc92   hello-world       "/bin/bash"   7 minutes ago        Created
             elegant_merkle
24e415e33cb0   hello-world       "/hello"      7 minutes ago        Exited (0) 7 minutes ago
             sharp_euler
6e0ab1ce5c80   hello-world       "/hello"      8 minutes ago        Exited (0) 8 minutes ago
             gracious_kirch
b4656ef734c1   hello-world       "/hello"      8 minutes ago        Exited (0) 8 minutes ago
             wonderful_stonebraker
saltanov@UbuntuPC:~$ docker rm $(docker ps -aq)
6aa6de39f32a
c450bc840880
a1240d1c726f
172c17b2ec46
31c63ec2d431
b17b6378e37d
dc0ba1304f5e
550ea856dc92
24e415e33cb0
6e0ab1ce5c80
b4656ef734c1
saltanov@UbuntuPC:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
saltanov@UbuntuPC:~$
```

Run container with options

Ref: https://docs.docker.com/engine/reference/run/

$docker run -d --rm --name webserver -p 8080:80 nginx

- `-d` # run as daemon
- `--rm` # remove container after its exit
- `--name` # specify future container name
- `-p <host_port>:<container_port>` # publish ports

Limit container resources: Ref: https://docs.docker.com/config/containers/resource_constraints/ - advanatage of usage the Linux cgroups

CPU:

- `--cpus=<max_amount_of_CPUs_available_for_the_container>`
- `--cpuset-cpus=<list_of_specific_CPUs_numbers>`

Memory:

- `--memory="<max_amount_of_memory_available_for_the_container>"`
- `--memory-reservation="<amount_of_memory>" # soft limit - activates when host busy and need to take some memory resource from this container to provide for the host, otherwise OOM Linux killer can start to destroy processes`

Use environment variables

- ```
  $docker run -d \
  -e <env1_name>=<env1_value> \
  -e <env2_name>=<env2_value> \
  <image_name>
  ```

- ```
  $docker run -d \
  --name some-postgres \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -v /custom/mount:/var/lib/postgresql/data \
  postgres
  ```

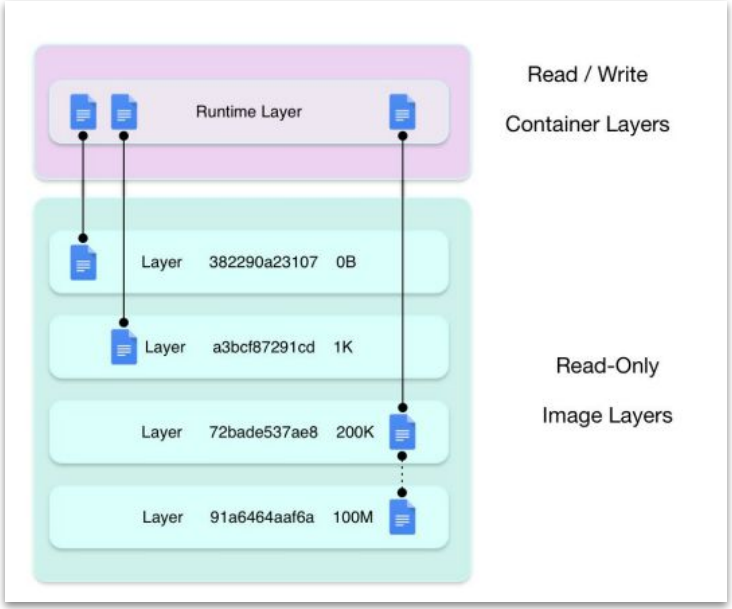
> **`PGDATA`**
>
> This optional variable can be used to define another location - like a subdirectory - for the database files. The default is `/var/lib/postgresql/data`. If the data volume you're using is a filesystem mountpoint (like with GCE persistent disks) or remote folder that cannot be chowned to the `postgres` user (like some NFS mounts), Postgres `initdb` recommends a subdirectory be created to contain the data.

Docker Images

An image - is a read-only template with instructions for creating a Docker container.

Consists of layers

- ❏ Each layer keeps changes delta (e. g. adds some package or modifies file to the previous one)
- ❏ Copy-on-Write (CoW) strategy



Docker Images: example

2 layers are created: one is the basis image and second is the test1 directory with the file.txt. Upper layer stores only the difference

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
RUN mkdir /test1 && echo "Hello" > /test1/file.txt
```

```
saltanov@linuxpc:~/temp$ docker image build -t testimage1 ./
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
284055322776: Pull complete
Digest: sha256:0fedbd5bd9fb72089c7bbca476949e10593cebed9b1fb9edf5b79dbbacddd7d6
Status: Downloaded newer image for ubuntu:18.04
 ---> 5a214d77f5d7
Step 2/2 : RUN mkdir /test1 && echo "Hello" > /test1/file.txt
 ---> Running in 731f1a8efb45
Removing intermediate container 731f1a8efb45
 ---> ae5da9297e0e
Successfully built ae5da9297e0e
Successfully tagged testimage1:latest
```

```
saltanov@linuxpc:/var/lib$ docker image ls
REPOSITORY     TAG       IMAGE ID       CREATED         SIZE
redis          latest    40c68ed3a4d2   5 weeks ago     113MB
ubuntu         latest    ba6acccedd29   2 months ago    72.8MB
hello-world    latest    d1165f221234   9 months ago    13.3kB
saltanov@linuxpc:/var/lib$ docker image ls
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
testimage1     latest    ae5da9297e0e   12 seconds ago   63.1MB
redis          latest    40c68ed3a4d2   5 weeks ago      113MB
ubuntu         latest    ba6acccedd29   2 months ago     72.8MB
ubuntu         18.04     5a214d77f5d7   2 months ago     63.1MB
hello-world    latest    d1165f221234   9 months ago     13.3kB
```

```
root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 36
drwx------ 2 root root 4096 дек 27 01:41 l
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 44
drwx------ 2 root root 4096 дек 27 01:43 l
drwx--x--- 4 root root 4096 дек 27 01:41 67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2
drwx--x--- 3 root root 4096 дек 27 01:41 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
```

```
root@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93
981bea255dc0309009701efc7f0eb38344d2/diff# ls
test1
```

Docker Images: example

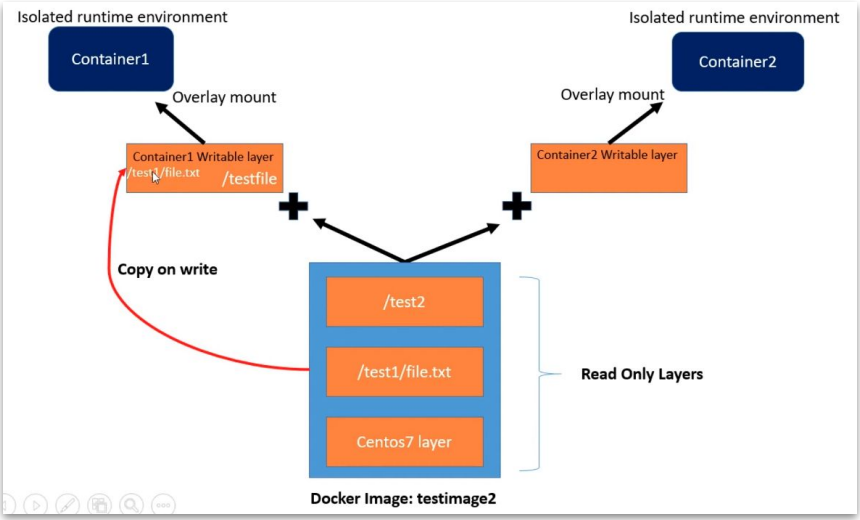
File **lower** contains the **link** for the lower layer (and file **link** in the lower layer contains the same string to reference to the upper layer)

```
root@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2# cat lower
l/POZLA6DCVR36LUXLNJNEKPL2JZroot@linuxpc:/var/lib/docker/overlay2/67b29b2c2c8c2d6a3b04fc4edd93981bea255dc030900970
```

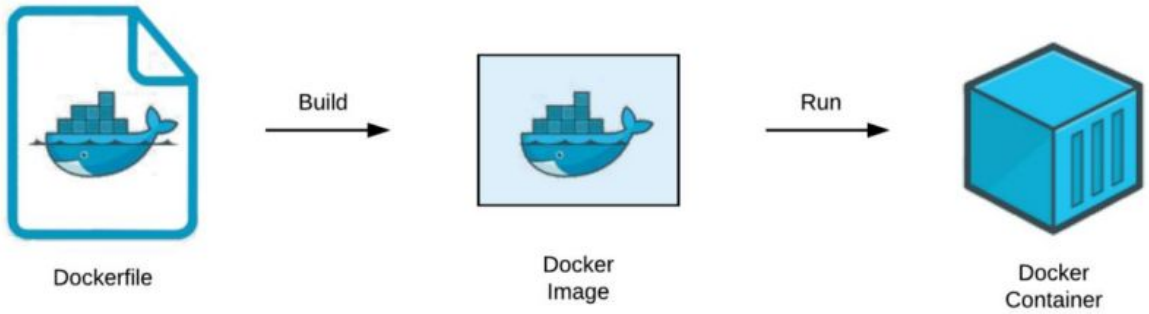
```
root@linuxpc:/var/lib/docker/overlay2# ls -tl
total 44
drwx------ 2 root root 4096 дек 27 01:43 l
drwx--x--- 4 root root 4096 дек 27 01:43 67b29b2c2c8c2d6a3b04fc4edd93981bea255dc0309009701efc7f0eb38344d2
drwx--x--- 3 root root 4096 дек 27 01:42 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3
drwx--x--- 4 root root 4096 дек  2 07:51 8e354e047da87ac59583345c7bf1a567ee83d47fe1645d9bbbe0e9ef9ab36814
drwx--x--- 4 root root 4096 дек  2 07:51 e457e0306d26e4123314700f08e352b67421879cda9bf68a14b4a045194acd32
drwx--x--- 4 root root 4096 дек  2 07:51 36420716cba4df5320ec70996a20d5320c75442e6bb1e38cd715562730b79caa
drwx--x--- 4 root root 4096 дек  2 07:51 59abcf5f57d9487ed5163d62c0102e038e5ede6dc7bfc195c86f52fedafc46af
drwx--x--- 4 root root 4096 дек  2 07:51 d069dc0460caafe47f6fc0a58f5a83ecb9763a75e8a1e057dfccf7061f643e84
drwx--x--- 3 root root 4096 дек  2 07:51 04154f77448cf13da51071c2a5b363caf92907541b1172bd9f60cf2711e3a2d4
drwx--x--- 3 root root 4096 ноя 20 01:00 795b8303720b409948618915bc52d5e3ea580a92b53f13cb2707b00eaffbb6fc
drwx-----x 3 root root 4096 авг 24 20:55 2e0122b1eadd400c40a5b6ddfcc2cc906f295bb8f3e933cf49c7e964f6652585
root@linuxpc:/var/lib/docker/overlay2# cd 3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3/
root@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3# ll
total 24
drwx--x---  3 root root  4096 дек 27 01:42 ./
drwx--x--- 13 root root 12288 дек 27 01:43 ../
-rw-------  1 root root     0 дек 27 01:43 committed
drwxr-xr-x 21 root root  4096 дек 27 01:42 diff/
-rw-r--r--  1 root root    26 дек 27 01:42 link
root@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c2882902df68b45cba3# cat link
POZLA6DCVR36LUXLNJNEKPL2JZroot@linuxpc:/var/lib/docker/overlay2/3974af62148a49cc98848dc963313fea18ca7ccb09555c288
```

Copy on Write operation

- Any new changes during runtime will be done only for the particular container within its writable layer, to make sure that underline layers are common and not change among the running containers
- If we want to change any file/directory from underlying layer, first file will be copied to writable layer and changed there
- Copy on write operations are heavy and can impact performance, so docker image should be carefully designed
- Deleted files are hidden, but still exist



Building Images



Dockerfile

Build images:

Ref: https://docs.docker.com/engine/reference/commandline/build

- docker build .
- docker build -t <image-name>[:<image-tag>] .
- docker build -f Dockerfile.dev .

Basic instructions:

define base image

- FROM <base-image-name>[:<base-image-tag>]

set metadata, e. g. author

- LABEL <label-name>="<label-value>"

execute some command

- RUN <command> or RUN ["executable", "arg1", "arg2"]

define metadata about used ports and for inter-container communications

- EXPOSE <port-number>

#define runtime environment or build arguments

- ENV <name>=<value> or ARG <name>[=<default-value>]

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev
linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Dockerfile instructions:

ADD and COPY:

COPY

- `COPY <src> <dst>`

ADD is the same but:

- can download remote files
- can extract (and decompress) locally-saved tarballs

NOTE: use COPY whenever possible

- that's because it's more transparent than ADD. COPY only supports the basic copying of local files into the container
- the best use for ADD is local tar file auto-extraction into the image, as in `ADD rootfs.tar.xz /.`
- If you have multiple Dockerfile steps that use different files from your context, COPY them individually, rather than all at once. This ensures that each step's build cache is only invalidated (forcing the step to be re-run) if the specifically required files change.

Dockerfile instructions:

CMD and ENTRYPOINT - both CMD and ENTRYPOINT instructions define what command gets executed when running a container.

There are few rules that describe their cooperation:

- Dockerfile should specify at least one of CMD or ENTRYPOINT commands
- ENTRYPOINT should be defined when using the container as an executable
- Default entrypoint is `/bin/sh -c`
- CMD should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container
- CMD will be overridden when running the container with alternative arguments
- ENTRYPOINT can be overwritten during the run-time with the `--entrypoint` option

The best use for ENTRYPOINT is to set the image's main command, allowing that image to be run as though it was that command (and then use CMD as the default flags).

```
ENTRYPOINT ["nginx"]
CMD ["--help"]
```

Dockerfile instructions:

CMD, Entrypoint and shell behavior depending on the syntaxis provided - shell or exec form **:**

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Useful tips:

- Keep images small:
  - Use right base image
  - Don't create unnecessary layers
- One container – one (main) process (unix signals in container only received for the PID1. If one app failed then it ll not be obvious once another process still exist and container continue to run)
- Optimize layers caching – order instructions from the least changed to the most frequently changes