Lecture 11: Docker I

DES 422 Web and Business Application Development

Setup docker on Windows

- https://docs.docker.com/docker-for-windows/install-windows-home/

- Requirement:

  Windows 10 Home, version at least 1903

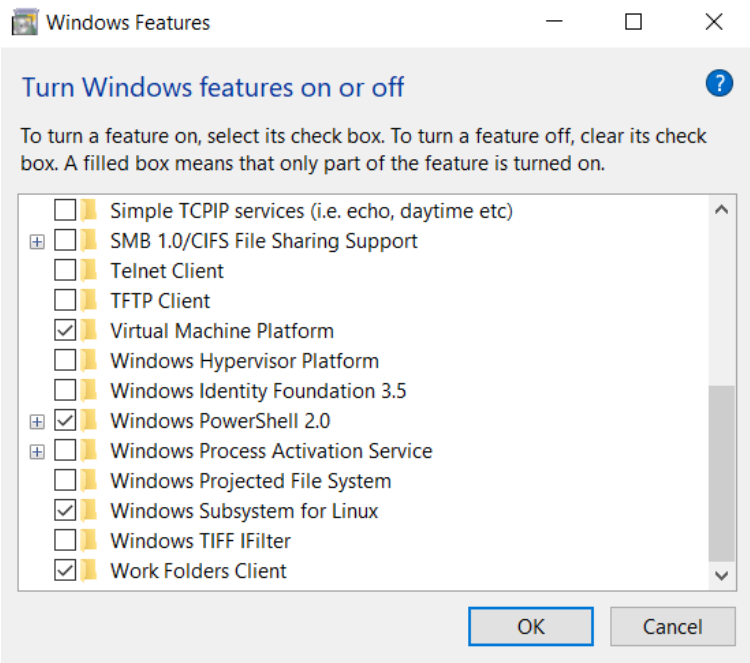
  win+r and type winver to check your Windows version

  Enable the WSL 2 feature on Windows

  https://docs.microsoft.com/en-us/windows/wsl/install-win10

Enable WSL2

- https://docs.microsoft.com/en-us/windows/wsl/install-win10
- check that WSL is enabled: start -> “Turn Windows features on or off” confirm that *Windows Subsystem for Linux* is checked



Install docker itself

- https://docs.docker.com/docker-for-windows/install-windows-hor

Install Docker Desktop on Windows Home

Estimated reading time: 5 minutes

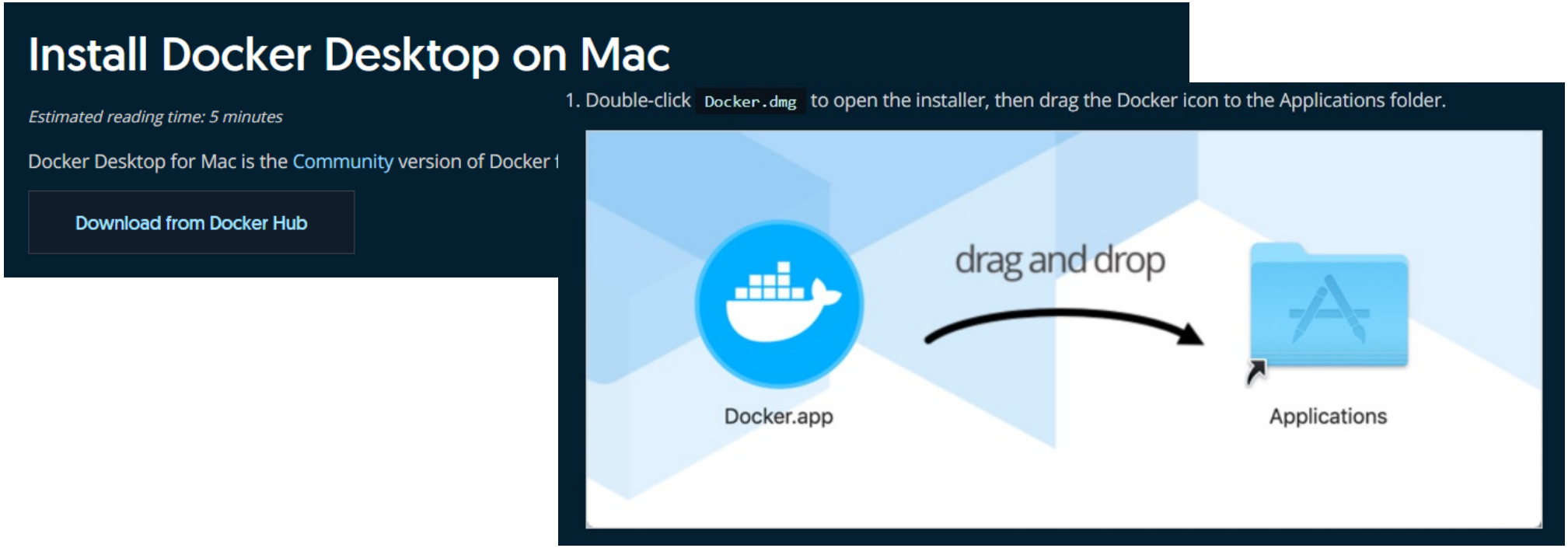
You can now install Docker Desktop on Windows Home machines using the WSL 2 backend. Docker Desktop on Windows Home is a full version of Docker Desktop for Linux container development.

This page contains information on installing Docker Desktop on Windows 10 Home. If you are looking for information about installing Docker Desktop on Windows 10 Pro, Enterprise, or Education, see Install Docker Desktop on Windows.

Download from Docker Hub

Setup docker on Mac

- https://docs.docker.com/desktop/install/mac-install/



Running your first container

- open cmd/powershell/terminal and run
- *docker run*

```
(base) PS C:\Users\acer> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

running nginx container

- *docker run -p 80:80 nginx*
- or
- *docker container run -p 80:80 nginx*
- ctrl+c to stop container
- on Windows: ctrl+c doesn't stop container. Must do docker stop <container_id>

Run container in the background

- *docker run -d -p 80:80 nginx*
- *d is for "detach"*

View logs

- docker logs <container_name>
- docker logs --tail n <container_name> (print only the last n lines)

List container

- docker ps (list running containers)
- docker ps -a (list all containers)
- docker container ls -a (new command format)

Remove container

- docker rm <container_name> (must stop container first)
- docker rm -f <container_name> (force remove)

Format of the run command

- docker run -p 80:80 --name <container_name> -d python:3 ls

change the default CMD

detach
(run in background)

port format
host:container

specify name for container

<image_name>:<tag>

- You can add the --rm option to remove the container immediately after it exits. Useful for running containers in interactive mode

Exercise – running multiple containers

1. start 2 containers: nginx, mysql in detach mode (name the containers the same as the image name)
2. nginx should forward port to 80 on host and mysql to 3306
3. on mysql, use -e to pass environment variable MYSQL_RANDOM_ROOT_PASSWORD=yes
4. use *docker container logs* on mysql to find the root password
5. stop and delete both containers with *docker stop* and *docker rm*

docker top, inspect and stats

- docker run -d --name nginx nginx
- docker top nginx
- docker inspect nginx
- docker stats nginx
- docker stats

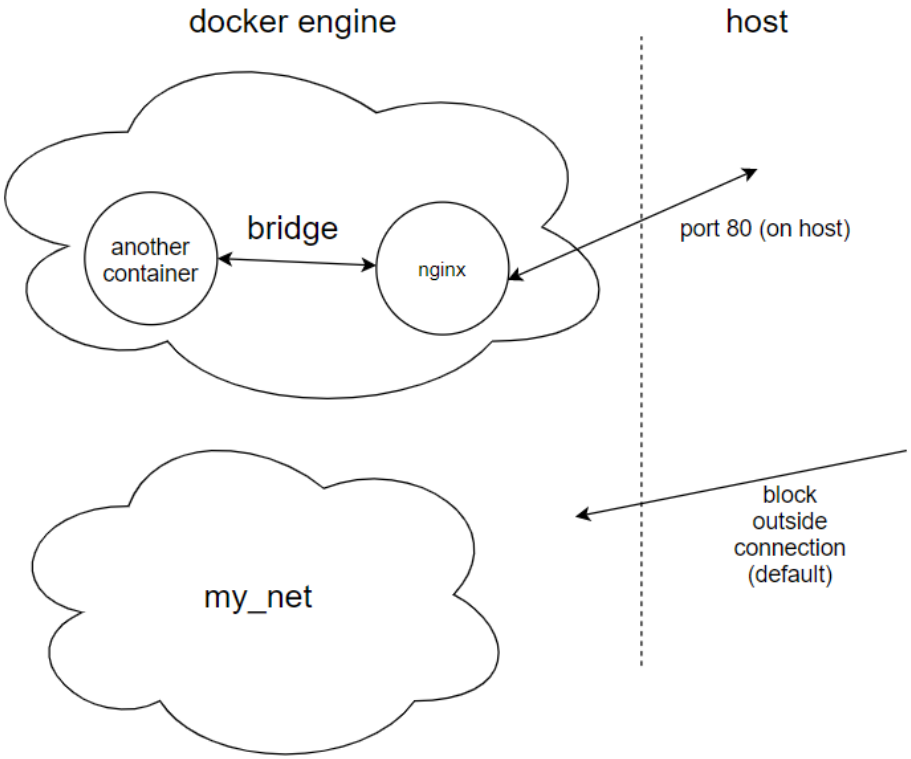
Getting “inside” a container

- docker exec -it <container_id>/bin/bash

Copy files host <-> container

- docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH
- docker cp [OPTIONS] SRC_PATH CONTAINER:DEST_PATH

Docker network



network commands

- docker network ls (list all networks)
- docker network inspect <network> (inspect a network)
- docker network create <network> (create a network)

E.g.

- docker network ls
- docker network inspect my_net
- docker network inspect bridge

```
(base) PS C:\Users\acer\Desktop\tmp> docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
72a94f67d147        bridge              bridge              local
628e52dc6ddd        docker_gwbridge     bridge              local
1f8093559e9f        host                host                local
vms2g4ffqdct        ingress             overlay             swarm
6b2a4d6676c9        my_net              bridge              local
c6ddae90b757        none                null                local
(base) PS C:\Users\acer\Desktop\tmp>
```

Connect container to a network

- docker network connect my_net nginx
- docker network inspect mynet
- docker inspect nginx

Connect to network when starting new container

- docker run --network <network> ……

DNS

- how can containers talk to each other?
- containers on the same network can find each other with their container name as DNS hostname
- we don’t need to remember (or even know) the IP address of each container

Exercise – ping a container from another

1. create a new network “my_net” (default network bridge doesn’t have DNS)
2. start two containers off the nginx image and add them to my_net (name them container1 and container2)
3. get inside container1
4. install *iputils-ping* package
5. ping container2
6. repeat 3-5 but the other way around (ping container2 from container 1)

Network Alias

- we can assign multiple container to the same (alias) DNS name
- E.g. a big website needs to run multiple containers (serving the same website) to keep up with demand
- *docker run -d --network my_net --name <name> --network-alias <alias> nginx*

Exercise - alias

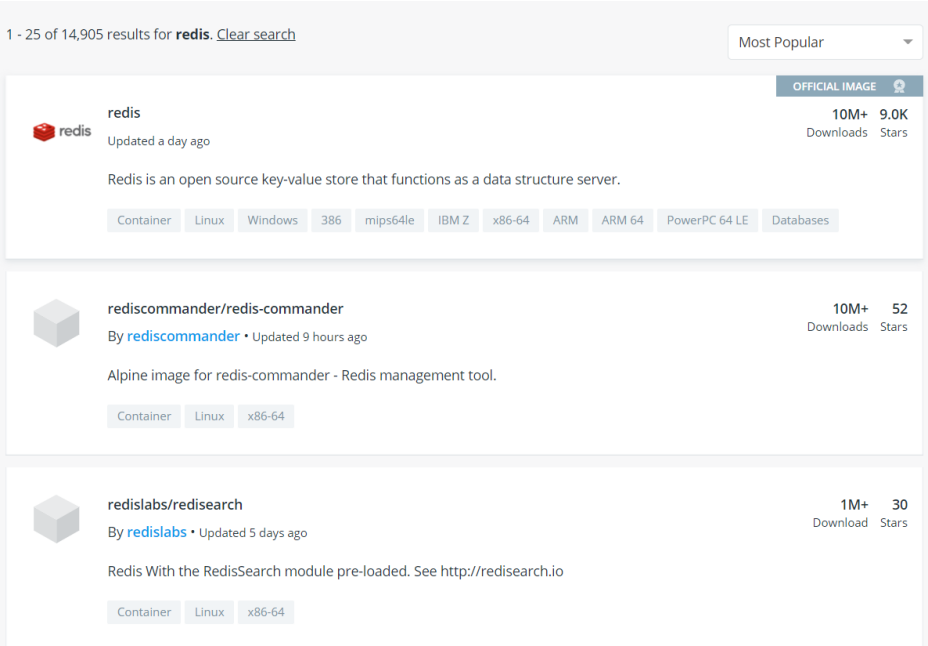
1. create a new network "my_net"
2. run two containers off the nginx image, have them join the my_net network
3. give each container the alias "web" using --network alias web
4. run another container off the alpine image, have it also join the my_net network
5. get inside the alpine container (alpine doesn't have bash, it has ash shell)
6. run *nslookup web* (inside the alpine container)
7. compare the output with *docker network inspect my_net*

Docker images commands

- *$ docker images*
- *$ docker pull <image_name>*
- *$ docker image rm <image_name>*

Docker Hub

- no / means official image
- everything else is <account_name>/<image>



Tags

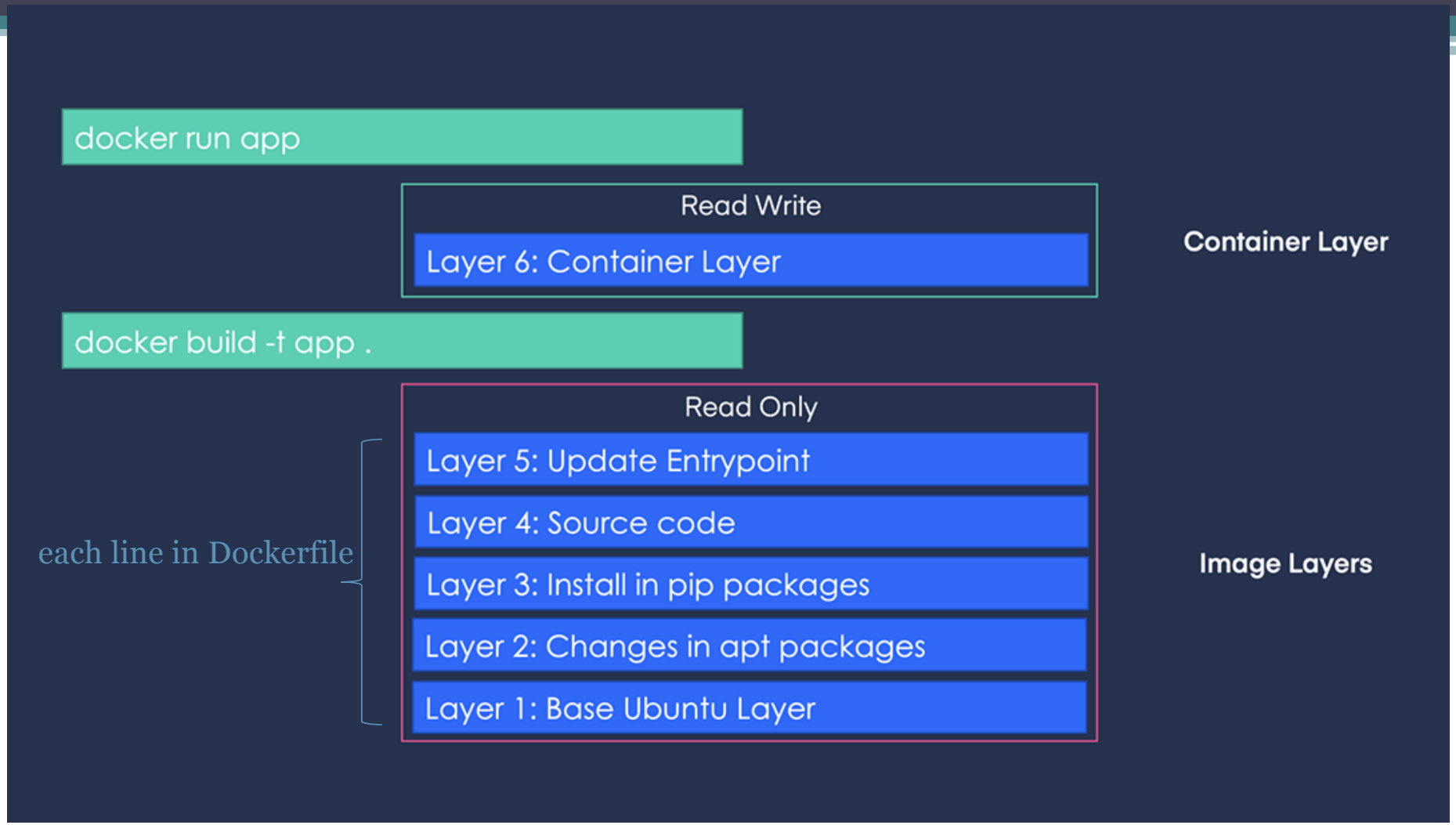
- one image can have multiple tags
- don’t tag your image as latest when shipping

Supported tags and respective `Dockerfile` links

- `1.19.6`, `mainline`, `1`, `1.19`, `latest`
- `1.19.6-perl`, `mainline-perl`, `1-perl`, `1.19-perl`, `perl`
- `1.19.6-alpine`, `mainline-alpine`, `1-alpine`, `1.19-alpine`, `alpine`
- `1.19.6-alpine-perl`, `mainline-alpine-perl`, `1-alpine-perl`, `1.19-alpine-perl`, `alpine-perl`
- `1.18.0`, `stable`, `1.18`
- `1.18.0-perl`, `stable-perl`, `1.18-perl`
- `1.18.0-alpine`, `stable-alpine`, `1.18-alpine`
- `1.18.0-alpine-perl`, `stable-alpine-perl`, `1.18-alpine-perl`

What is an image

- https://github.com/moby/moby/blob/master/image/spec/v1.md
- “An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime.”
- No OS.

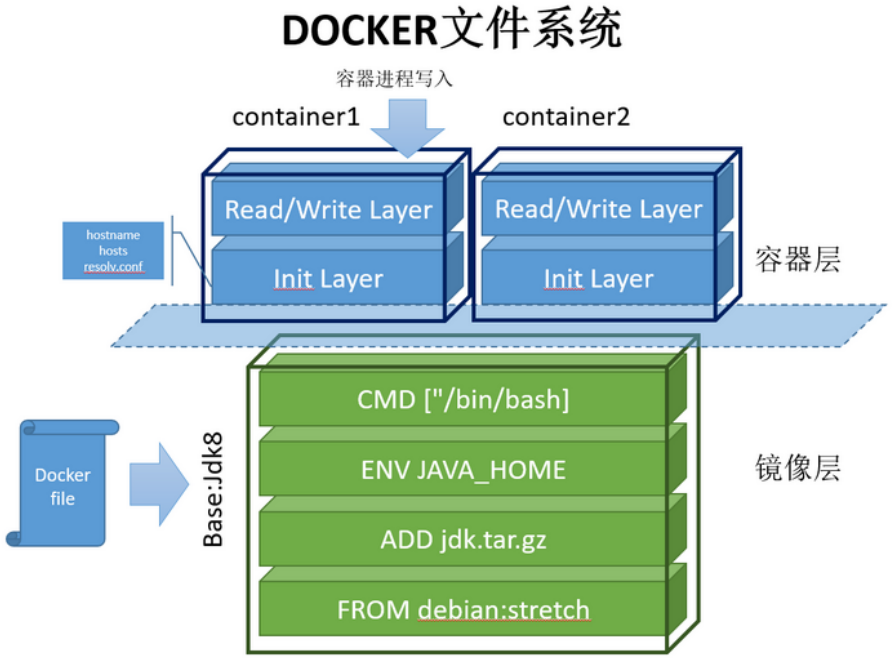


https://towardsdatascience.com/docker-storage-598e385f4efe

Inspect and history

- *$ docker image inspect <image_name>*
- *$ docker history <image_name>*

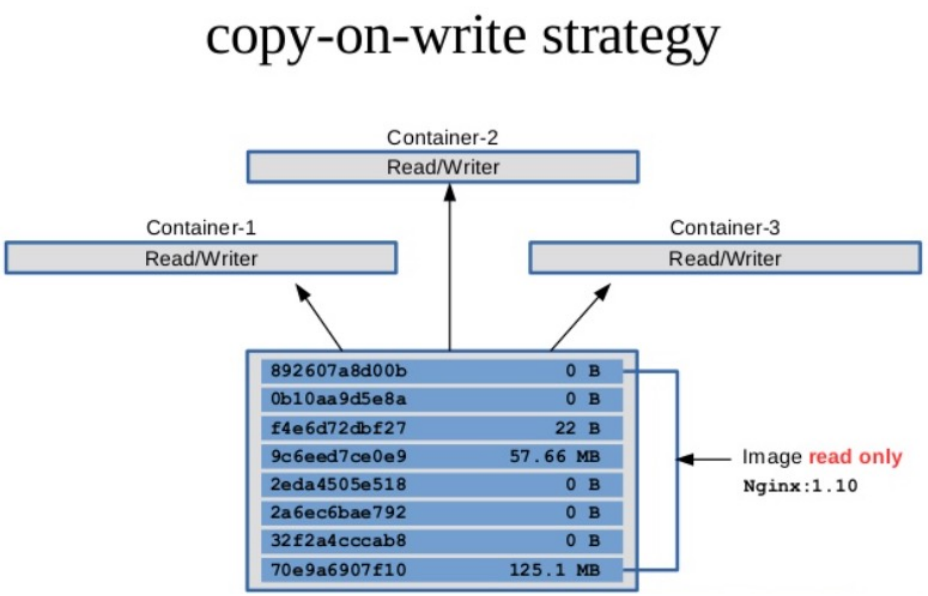
Layers can be shared



https://developpaper.com/practice-of-docker-file-system/

Containers are also layers

- if you modify a file in a running container
- that file get copied from base layer into the container layer
- so “copy on write”



Exercise

- Create an account on Docker Hub

Tagging Images

- $ docker tag <image>:<old_tag> <image>:<new_tag>
- Remember that “image name” is colloquial, actually it’s <account>/<repository>:<tag>

Saving container to a new image

- docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

Exercise

- run the “alpine” image interactively (remember the shell in alpine is “ash” not “bash”)
- create a new text file inside the container (see https://vitux.com/3-ways-to-create-a-text-file-quickly-through-the-linux-terminal/) and write 1 line of text
- exit from the container
- save the container to a new image <user>/alpine:test
- log in to your Docker Hub (in the terminal) *$ docker login*
- push the new image to Docker Hub *$ docker push <image>*

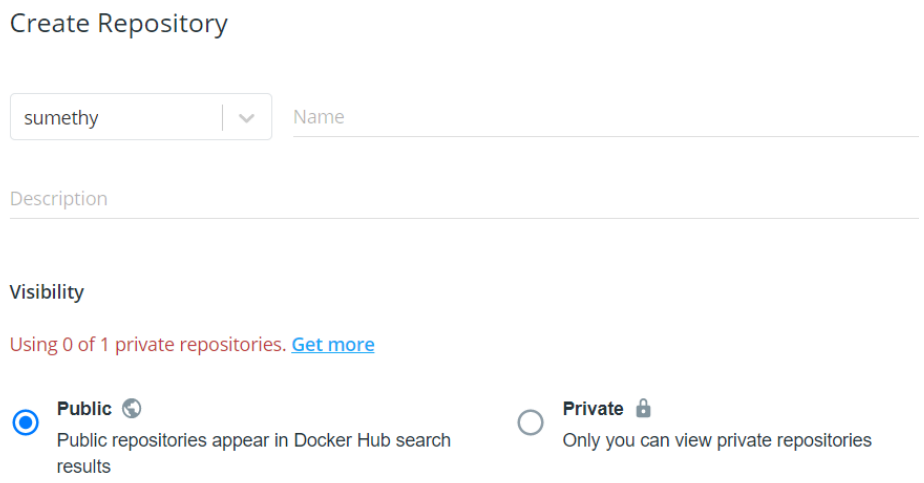
Exercise

- Delete the local copy of the image you just uploaded
- Pull it back from Docker Hub
- Run it and check if the file you created is still there

```
(base) PS C:\Users\acer> docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
ubuntu                    latest              f63181f19b2f        5 days ago          72.9MB
redis                     latest              621ceef7494a        12 days ago         104MB
postgres                  latest              1f1bd4302537        3 weeks ago         314MB
alpine                    latest              389fef711851        5 weeks ago         5.58MB
sumethy/alpine            test                389fef711851        5 weeks ago         5.58MB
nginx                     latest              ae2feff98a0c        5 weeks ago         133MB
sumethy/nginx             test                ae2feff98a0c        5 weeks ago         133MB
centos                    latest              300e315adb2f        7 weeks ago         209MB
docker/getting-started    latest              67a3629d4d71        2 months ago        27.2MB
bretfisher/jekyll-serve   latest              c098d75a5697        7 months ago        383MB
hello-world               latest              bf756fb1ae65        12 months ago       13.3kB
liaad/yake-server         latest              ca20ec436a95        23 months ago       375MB
liaad/yake                latest              5d928c8817c8        24 months ago       345MB
(base) PS C:\Users\acer>
```

How to create private repo?

- simply create the repo before you upload



Dockerfile

What is Dockerfile?

- A “recipe” to create your own image

```
FROM python:3.7
WORKDIR /app
COPY requirements.txt ./requirements.txt
RUN pip install --upgrade pip \
    && pip install -r requirements.txt
```

Example of Dockerfile

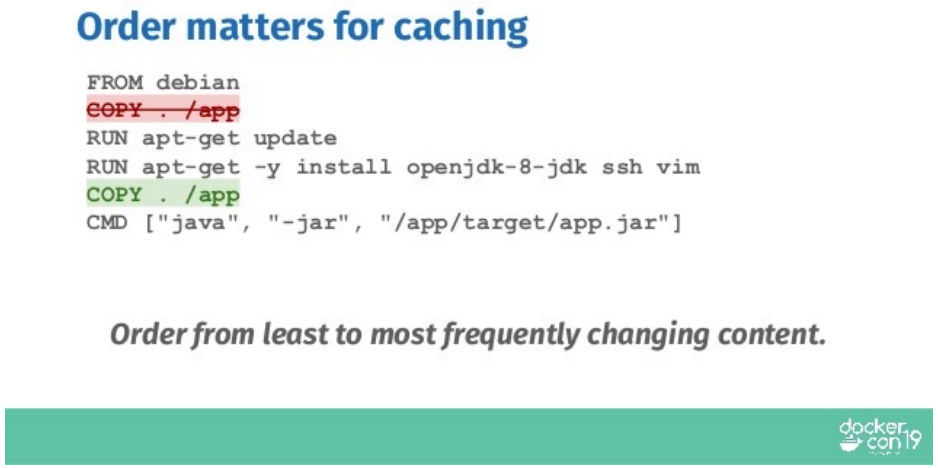
- https://github.com/nginxinc/docker-nginx/blob/41156d8a36bd03b2fb36353ba31f16ada08d9e48/mainline/debian/Dockerfile

Dockerfile keywords

- FROM base image
- LABEL just label
- ENV environment variable, set keys and values in container
- RUN run linux command
- COPY copy file to container
- ENTRYPOINT default executable
- EXPOSE expose port(s)
- STOPSIGNAL skip
- CMD default command

Writing Docker file

- Order matters
- Use && to group multiple Linux commands in to one command



https://www.slideshare.net/Docker/dcsf19-dockerfile-best-practices

Building an image

- create an empty folder
- put docker file in that folder, along with all the other files used in Dockerfile
- (run from inside the foder) *$ docker build -t <tag> .* (note the . at the end)
- The . means the **build context** is the current directory

Exercise – build and run the image

```
build_dir/
        Dockerfile
        index.html
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Some Page</title>
    </head>
    <body>
        <p>Hello World!</p>
    </body>
</html>
```

```dockerfile
FROM nginx:latest
WORKDIR /usr/share/nginx/html
COPY index.html index.html
```

Exercise

- Here is a Python script that reads two numbers (a,b) from the command line and generate a 10-length array random integer between a and b
- Write Dockerfile
- Build image, use *python:3.11.12-slim-bookworm* as the base image
- Run the image, overriding the default CMD to *python /app/script.py 1 10*
- Verify that output is random numbers from 1-9
- Submit the Dockerfile

```python
# save this file as script.py
# copy it into image at /app

import sys
import numpy # need to install this

if __name__ == '__main__':
    a = sys.argv[1]
    b = sys.argv[2]
    arr = numpy.random.randint(a,b,(10,))
    print(arr)
```

Exercise

- Your Dockerfile should:
  - use python:3.11.12-slim-bookworm as base image
  - change working directory to /app
  - copy script.py (host) to /app (container)
  - install numpy
  - no need for CMD as we will override it