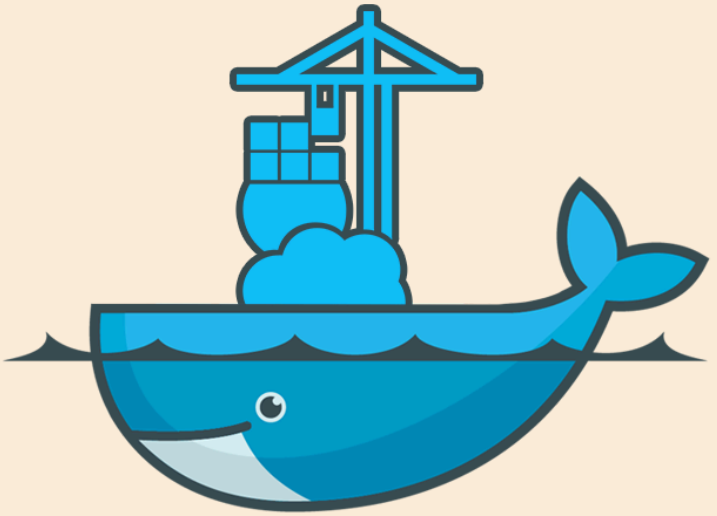
Docker & Portainer



UMD Homelab Club Meeting 2025-10-21

Overview

1. Docker
    - Packages
    - Architecture
    - Registries & repositories
    - Images
    - docker run
    - Volumes
    - Containers
    - Dockerfile / docker build
    - compose.yaml docker compose
2. Portainer

Docker

Docker: Install Packages

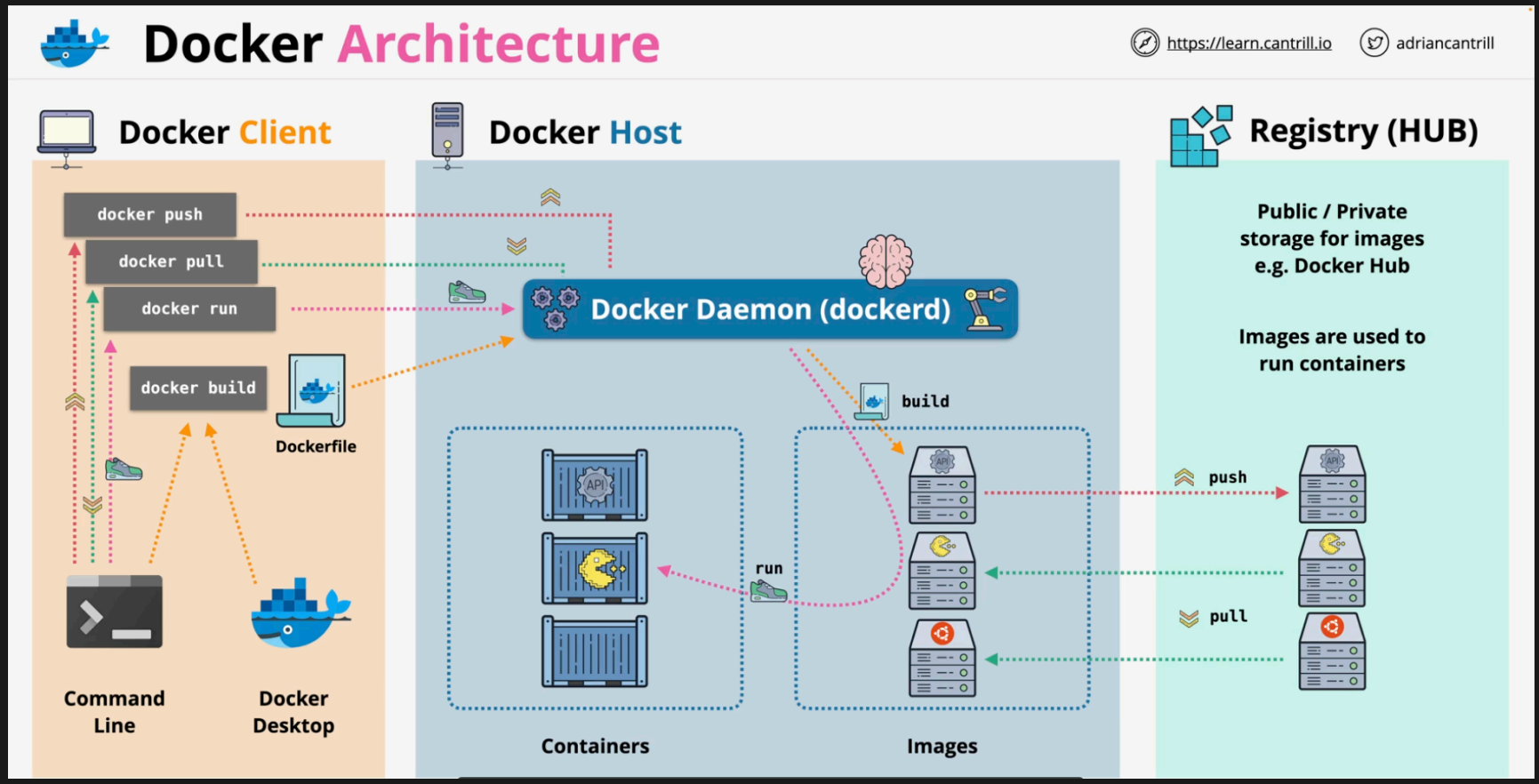
Normally we would install Docker, but since we are doing this offline this has already been baked into the image.

The newer versions of Raspberry Pi OS follow the generic Debian instructions [0].

We have notably also added our user to the docker group so that we can use Docker without root privileges.

[0] https://docs.docker.com/engine/install/debian/
[1] https://docs.docker.com/engine/install/linux-postinstall/

Docker Architecture



https://learn.cantrill.io/p/docker-fundamentals

Docker: System Info

System info: `docker info`
System Resources: `docker system df`

Example:

```
labclub@homelabpi:~ $ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          3         2         371.3MB   180.4MB (48%)
Containers      2         1         0B        0B
Local Volumes   1         1         526.1kB   0B (0%)
Build Cache     31        0         155.1MB   155.1MB
```

Docker Registry

a centralized location that stores and manages container images (e.g. Docker Hub [1] is a public registry)

Docker Repository

a collection of related container images within a registry (like a folder)

Docker Tags

the version/variant label (e.g. :latest, :v1)

[0] https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-registry/
[1] https://hub.docker.com/

Docker: Registries & Repositories (Theory)

Full reference: <registry>/<repository>:<tag>

Download image:

```
docker pull <image-reference> \
```

Upload image:

```
docker push <image-reference>
```

We are not doing this in practice since we are operating offline.

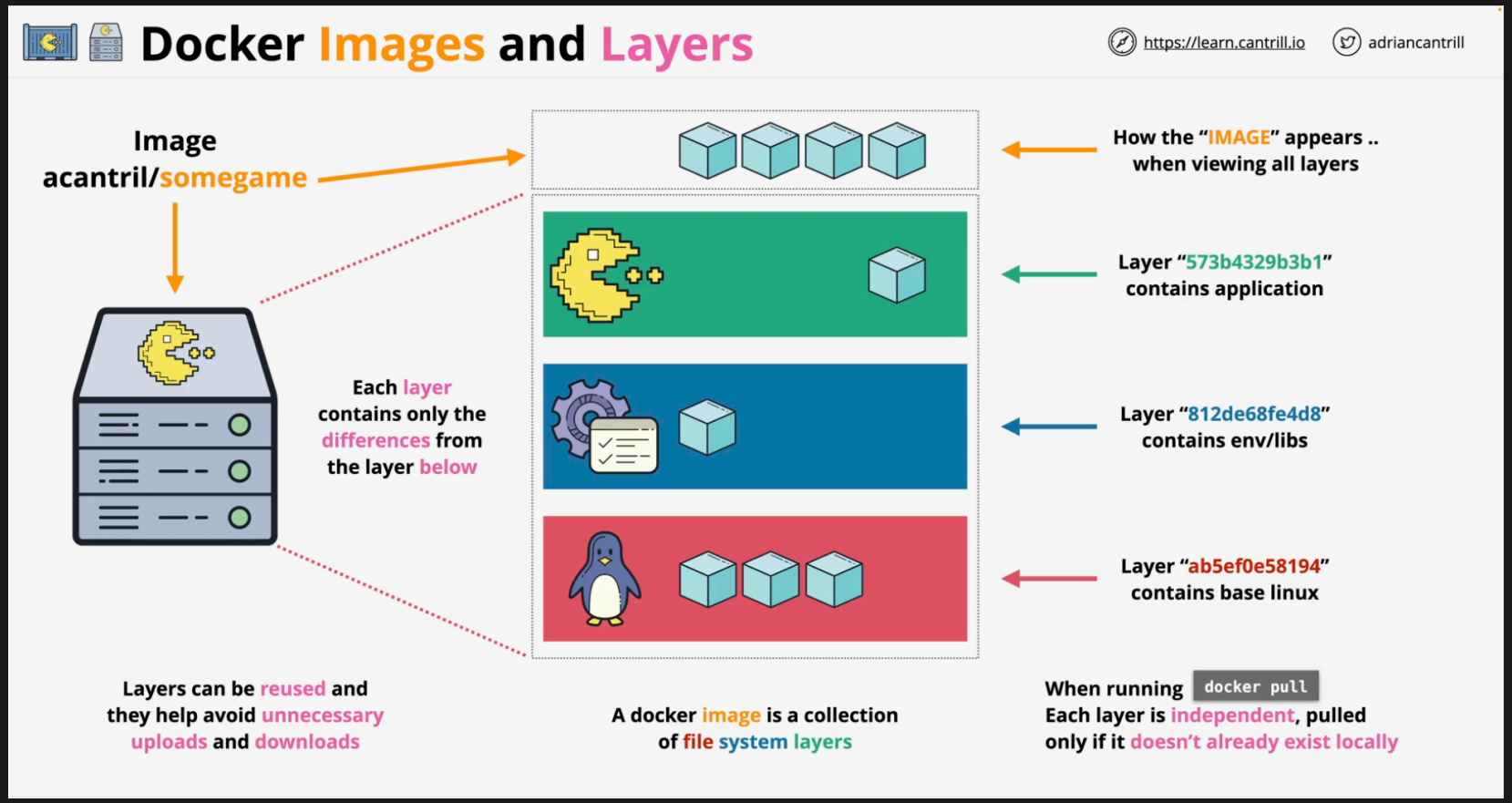
Docker Images

read-only filesystem + metadata layers describing what a container should look like (like a blueprint)

Docker Containers

A process created from an image, with its own isolated filesystem (a read-write layer), network, etc.

Docker: Images (Diagram)



Docker: Images (Theory)

List downloaded images: `docker images`
Get image metadata: `docker inspect <image or image-id>`
Remove image: `docker rmi <image-id>`

Docker: Images (Practice)

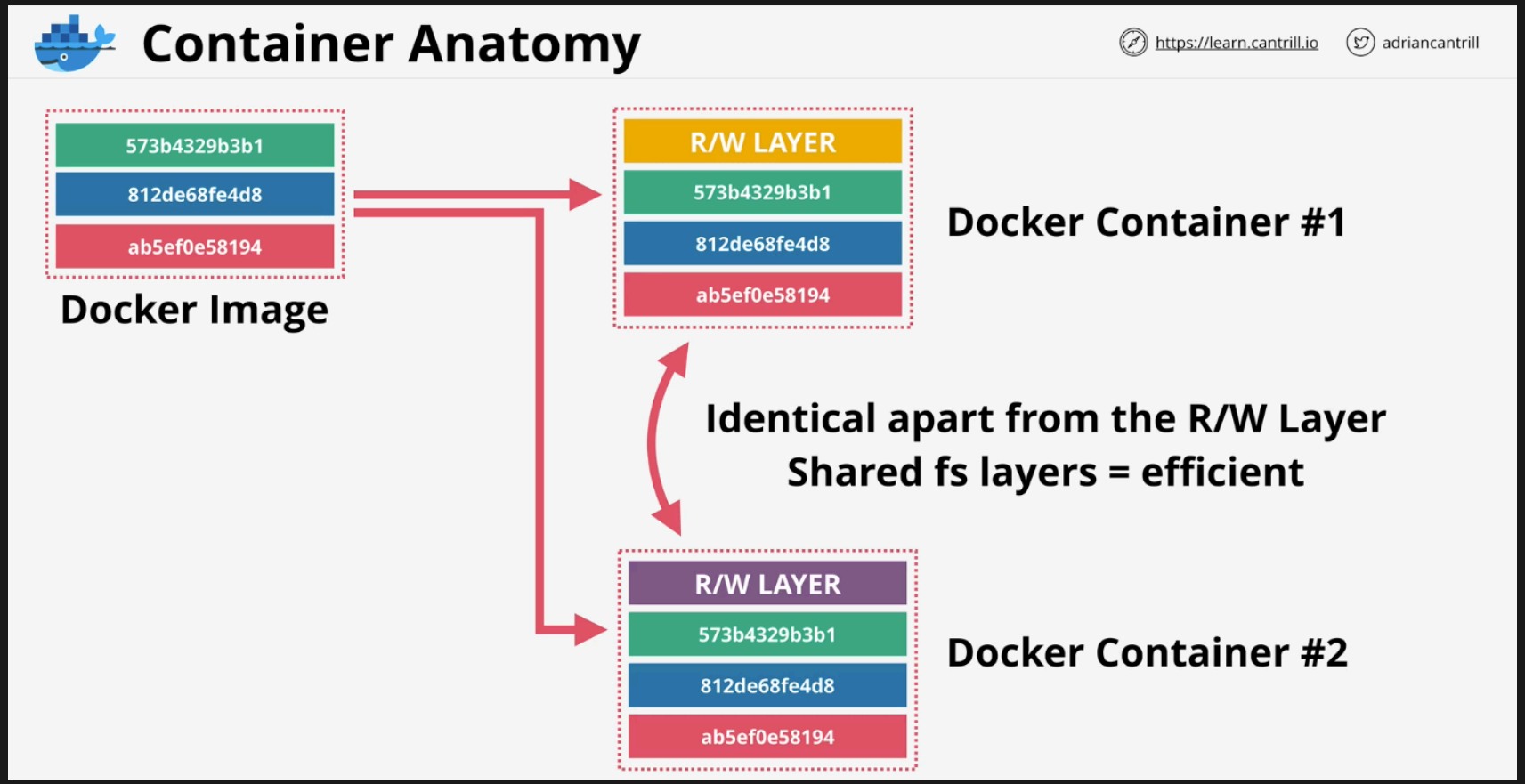
Run:

```
docker images
```

Pick one of the images to inspect:

```
docker inspect <image or image-id>
```

Docker: Containers (Diagram)



Docker: run (Theory)

Give container a name:
`docker run --name <name> ...`

Publish a container's port(s) to the host:
`docker run -p <port-on-host>:<port-within-container> ...`
e.g. http://localhost:8081 --> `docker run -p 8081:80 ...`

Run in background / Detach terminal from container:
`docker run -d ...`

Bind mount (mount directory on host in container):
`docker run --mount type=bind,source=<src-dir-path>,target=<target-dir-path> ...`
or `docker run -v <src-dir-path>:<target-dir-path> ...`

Docker: run (Practice)

Example 1: Default NGINX

```
docker run -p 8080:80 nginx:latest
```

Navigate to http://localhost:8080/ in a web browser.

Ctrl-C when you are ready to move on.

Example 2: Hello World

```
mkdir ~/test-site
cd ~/test-site/
echo "Hello World" > index.html
docker run -d -p 8080:80 -v ~/test-site:/usr/share/nginx/html/ nginx:latest
```

Navigate to http://localhost:8080/ in a web browser.

Docker: Volumes (Theory)

Create a volume: `docker volume create <volume-name>`
List volumes: `docker volume ls`
Get volume metadata: `docker inspect <volume-name>`
Delete a volume: `docker volume rm <volume-name>`

Mount a docker volume when using `run`:
`docker run --mount source=<volume-name>,target=<target-dir-path>`
or `docker run -v <volume-name>:<target-dir-path>`

Bind Mounts vs Volumes:
Bind mounts directly map a file/directory on the host machine to a location within the container. Can use for quick development or to utilize files from the host.
Volumes are storage created and managed by Docker. Can use for persistent storage / does not go away when the container goes away.

We will use a volume later for Portainer.

Docker: Containers (Theory)

List running containers: `docker ps`
List all containers: `docker ps -a`
Get container metadata: `docker inspect <container-id or name>`
List port mappings: `docker port <container-id or name>`
Execute commands within a container:
`docker exec ... <container-id or name> <command> <arguments>`
Run a shell in a container: `docker exec -it <container-id> sh`
Restart container: `docker restart <container-id>`
Stop container: `docker stop <container-id or name>`
Start container: `docker start <container-id or name>`
Get logs (with timestamps): `docker logs -t <container-id or name>`
Remove container: `docker rm <container-id>`

Docker: Containers (Practice)

Run and take note of the ID or Name of the container we previously started:
`docker ps`

List its port mappings:
`docker port <container-id or name>`

Overwrite the `index.html` from within the container:
`docker exec <container-id or name> bash -c "echo 'Hello Container' > /usr/share/nginx/html/index.html"`

Navigate to http://localhost:8080/ in a web browser.

Overwrite the `index.html` from the Pi:
`echo "Hello Pi" > ~/test-site/index.html`

Navigate to http://localhost:8080/ in a web browser.

Stop the container:
`docker stop <container-id or name>`
It should no longer show up in: `docker ps`
But should show up in: `docker ps -a`

Remove the container:
`docker rm <container-id>`

Dockerfile

A set of instructions for building a new image

Docker: Dockerfile (Theory)

```
# this is a comment
FROM - Sets the base image for a build (e.g. Alpine)
LABEL - Adds metadata (e.g. description, maintainer)
# each of the below creates a new layer
RUN - Runs commands in a new layer (e.g. installs or configuration)
COPY - Copies NEW files/folders from src (client machine) to destination (new image layer)
ADD - as above, but can add from a remote URL & do extraction etc (e.g. adding application/web files)
CMD - set the default command of a container & arguments, can be overridden by providing command line args in docker run
ENTRYPOINT - Similar to CMD, explicitly overridden by --entrypoint. When used together, CMD is appended to ENTRYPOINT
EXPOSE - Informs docker what port the container app is running on (metadata only, still need -p) (default: tcp)
```

Build an image from a Dockerfile:

`docker build <working-directory>`

Docker: Dockerfile (Practice)

```
sudo chown -R labclub:labclub /petcontainer/
cd /petcontainer
```

Take a look at what's already there:

```
ls
cat index.html \
```

Create a subdirectory and move the current files in there:

```
mkdir site
mv ./index.html ./site/
mv ./petimage* ./site/
```

Docker: Dockerfile (Practice)

Create a Dockerfile:
vim Dockerfile

```
FROM nginx:latest

LABEL maintainer="HomelabClub"

COPY ./site/index.html /usr/share/nginx/html/
COPY ./site/petimage*.jpg /usr/share/nginx/html/

RUN echo "<html><body><h2>Image built on $(date)</h2></body></html>" \
    > /usr/share/nginx/html/buildtime.html

ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]

EXPOSE 80
```

Build an image from the Dockerfile you just made:
docker build -t petcontainer .
docker images
docker history petcontainer:latest

Try using your new image:
Show that you can override what you put in CMD:
docker run -p 8080:80 petcontainer:latest -v
Run the actual container:
docker run -p 8080:80 petcontainer:latest

Navigate to http://localhost:8080/ in a web browser.

Docker: Dockerfile (Practice)

Now I know what you are thinking, there is room for more pets on the site. Lets fix this.

```
cp ./site/index.html ./mod-index.html
```

Add two more lines for petimage2.jpg and petimage3.jpg:

```
vim ./mod-index.html
```

```
docker run -d -p 8080:80 -v ./mod-index.html:/usr/share/nginx/html/index.html petcontainer:latest
```

Navigate to http://localhost:8080/ in a web browser.

compose.yaml

YAML file describing how one or more containers work / work together (network, volumes, environment).

Docker: Compose (Theory)

Having a unique run command long term can be annoying, and the deployment of your containers can get more and more complicated as you have more of them.

Docker: Compose (Practice)

```
vim compose.yaml
```

```yaml
services:
  petsite:
    image: petcontainer:latest
    container_name: petsite
    ports:
      - "8080:80"
    volumes:
      - ./mod-index.html:/usr/share/nginx/html/index.html:ro
    restart: unless-stopped
```

```
docker compose up -d
docker ps
```

Navigate to http://localhost:8080/ in a web browser.

```
docker compose down
docker ps
```

Portainer

UI to deploy and manage containers

[0] https://www.portainer.io/

Portainer

And we are going to run it... in a container.

Then make a persistent volume for portainer's data:

```
docker volume create portainer_data
docker volume ls
```

And run (one line):

```
docker run -d -p 8000:8000 -p 9443:9443 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

Navigate to https://localhost:9443/ in a web browser.

(Optional)
Since we've been doing a lot of testing, we can prune the old stopped containers that we are no longer using:

```
docker system prune -f
```

Portainer

Enter a password for the admin user: H0melabclub!
Click "Get Started" in the Environment Wizard
Click on the local environment
Click on "Containers"
Click "+ Add Container"

Name: petsite
Image: petcontainer:latest
(uncheck 'Always pull the image')
(click "+ Map additional port")
Host: 8080 --> Container: 80
(click "Deploy the container")

Navigate to http://localhost:8080/ in a web browser.

If you want more...

Recommended longer course (it's free):
https://learn.cantrill.io/p/docker-fundamentals