





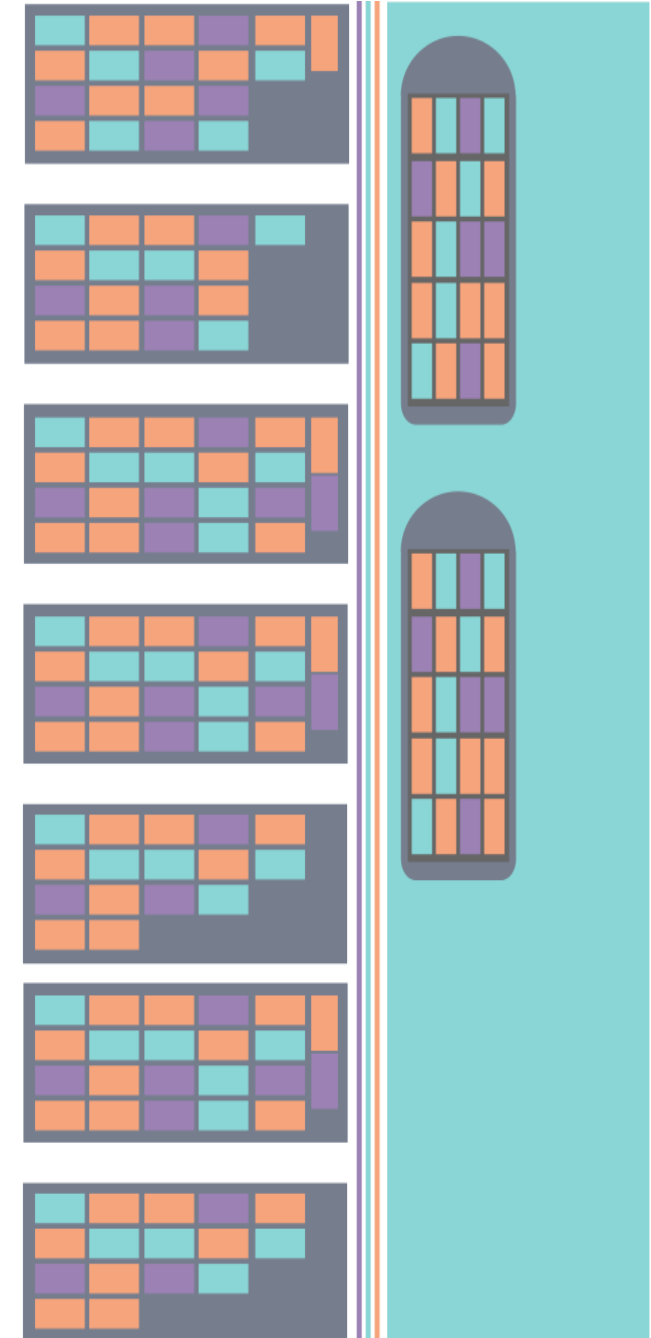
Introduction to Docker for reproducible analysis

What are CONTAINERS ?

- **Common definition:** Object to **HOLD** and **TRANSPORT** something

How are containers related to informatics ?

- **Co-relation in informatics:** It is a **PORTABLE** package **HOLDING** applications and all its necessary means.

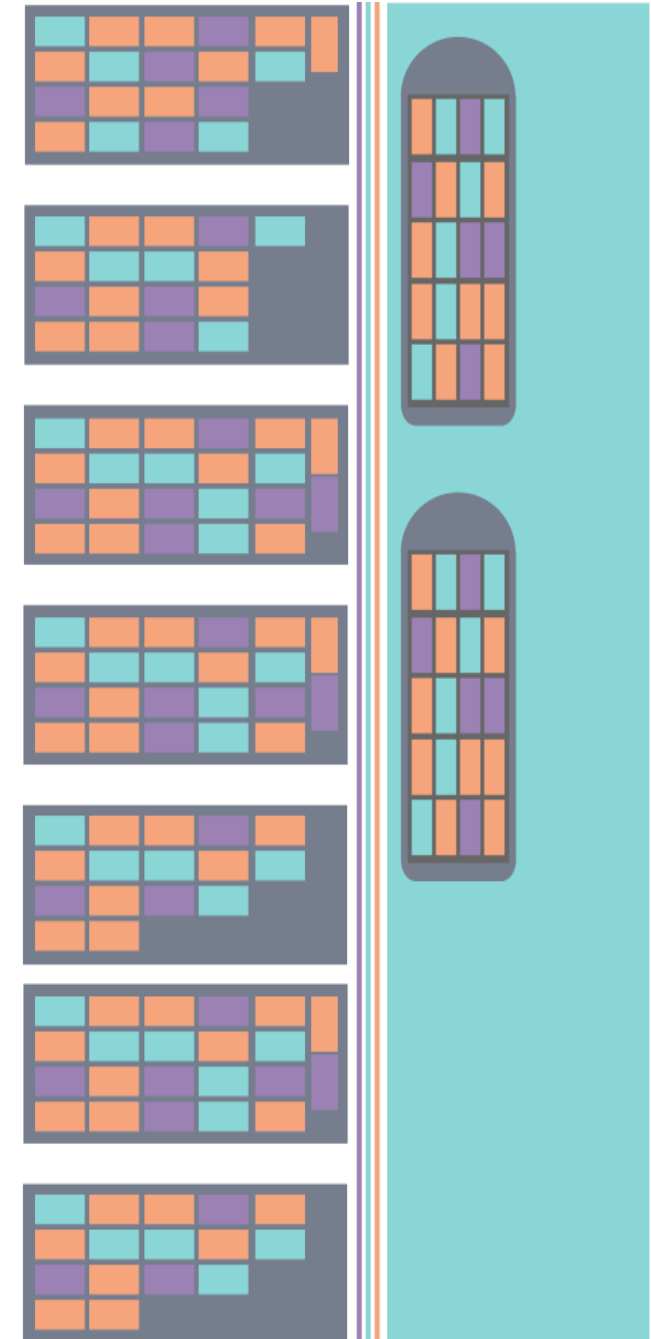


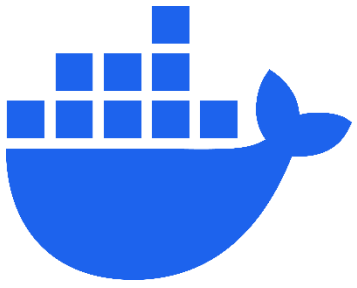
What is a DOCKER ?

- **Common definition:** Person working in a port, responsible for **LOADING** and **UNLOADING** containers.

How is a DOCKER related to (bio)informatics?

- **Co-relation in informatics:** It is an open-source platform to
 - **CREATE** (loading),
 - **MANAGE** (running)
 - **SHIP** (sharing)containers with their applications.



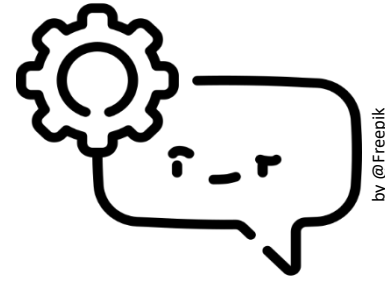


What is a container ?

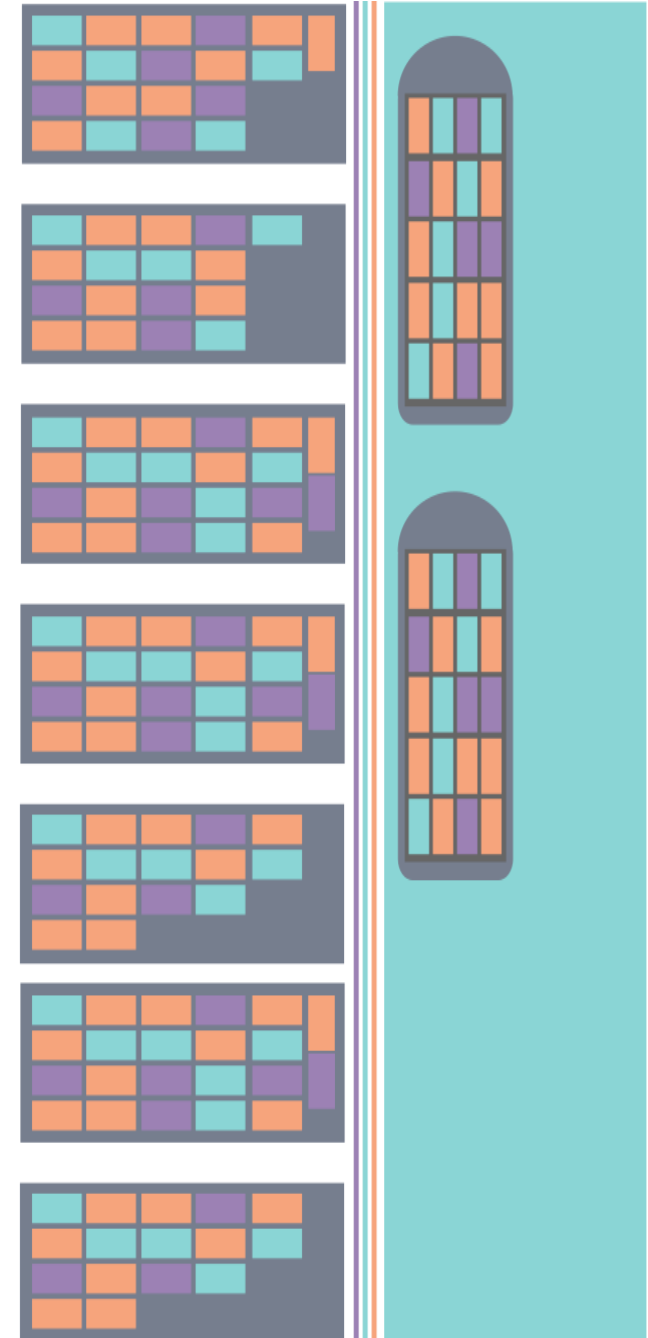
“ A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, routine, system, tools, libraries, settings ”

- <https://www.docker.com/what-container> -

“Well, it works on my machine...”



- Potential general barriers
 - Different Operational system
 - Different hardware
 - Different software versions
 - Technical ability



Docker use cases



- Web application

- Galaxy, GitLab



- Analysis pipeline

- Nextflow, Snakemake



- Testing & continuous integration

- Jenkins, Drone CI



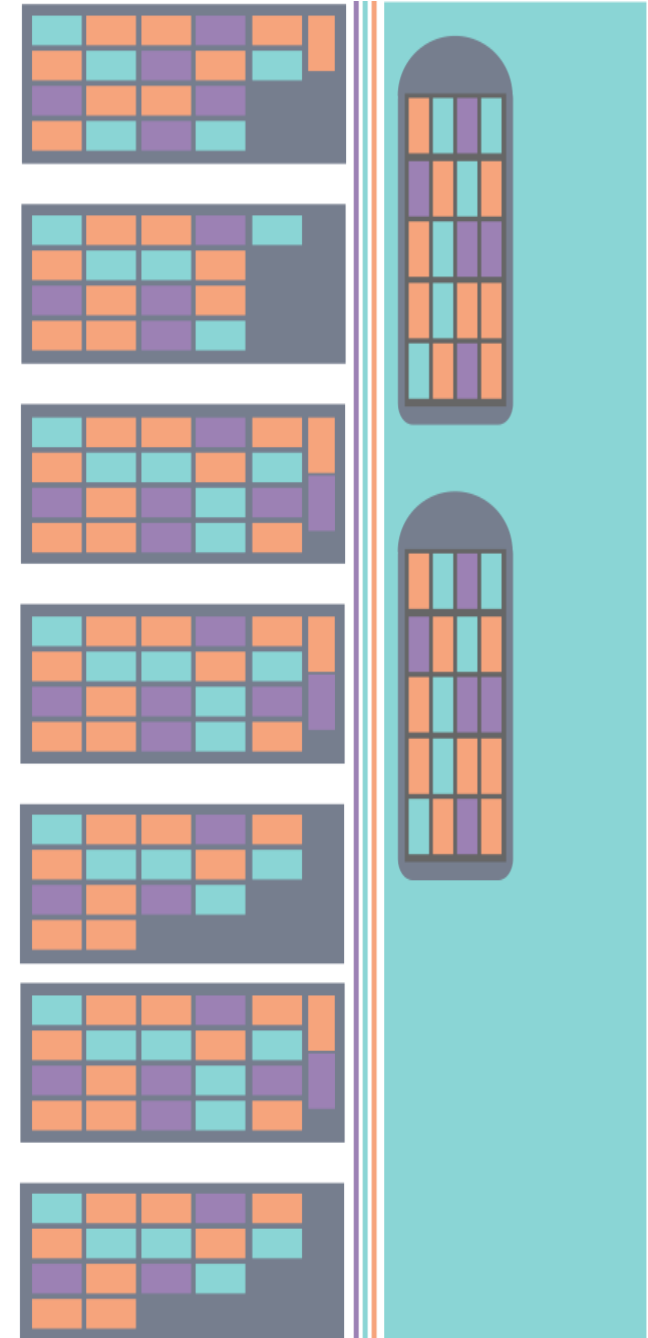
- Difficult to compile apps

- PennCNV, hap.py



- Need for reproducible environments

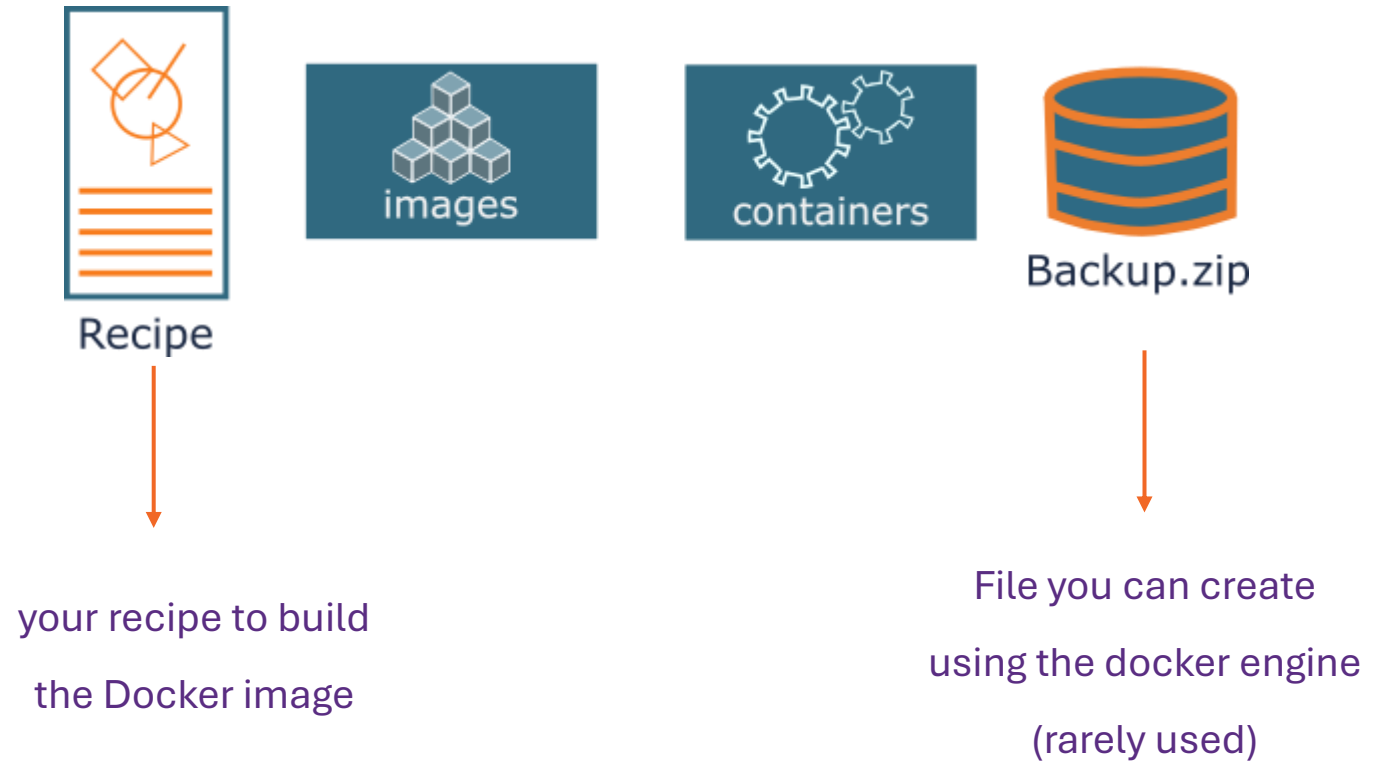
- Jupyter notebooks





How does it work?

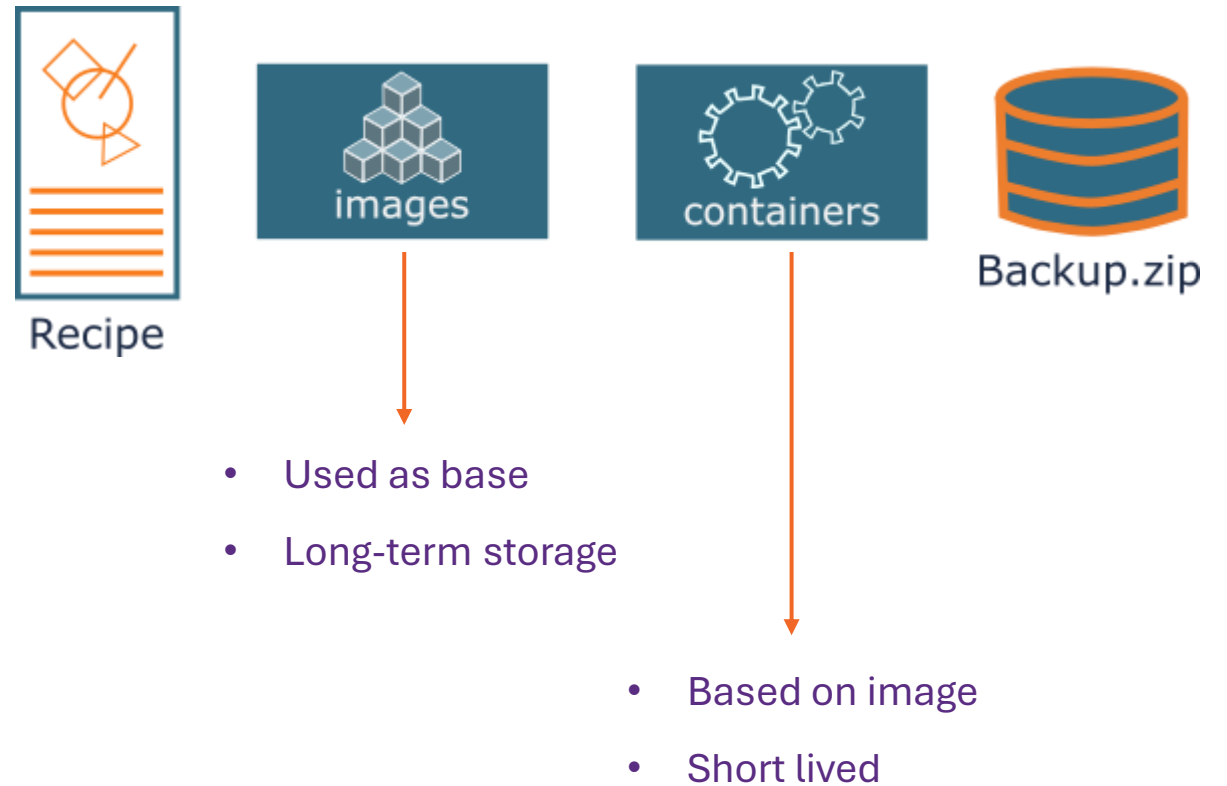
- Important concepts
 - **Dockerfile:** Your recipe
 - **Docker image:** Static artifact
 - **Container:** Running image (functional)
 - **Backup.tar:** Compacted file
 - **Docker engine:** 'Manager'





How does it work?

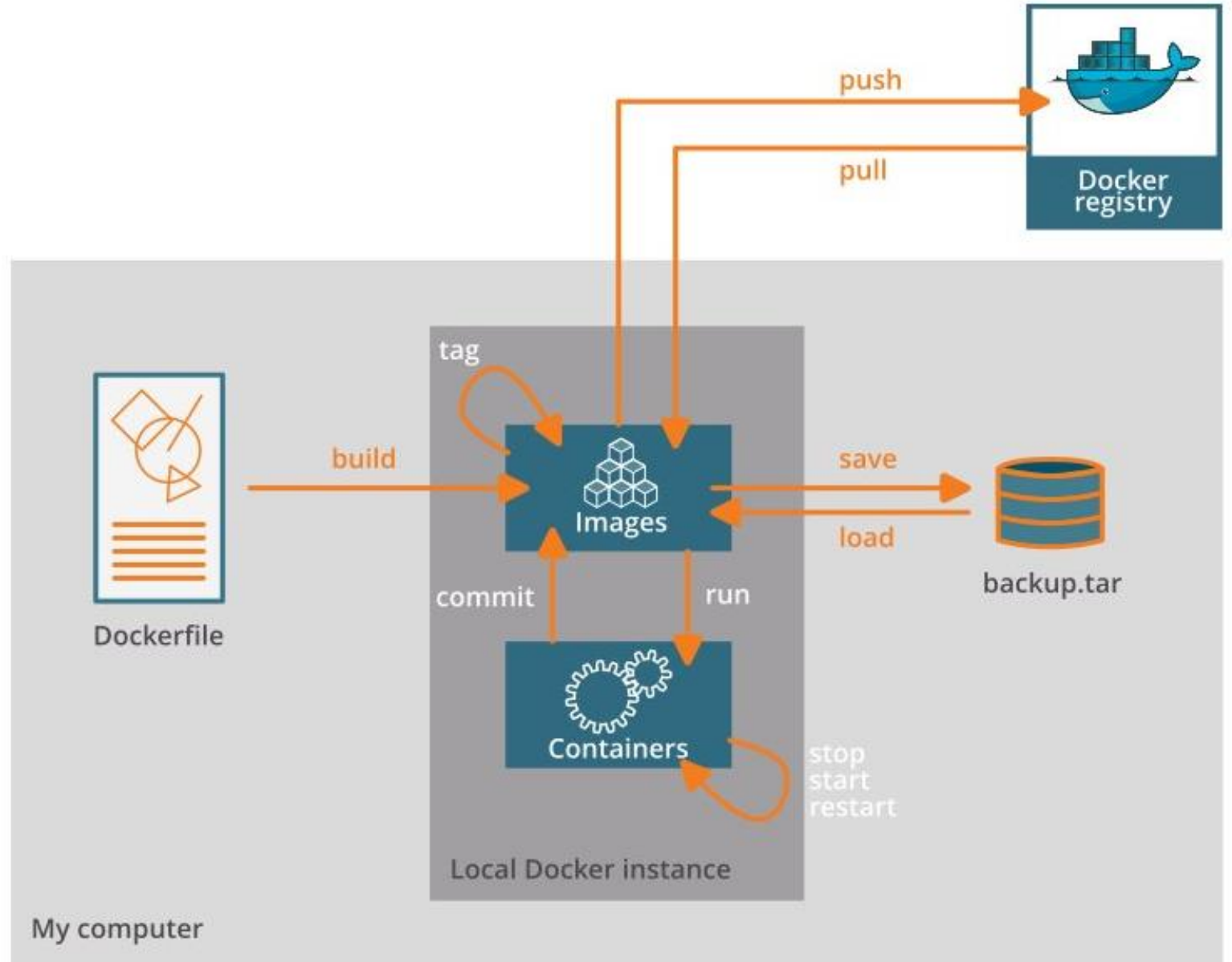
- Important concepts
 - **Dockerfile:** Your recipe
 - **Docker image:** Static artifact
 - **Container:** Running image (functional)
 - **Backup.tar:** Compacted file
 - **Docker engine:** 'Manager'





How does it work?

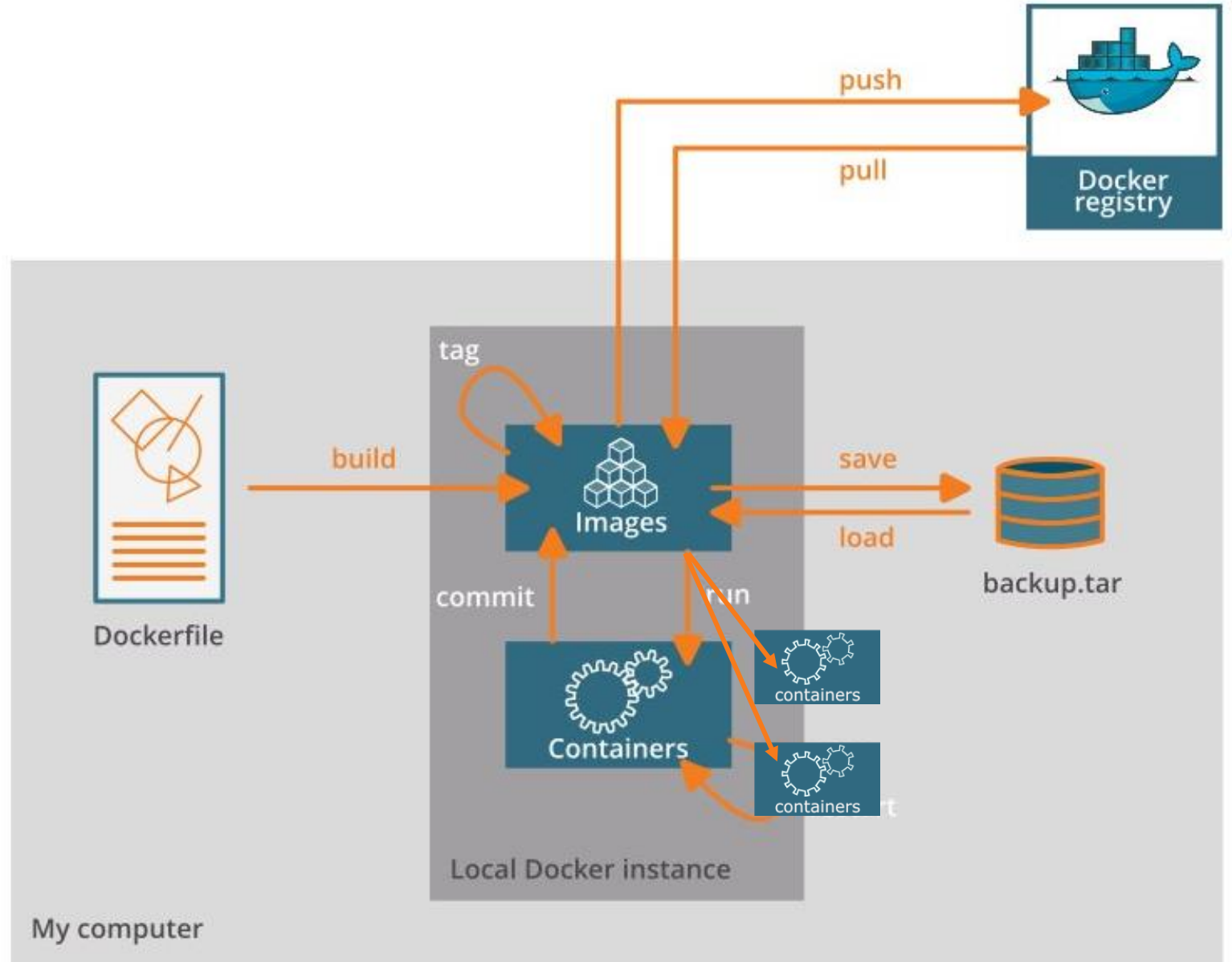
- Important concepts
 - **Dockerfile:** Your recipe
 - **Docker image:** Static artifact
 - **Container:** Running image
 - **Backup.tar:** Compacted file
 - **Docker engine:** 'Manager'

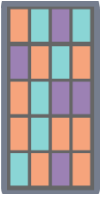




How does it work?

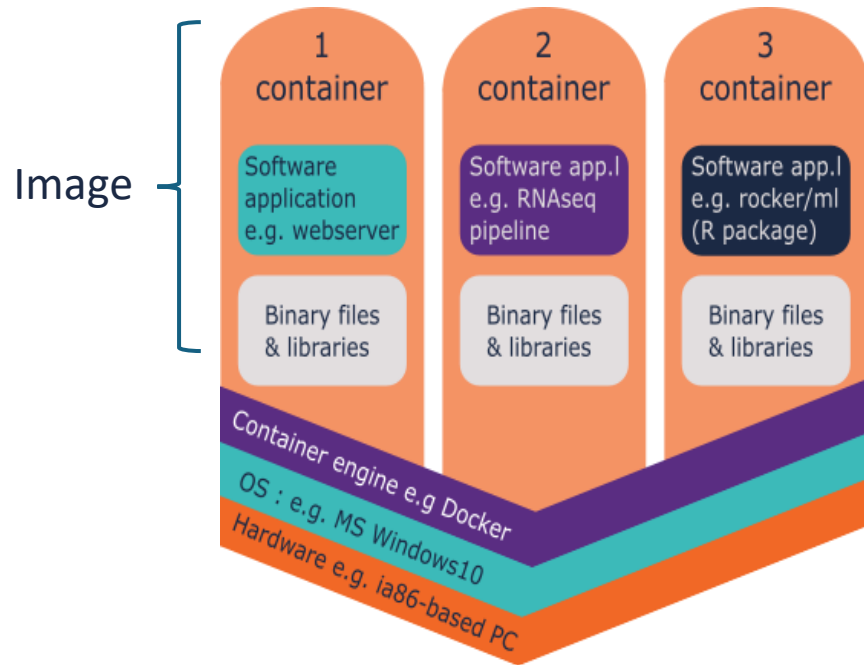
- Important concepts
 - **Dockerfile:** Your recipe
 - **Docker image:** Static artifact
 - **Container:** Running image
 - **Backup.tar:** Compacted file
 - **Docker engine:** 'Manager'



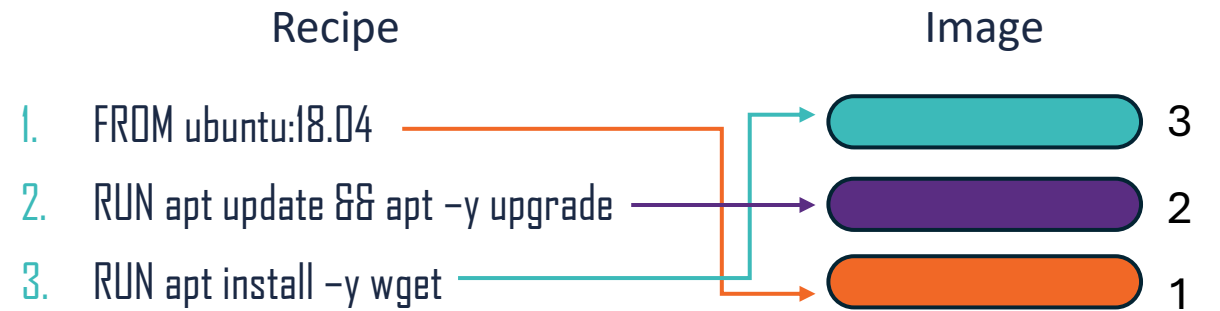


How is it organized?

- General components



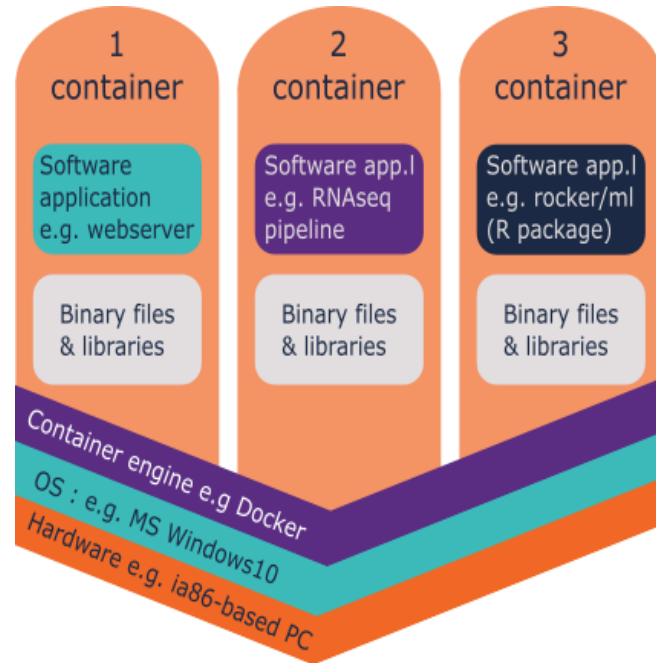
- Docker images layers



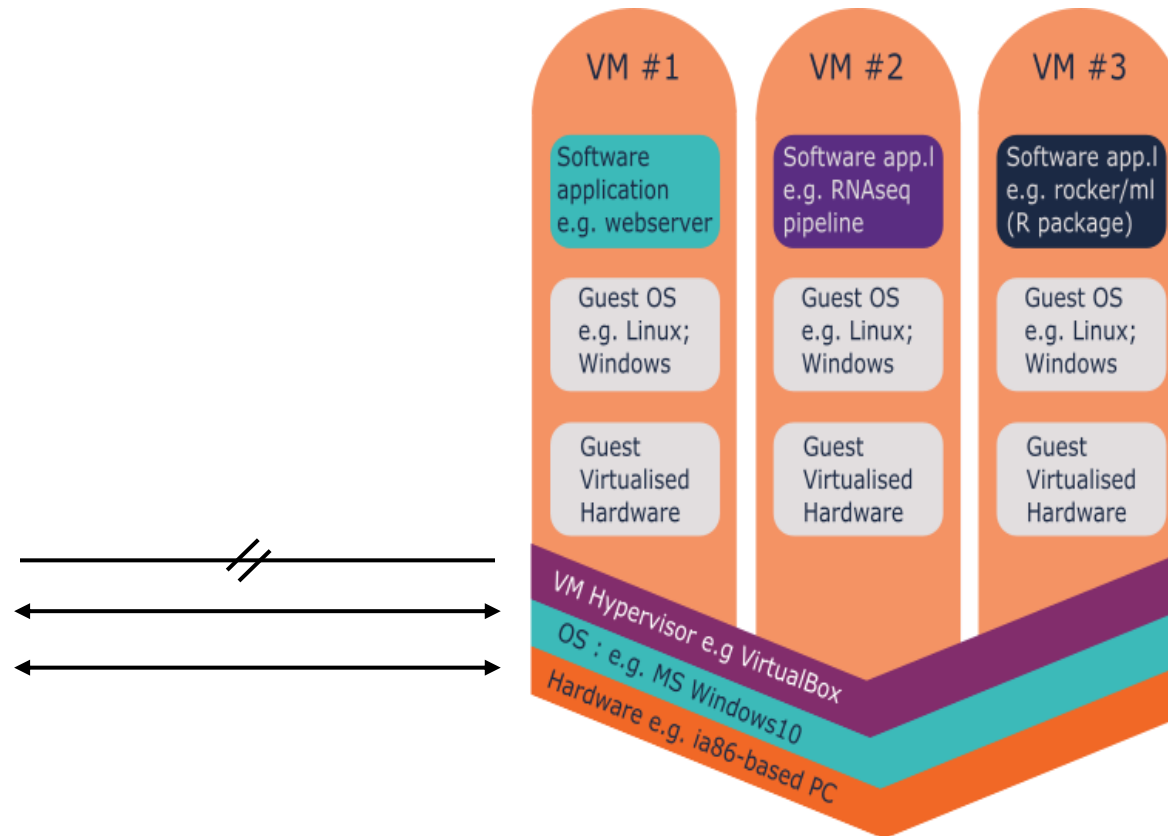


How is it organized?

- General components



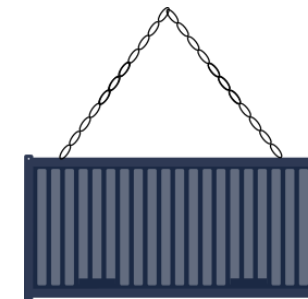
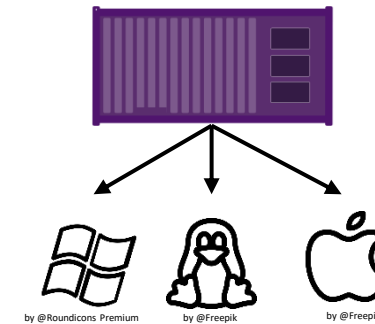
- Docker images layers





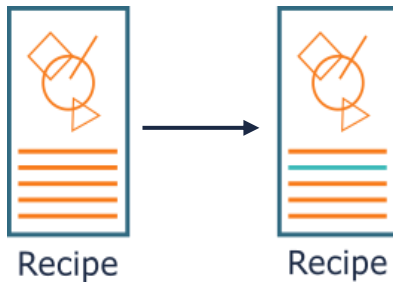
Advantages

- **Bundled Dependencies**
 - Contain all their dependencies = installing hurdles
- **Cross-platform Installation**
 - Contain their own operating system = run on any platform (even Windows!)
- **Easy Distribution**
 - Shared on Docker Hub or as 'image.tar' file





Advantages



- **Safety**
 - Can't access files on the host machine
- **Ease-of-Use**
 - Can always be run using one single command
- **Easy Upgrades**
 - Easily swapped out for newer versions
 - All persistent data can be retained in a data volume



Other container software



podman



APPTAINER



An aerial photograph of a large port facility. In the foreground and middle ground, there are massive stacks of colorful shipping containers in various colors like orange, blue, red, and green. Several large yellow gantry cranes are positioned along the waterfront, used for loading and unloading ships. The port is situated next to a body of water, with a long pier extending into the sea. The sky is clear and blue.

Using and Reusing available docker images

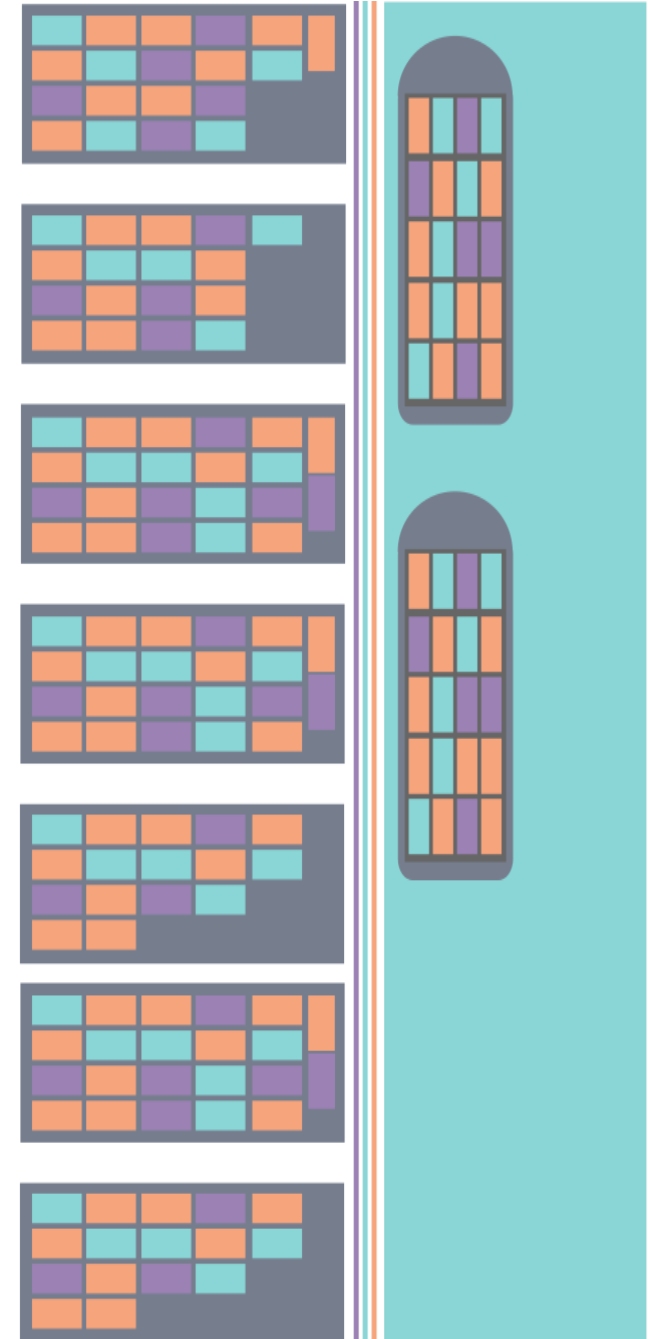
Where can we storage and find Docker images?



Local storage



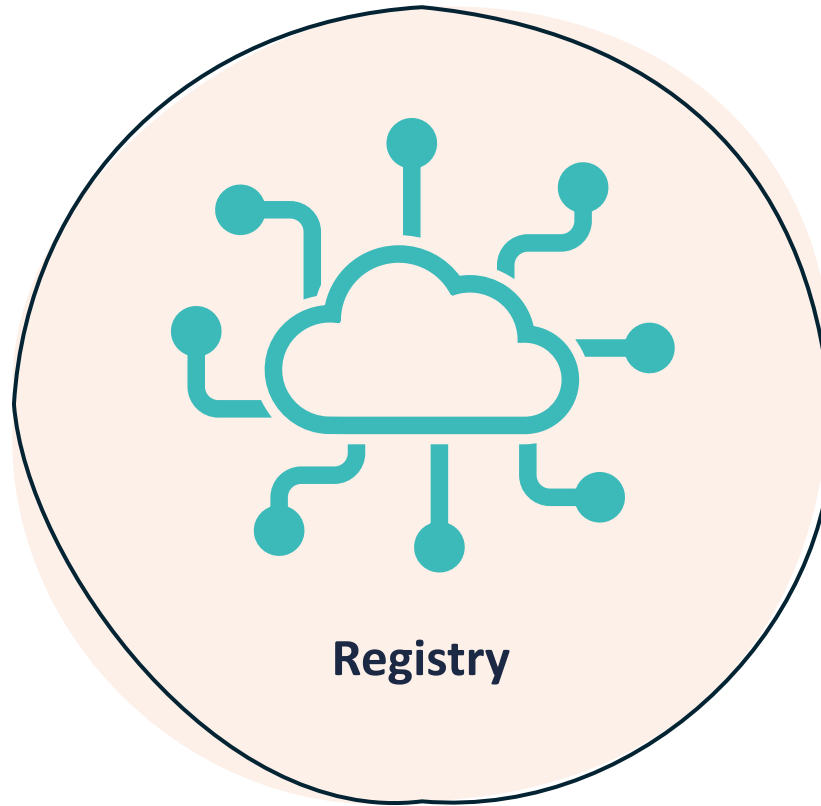
Registry



Where can we storage and find Docker images?

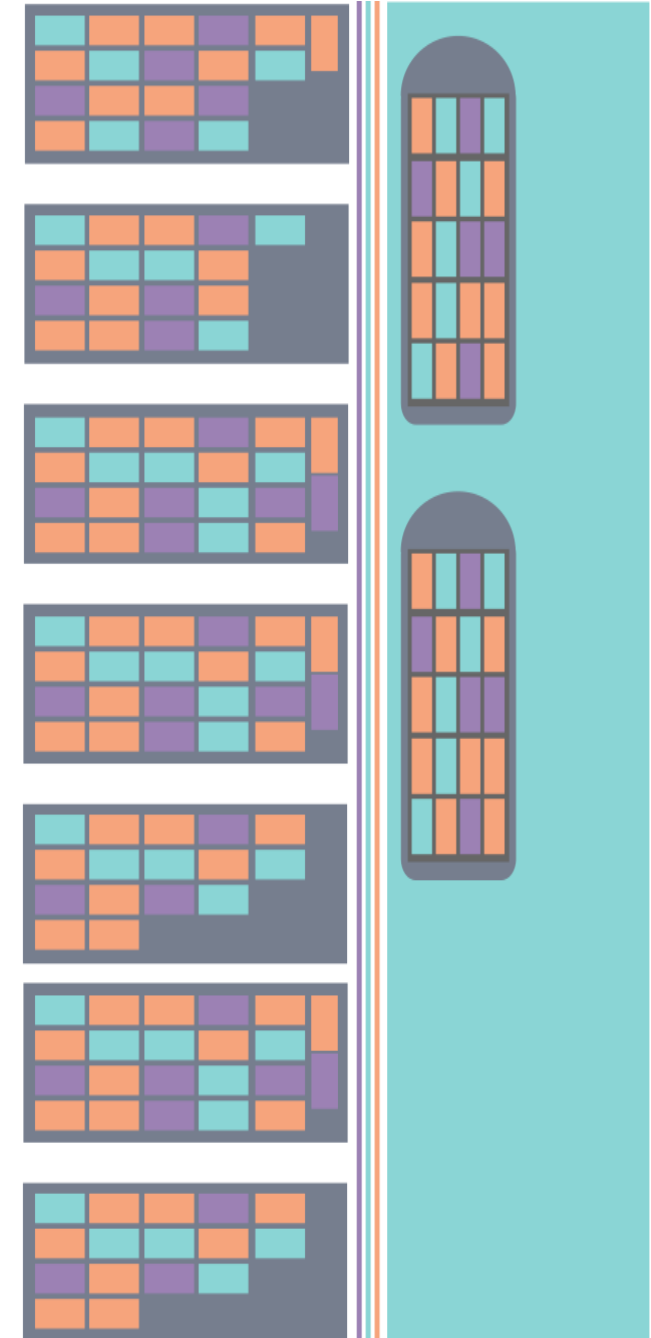


Local storage



Registry

Preferable



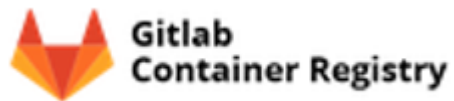


Container Registries

- Main registry for DOCKER containers



- Alternative registries



Amazon ECR



Container Registries

- Main registry for DOCKER containers



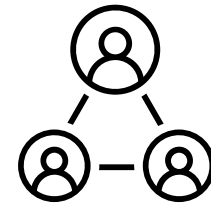
- Alternative registries



Amazon ECR

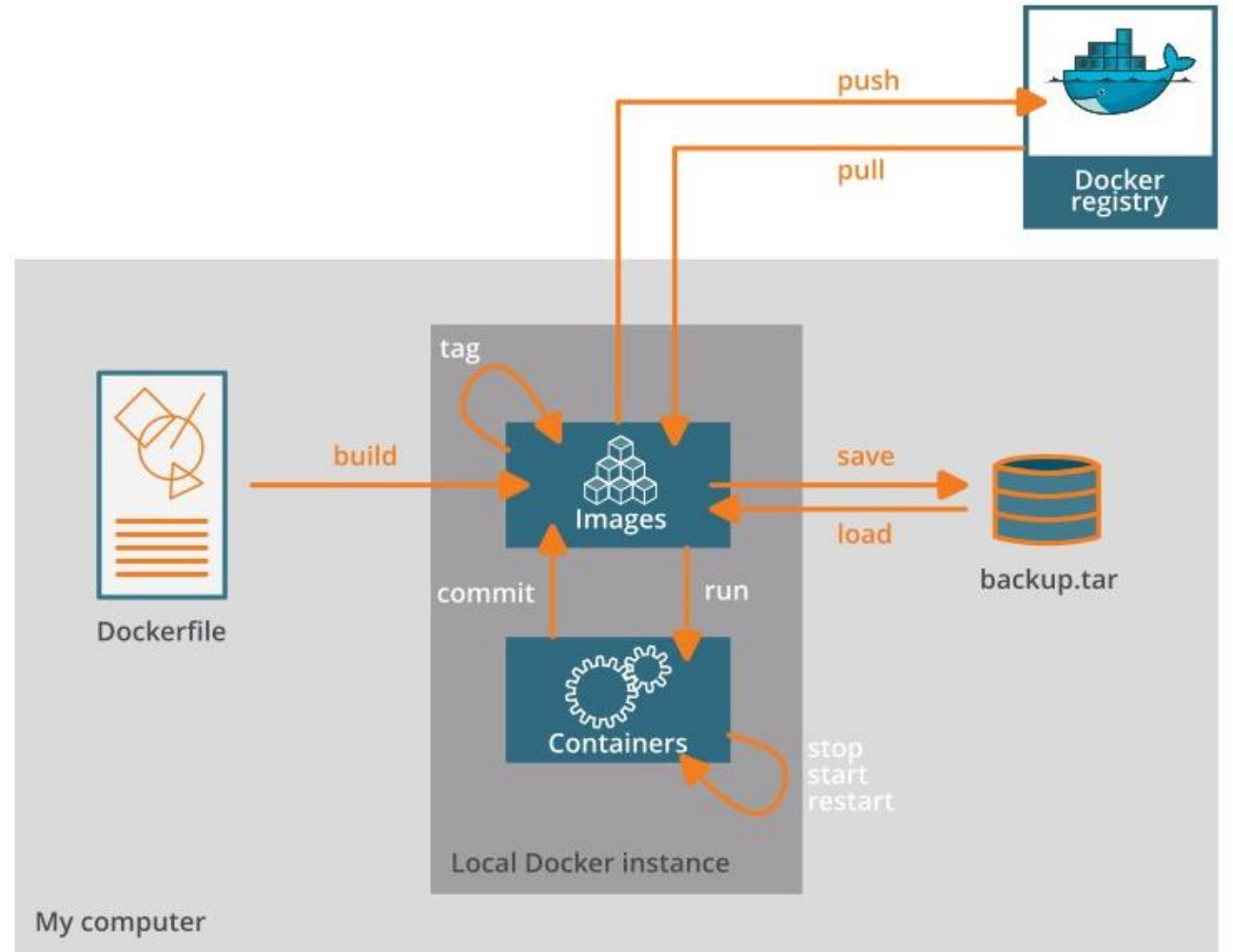


BioContainers





How do we get a container image?

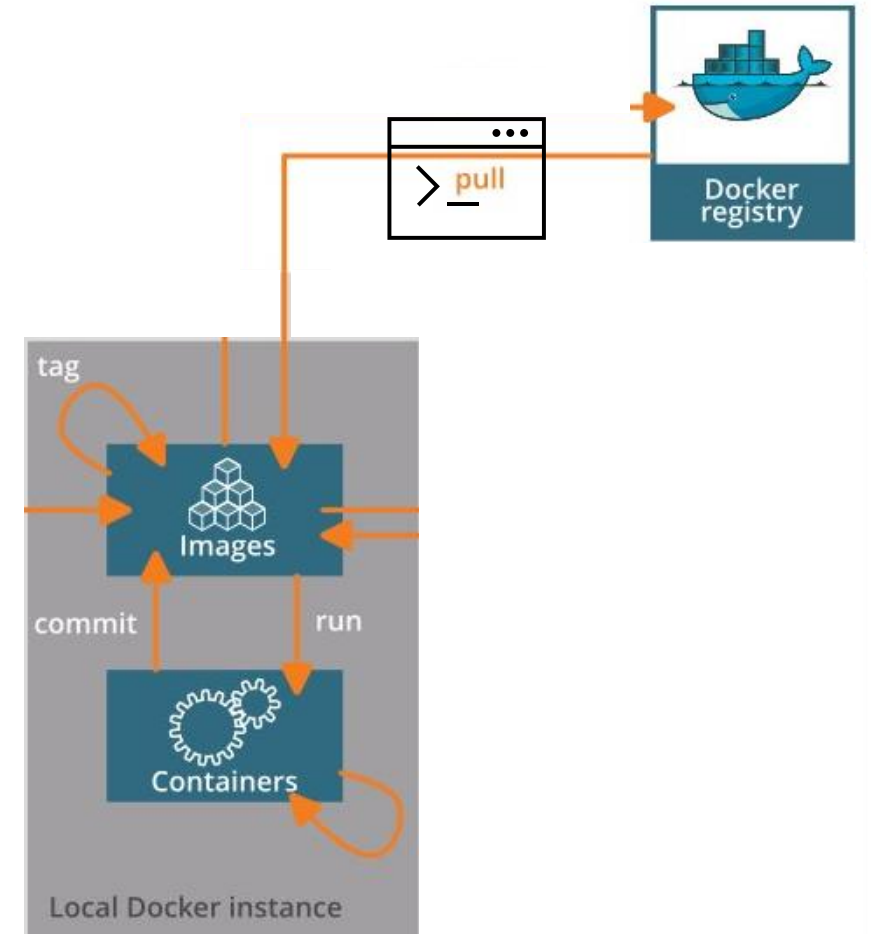


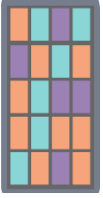


How do we get a container image?

- `docker pull <image>`
- `docker pull <registry/image>`
 - Example: getting Ubuntu image

```
$ docker pull ubuntu  
  
docker pull ubuntu  
Using default tag: latest  
latest: Pulling from library/ubuntu  
d51af753c3d3: Pull complete  
fc878cd0a91c: Pull complete  
6154df8ff988: Pull complete  
fee5db0ff82f: Pull complete  
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7  
Status: Downloaded newer image for ubuntu:latest
```





How do we get a container image?

- `docker pull <image>`
- `docker pull <registry/image>`
 - Example: getting Ubuntu image

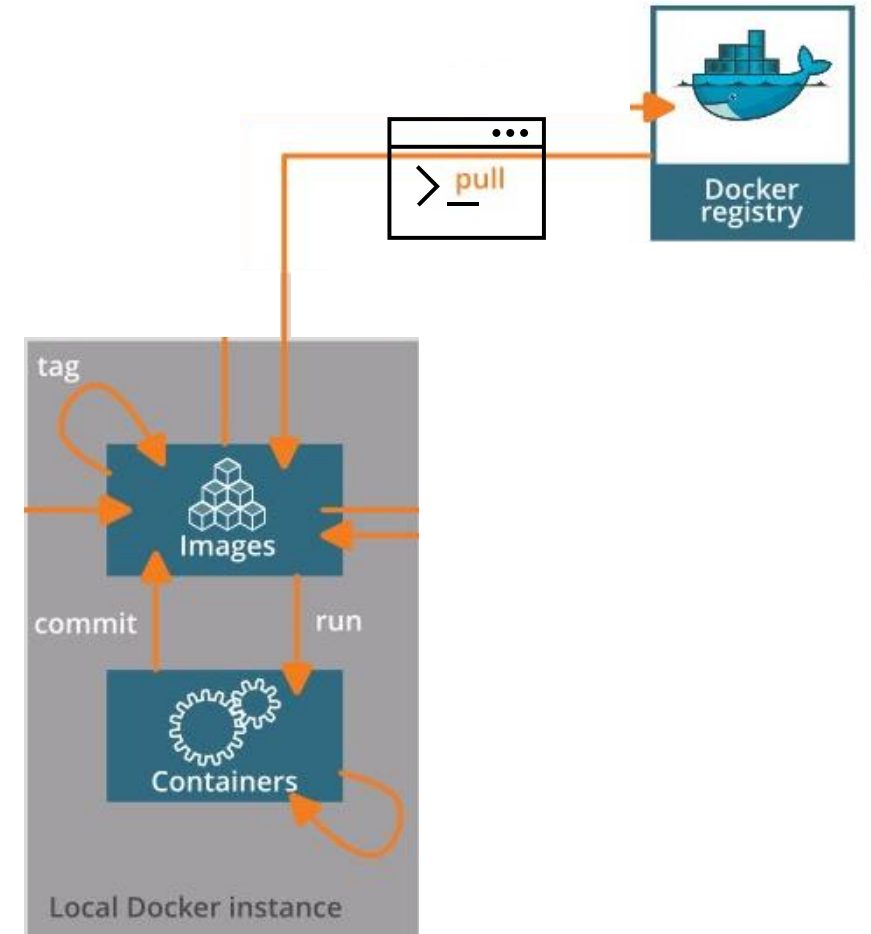
```
$ docker pull ubuntu
```

```
docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
```

- Example: Get a specific version

```
$ docker pull ubuntu:18.04
```

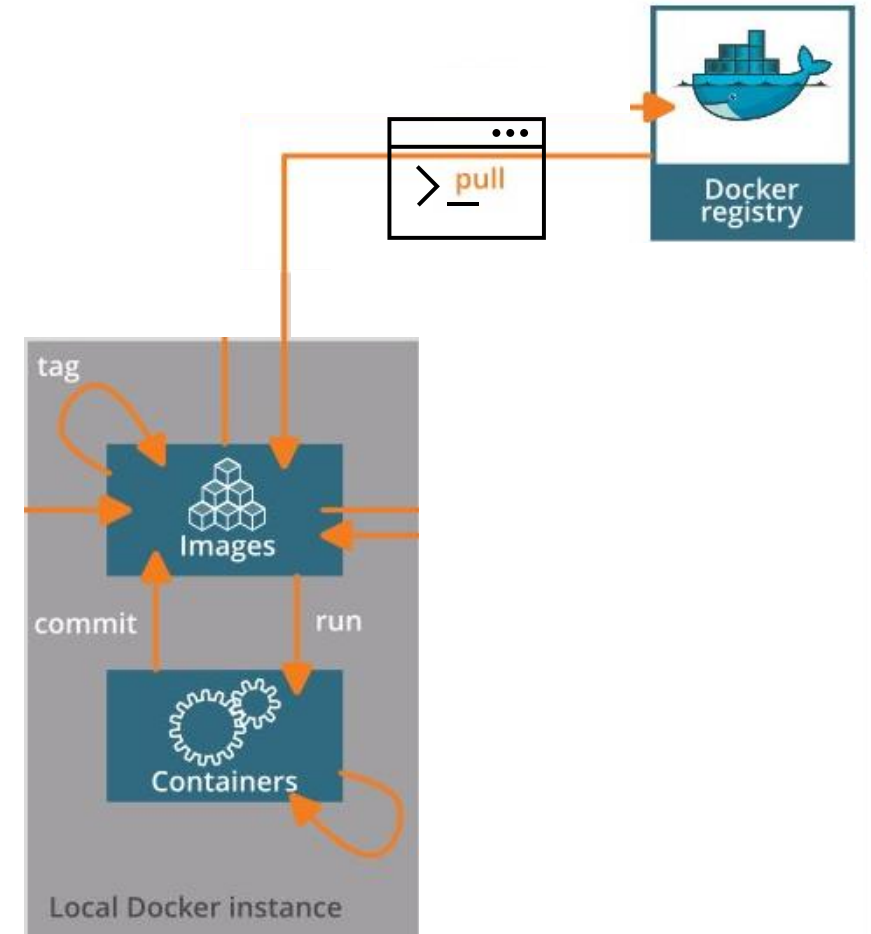
```
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
```

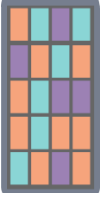




QUIZ TIME: How do we get a container image?

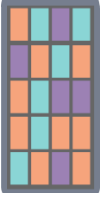
- Is there a better way to 'pull' ??
- If yes, which one?
 - `docker pull ubuntu`
 - `docker pull ubuntu:18.04`
- Why?





Practice time: How do we get a container image?

- Pull ubuntu in your computer:
 - Version 18.04
- Pull from Docker hub
 - Fastqc (A quality control tool for high throughput sequence data)
 - Version 0.11.9_cv7



What else can we do?

- Check all images that you have

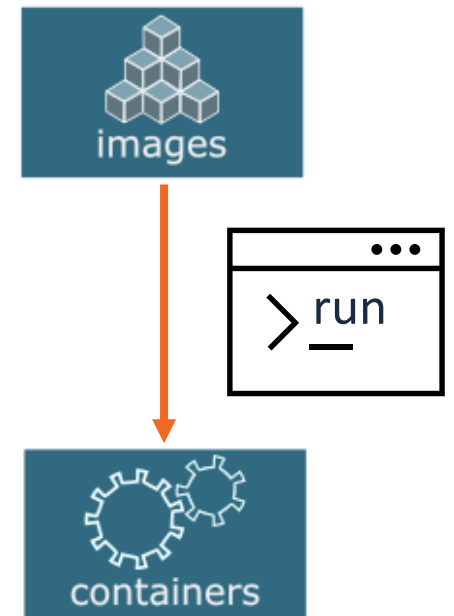
```
$ docker images
```

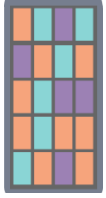
```
$ docker image ls
```

```
$ docker images -a
```

- Run a container with your analysis

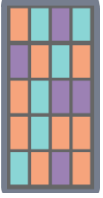
```
$ docker run [docker_options] <container> [container_arguments]
```





Practice time: List your imaged

- List all the images you have pulled or build



Practice time: Run your 1st image

- List all the images you have pulled or build

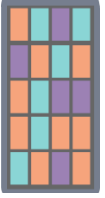
You can execute any program/command that is stored inside the image.

- Run your 1st container

```
$ docker run ubuntu:18.04 /bin/ls
```

```
$ docker run ubuntu:18.04 /bin/whoami
```

```
$ docker run ubuntu:18.04 /
```



Practice time: Run your 1st image

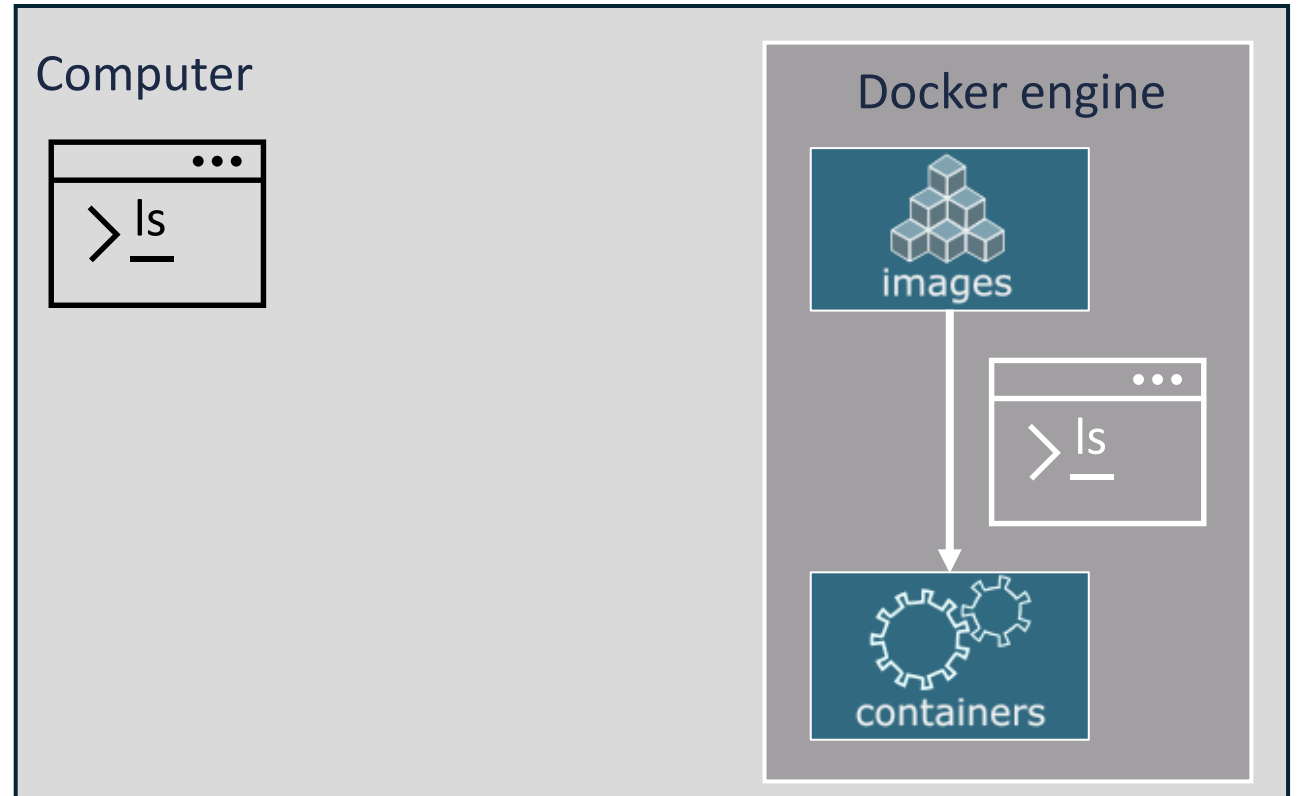
- List all the images you have pulled or build

You can execute any program/command that is stored inside the image.

- Run your 1st container

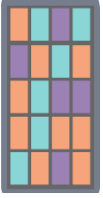
```
$ docker run ubuntu:18.04 /bin/ls
```

- If you run `ls` in your current directory, do you have the same?
- Why?





Docker detach, what does It do?



Docker detach, what does It do?

- Run in the background
 - Detached from the shell

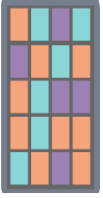
```
$ docker run [docker_options] <container> [container_arguments]
```

```
$ docker run --detach <container> [container_arguments]
```



- Name your container to check later

```
$ docker run --detach --name <my_ctn_name> <container> [container_arguments]
```

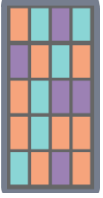


Practice time:

- Run the following without naming it:

```
$ docker run --detach nginx
```

Check the name of your container by checking the process status
- Challenge: How do you list running containers?



Practice time:

- Run the following without naming it:

```
$ docker run --detach nginx
```

Check the name of your container by checking the process status
- Challenge: How do you list running containers?

- List running containers (process status)

```
$ docker ps
```

- List all containers (whether or not running)

```
$ docker ps -a
```



Practice time:

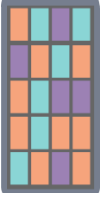
- Run again naming the container

```
$ docker run --detach --name MyUbuntu nginx
```

Check the name of your container by checking the process status

```
$ docker ps
```

```
$ docker ps -a
```

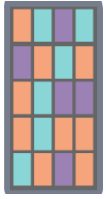


Practice time:

```
bpiereck@LaptopBruna:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
31a3b18e5374	ubuntu:18.04	"/bin/bash -c 'while...'"	6 seconds ago	Up 5 seconds		MyUbuntu
81b5e01f79d8	ubuntu:18.04	"/bin/bash -c 'while...'"	19 seconds ago	Up 18 seconds		upbeat_borg

- Why are IDs and Names useful for?

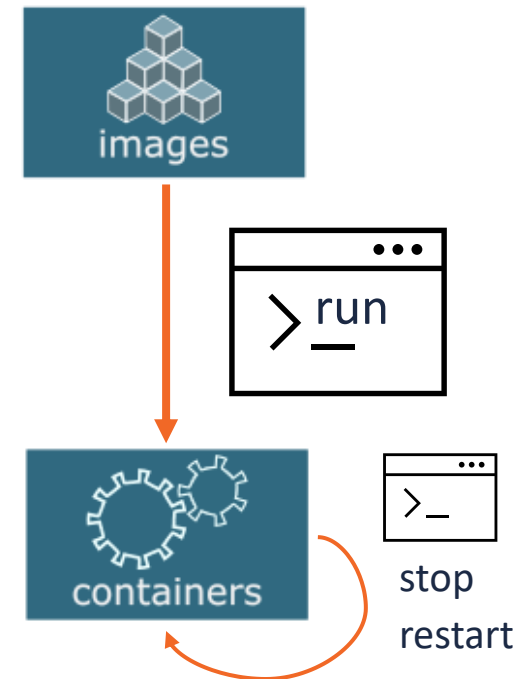


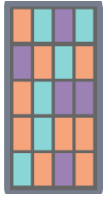
Practice time:

```
bpiereck@LaptopBruna:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
31a3b18e5374	ubuntu:18.04	"/bin/bash -c 'while...'"	6 seconds ago	Up 5 seconds		MyUbuntu
81b5e01f79d8	ubuntu:18.04	"/bin/bash -c 'while...'"	19 seconds ago	Up 18 seconds		upbeat_borg

- Why are IDs and Names useful for?
 - Stop a container
 - Restart a container





Tagging

- Define image Name and Version!

```
docker tag <image ID> <container:tag_name>
```

Let's try to do it!



Tagging

- Define image Name and Version!

```
docker tag <image ID> <container:tag_name>
```

Let's try to do it!

- Check the ID of your images
- Chose one of them to change or add a tag



QUIZ TIME: Naming objects in Docker

- What is the difference between ... ?
 - Tag
 - --name



QUIZ TIME: Naming objects in Docker

- What is the difference between ... ?
 - Tag
 - --name
- Name the container (docker run --name)
 - One image can create +1 container

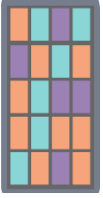
```
bpierreck@LaptopBruna:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
31a3b18e5374	ubuntu:18.04	"/bin/bash -c 'while...'"	6 seconds ago	Up 5 seconds		MyUbuntu
81b5e01f79d8	ubuntu:18.04	"/bin/bash -c 'while...'"	19 seconds ago	Up 18 seconds		upbeat_borg

- Name an image
 - Define the version of an image (docker tag <image ID>)

```
bpierreck@LaptopBruna:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	c613f16b6642	8 weeks ago	187MB
docker/welcome-to-docker	latest	c1f619b6477e	5 months ago	18.6MB
ubuntu	18.04	f9a80a55f492	10 months ago	63.2MB

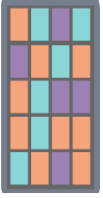


Docker and disk space

- Check space usage (**D**isk **i**n**Fo**)
 - Output
 - Type object
 - Total number of objects
 - Size
 - etc

```
$ docker system df
```





Docker and disk space

- Docker objects are not automatically removed
 - Images
 - Containers
 - Networks
 - Volumes
- Check system space
- Pruning the system
 - The whole system
 - Dangling images
 - Not tagged
 - No references
 - All images not associated to a container



Recipe



My_image





Docker and disk space

- Docker objects are not automatically removed
 - Images
 - Containers
 - Networks
 - Volumes
- Check system space
- Pruning the system
 - The whole system
 - Dangling images
 - Not tagged
 - No references
 - All images not associated to a container



Recipe



Recipe +



images

???



images

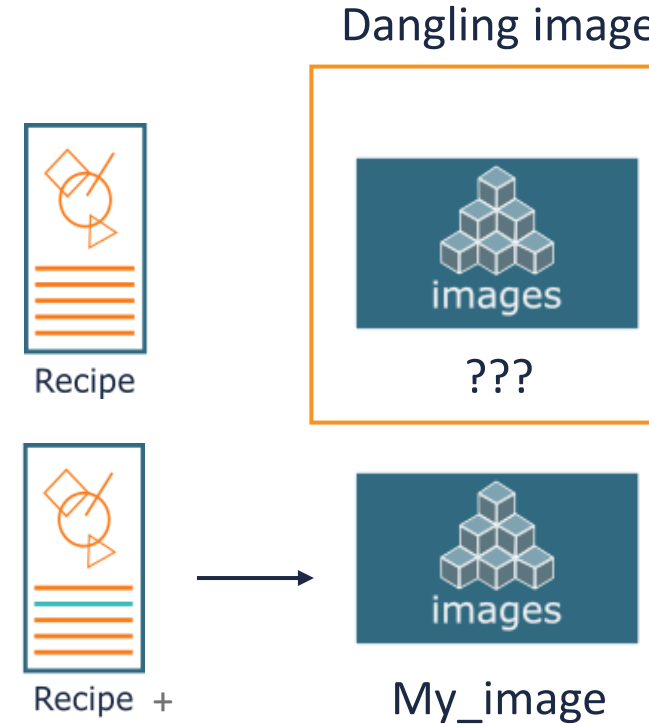
My_image





Docker and disk space

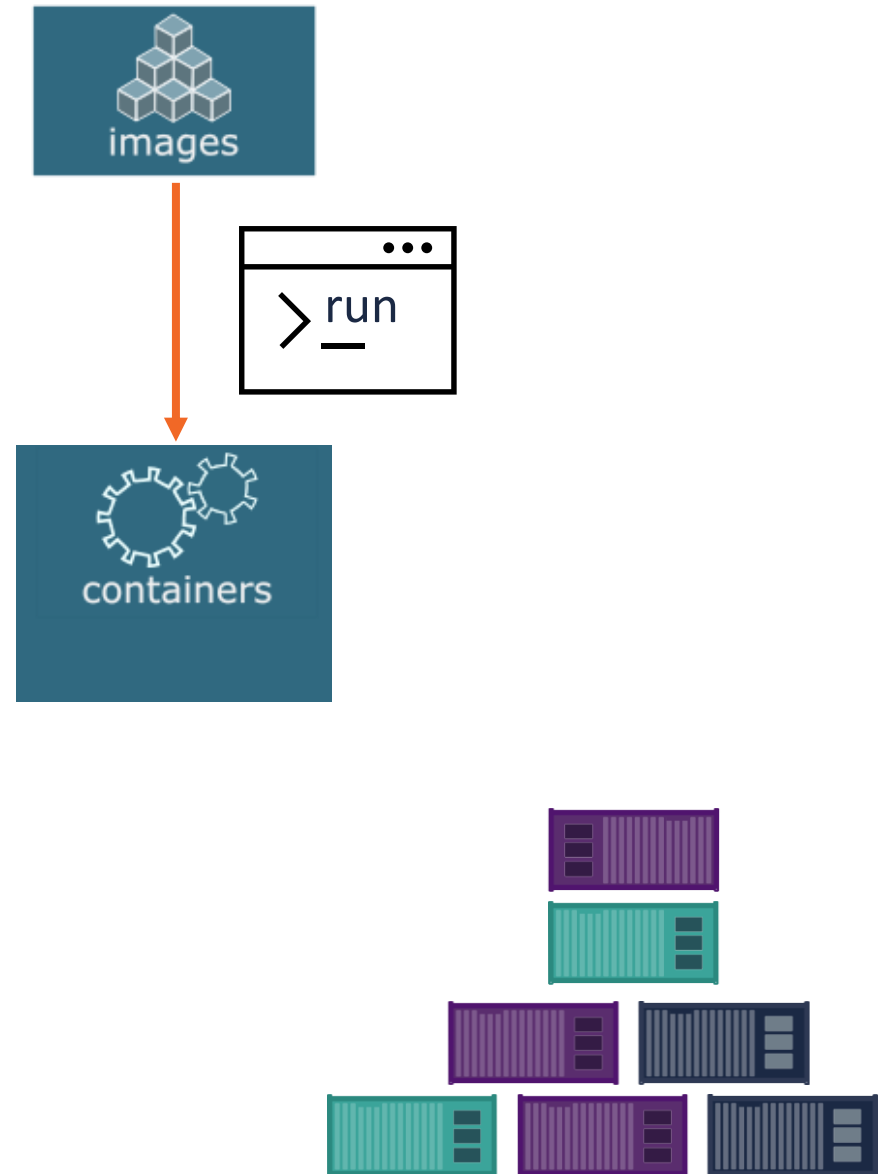
- Docker objects are not automatically removed
 - Images
 - Containers
 - Networks
 - Volumes
- Check system space
- Pruning the system
 - The whole system
 - Dangling images
 - Not tagged
 - No references
 - All images not associated to a container

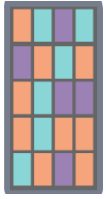




Docker and disk space

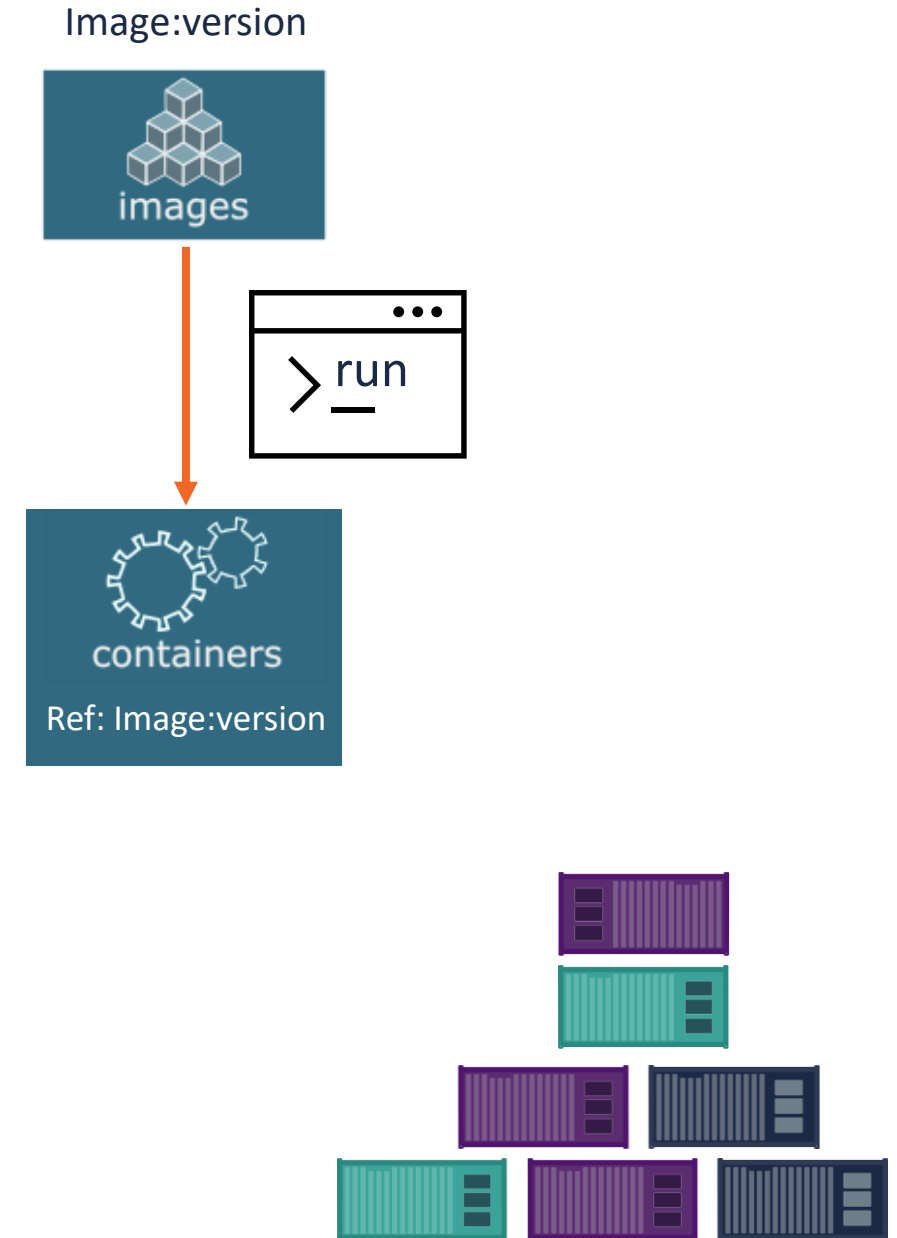
- Docker objects are not automatically removed
 - Images
 - Containers
 - Networks
 - Volumes
- Check system space
- Pruning the system
 - The whole system
 - Dangling images
 - Not tagged
 - No references
 - All images not associated to a container

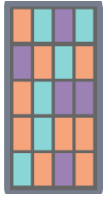




Docker and disk space

- Docker objects are not automatically removed
 - Images
 - Containers
 - Networks
 - Volumes
- Check system space
- Pruning the system
 - The whole system
 - Dangling images
 - Not tagged
 - No references
 - All images not associated to a container





Docker and disk space

- Check space usage (**D**isk **i**n**Fo**)
 - Output
 - Type object
 - Total number of objects
 - Size
 - etc

```
$ docker system df
```



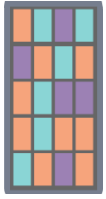
- Clean up
 - Remove (rm)
 - Specify

- Image
- Container



```
$ docker rm -f <container>
```

```
$ docker rmi <image>
```



Docker and disk space

- Check space usage (**D**isk **i**n**F**o)
 - Output
 - Type object
 - Total number of objects
 - Size
 - etc

```
$ docker system df
```



- Clean up

- Remove (rm)
 - Specify
 - Image
 - Container



```
$ docker rm -f <container>
```

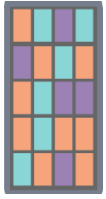
```
$ docker rmi <image>
```

- Major clean up

- Clean all dangling objects
- ```
$ docker system prune
```

- Clean all dangling images
- ```
$ docker image prune
```

- Clean unused containers
- ```
$ docker container prune
```



# Docker and disk space

- Check space usage (**Disk inFo**)
  - Output
    - Type object
    - Total number of objects
    - Size
    - etc

```
$ docker system df
```

Clean all UNUSED objects

**\$ docker system prune -a**

Use it carefully !!!

- Clean up
  - Remove (rm)
    - Specify
      - Image
      - Container



```
$ docker rm -f <container>
```

```
$ docker rmi <container>
```

- Major clean up
  - Clean all dangling objects
    - \$ docker system prune
  - Clean all dangling images
    - \$ docker image prune
  - Clean unused containers
    - \$ docker container prune

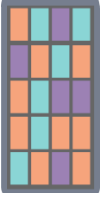




## Practice time: Check and clean

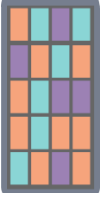


- Check how much disk space you've used
- Clean stopped or unused containers
- Can you combine docker options?
  - run
  - rm
  - prune
  - tag



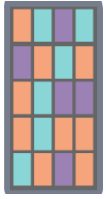
# Working interactively

- When to do it?
  - Debug
- How to do it:
  - Enter the container and interact:
    - `docker run -it <image> <command>`
    - `dokcer run -it -rm <image> <command>`
  - Execute a command in a running container
    - `dockcer exec <containerrr>`



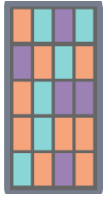
# ACTIVITY TIME:

- Activity 1.1
  - Get the data  
(<https://github.com/vibbits/containers-workshop/>)
  - You will need fastqc
    - check all your images
    - Pull image if needed
  - Fastqc, version 0.11.9\_cv7
    - Docker hub
  - Run fastqc -h
- Activity 1.2
  - Check for running containers
  - Remove the container
  - Run the container interactively
    - Start it with bash
  - docker run -it <image> <command>
  - dokcer run -it -rm <image> <command>



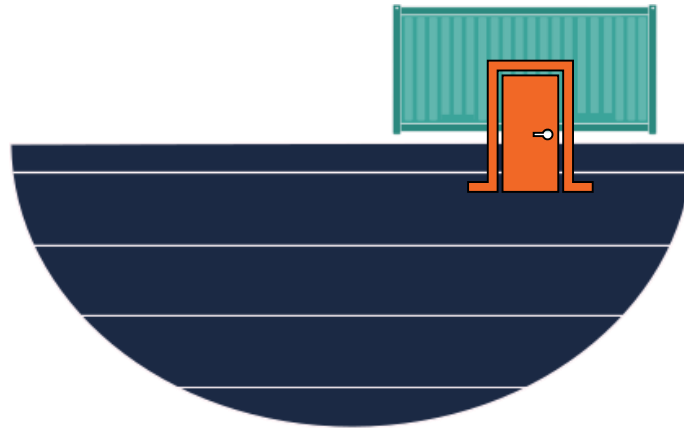
Docker a closed environment

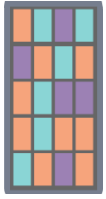




## Docker a closed environment

- Mounting volumes
- Using Ports

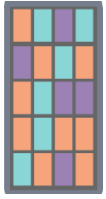




## Volume mounting: I/O

- Container is isolated from host



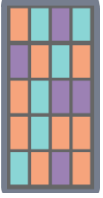


## Volume mounting: I/O

- Container is isolated from host
- Data in the container is NOT kept







# Volume mounting: I/O

- Container is isolated from host
- Data in the container is NOT kept
- Solution:

- Binding volume

**-v /path/in/host:/path/in/container**

```
docker run --detach \
```

```
--volume path/in/host/datatest:/path/in/container/dataset \
```

```
--name <container_name> <container:version> <container_options>
```

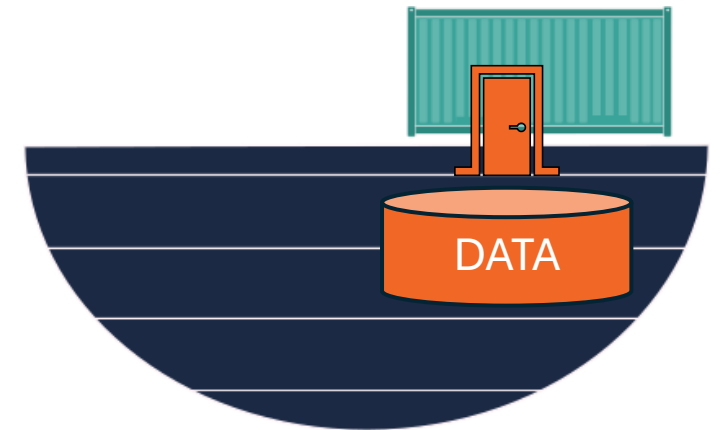
- Working directory

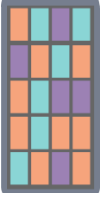
**-w /path/in/host**

```
docker run -w /path/in/host
```

```
--volume path/in/host/datatest:/path/in/container/dataset \
```

```
--name <container_name> <container:version> <container_options>
```





# ACTIVITY TIME:

- Activity 2.1

- Run interactively and mount the **local data/** folder to the **container /data**
  - `biocontainers/ fastqc:v0.11.9_cv7.`
- Remove the container after it has run

- Binding volume

**`-v /path/in/host:/path/in/container`**

- Activity 2.2

- Do a quality control on the WT samples
  - Use the command

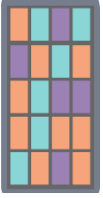
```
fastqc /data/WT_lib1_R1.fq.gz or fastqc /data/ecoli_1.fastq.gz.
```

- Why do we need to add **/data/** in the fastqc command?



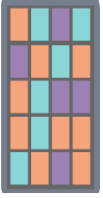
## ACTIVITY TIME: 3

- Who is the default user within the container?
- Run interactively and mount the local data/ directory to /scratch in the container
  - `biocontainers/ fastqc:v0.11.9_cv7`
- In the container directory
  - Create a temporary file `file1.txt` in the `scratch/`
- Quit the interactive session.
  - On your host, check the file permissions.
- On the host
  - Create a temporary file `file2.txt` in the `data/` directory.
- Run interactively and inspect the file permissions of this file
  - the `fastqc` container
- Check the file permissions of this file in the container (`scratch/` directory).
- On the host, find out which UID and GID you have.
  - Tip: you can find your UID and GID with: `id -u` and `id -g`.
- Run a docker container by using the `-u` parameter
  - In the meantime creating a temporary file `file3.txt` with `touch`.
- Mount your current directory to `/data`
  - Within `quay.io/biocontainers/fastqc:0.11.9--0`.
  - Check the file permissions of this file in the container.



## ACTIVITY TIME: 3

- Who is the default user within the container?
- Run interactively and mount the local data/directory to /scratch in the container
  - `biocontainers/ fastqc:v0.11.9_cv7`
- In the container directory
  - Create a temporary file `file1.txt` in the `scratch/`
- Quit the interactive session.
  - On your host, check the file permissions.
- Host:
  - `ls -ls`
  - `-rw-r--r-- 3 u0088910 u0088910 file1.txt`
- Container:
  - `ls -ls`
  - `-rw-r--r-- 3 root root file1.txt`



## ACTIVITY TIME: 4

- Exercise 4.1

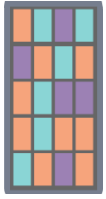
- Execute the container:  
`quay.io/biocontainers/fastqc:0.11.9_cv7`
  - Use working directory option **-w** for a directory **scratch/**
  - Create a temporary file `file4.txt` with `touch`.
  - Mount your current directory to **scratch/** within the Docker container
  - Check the file location of this file on the host.

- Exercise 4.2

- Execute the container:  
`quay.io/biocontainers/fastqc:0.11.9_cv7`
  - Use your user and group ID running
  - Do quality control of the file **WTXXX.fq.gz**.
  - mount your current directory to **the default working directory** within the Docker container
  - Verify that the HTML report is created with the correct file permissions.

- **Extra:**

- Can you analyze **all fastq** files using a glob-pattern (`WT*.fq.gz`)?
- What do you need to change to make this work?

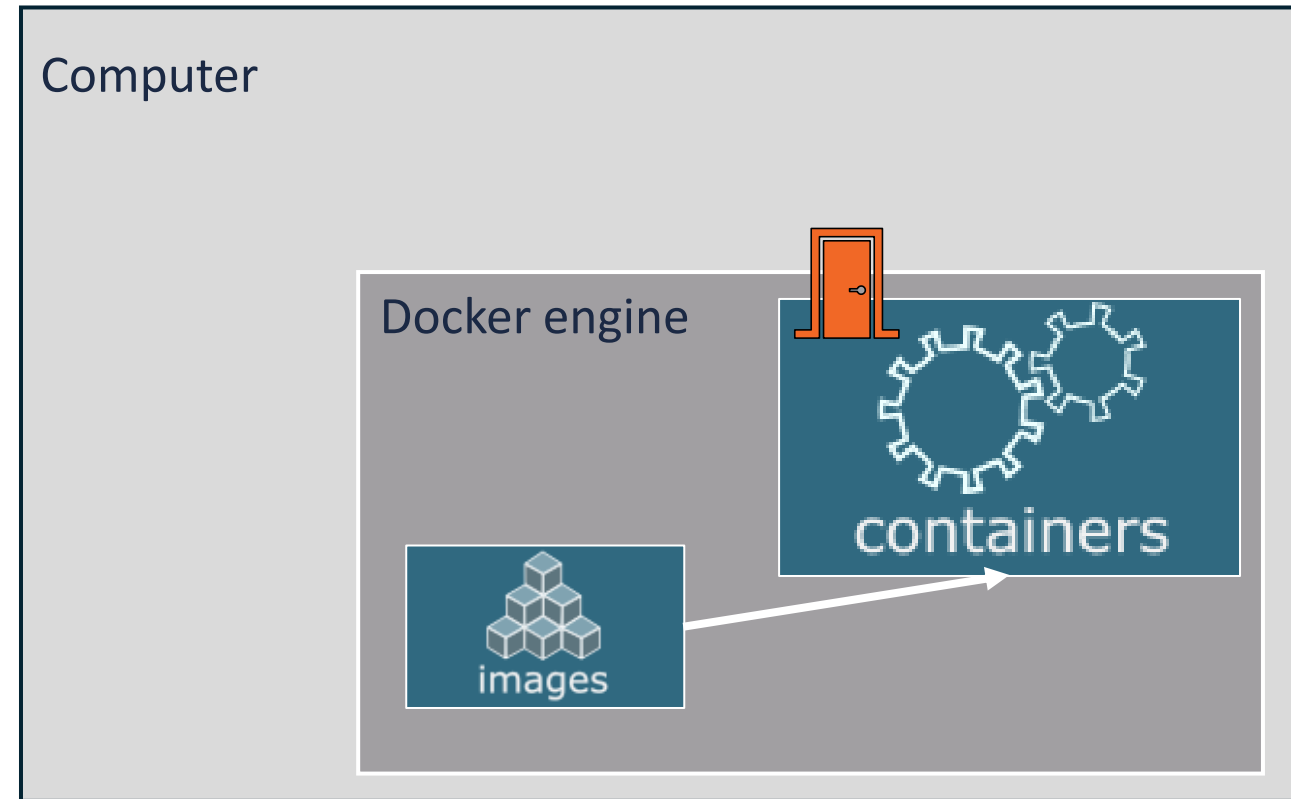


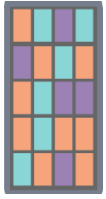
# Ports

- Establish communication with webserver

```
$ docker run --detach --name webserver nginx
```

```
$ curl localhost:80
```





# Ports

- Establish communication with webserver

```
$ docker run --detach --name webserver nginx
```

```
$ curl localhost:80
```

Nginx (Engine-x) : creates a local webserver

curl : Client URL

Enables communication between  
the host and the server

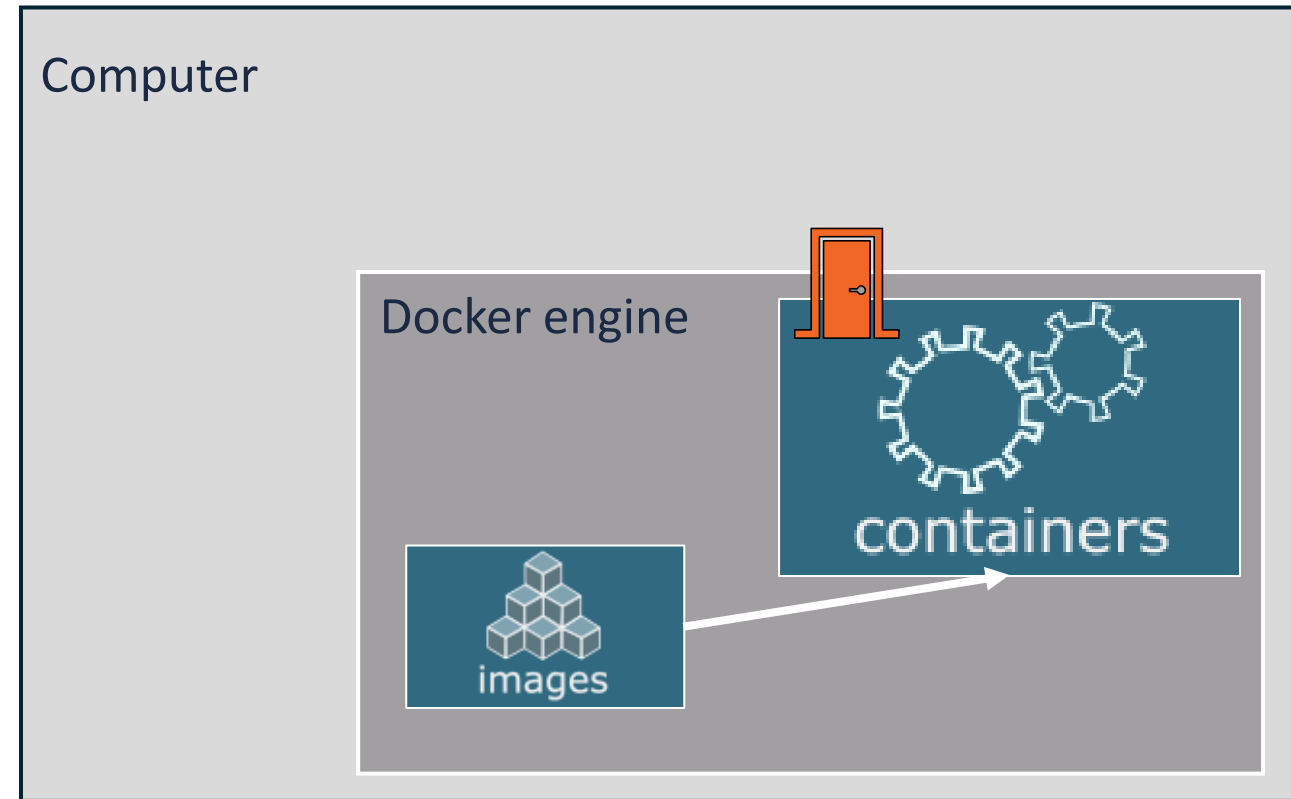




# Practice time:

```
$ docker run --detach --name webserver nginx
```

```
$ curl localhost:80
```





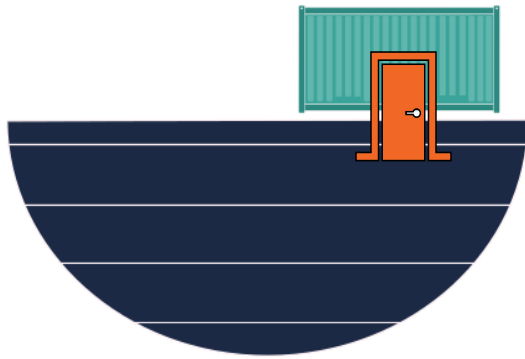
# Ports

- Establish communication with webserver

```
$ docker run --detach --name webserver nginx
```

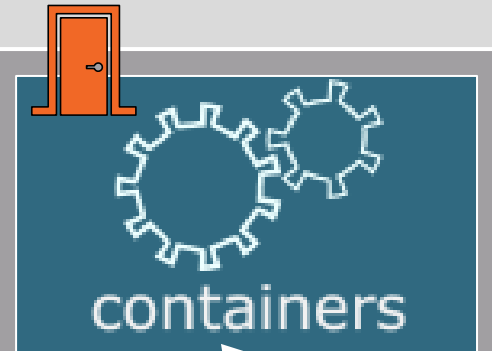
```
$ curl localhost:80
```

- Container **X** external environment



## Computer

### Docker engine



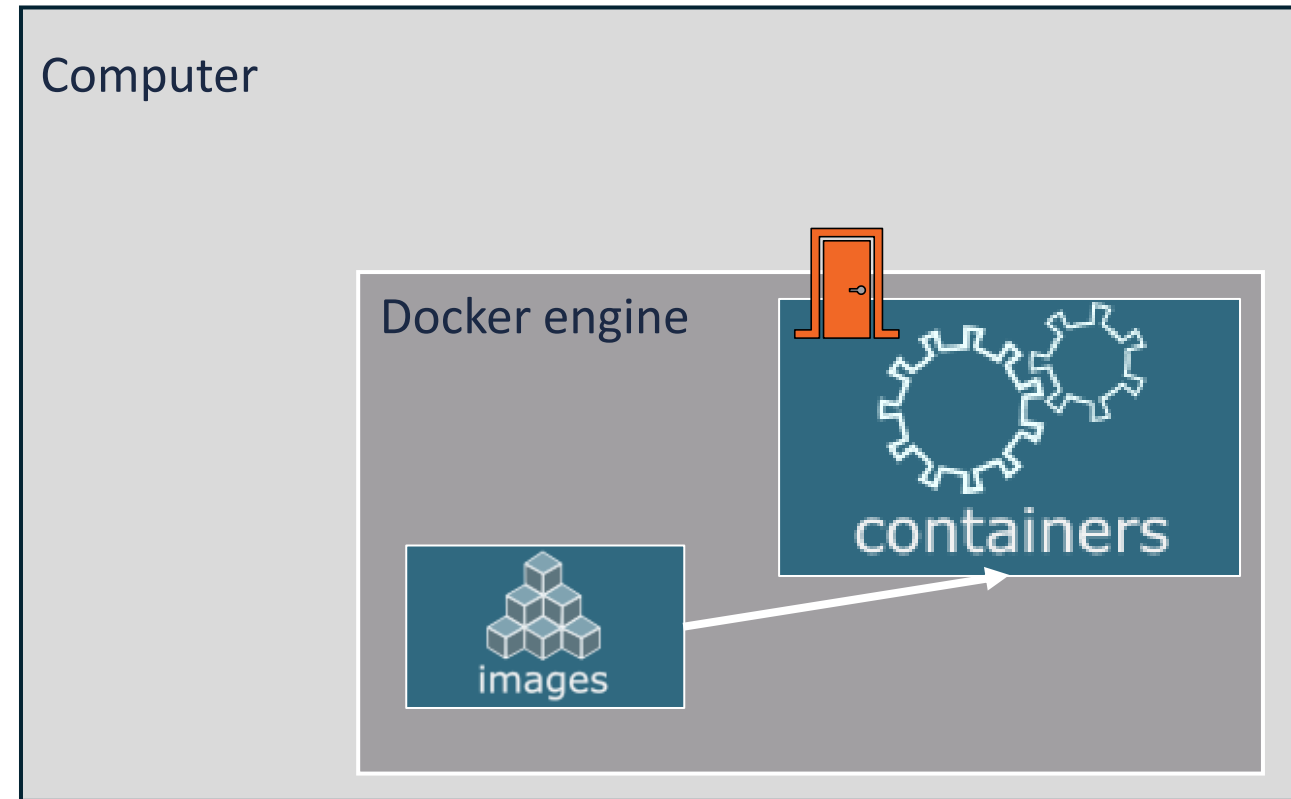


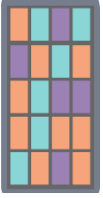
## Practice time:

```
$ docker run --detach --name webserver nginx
```

```
$ docker exec webserver curl localhost:80
```

What is the difference  
between `run` and `exec`?





# Ports

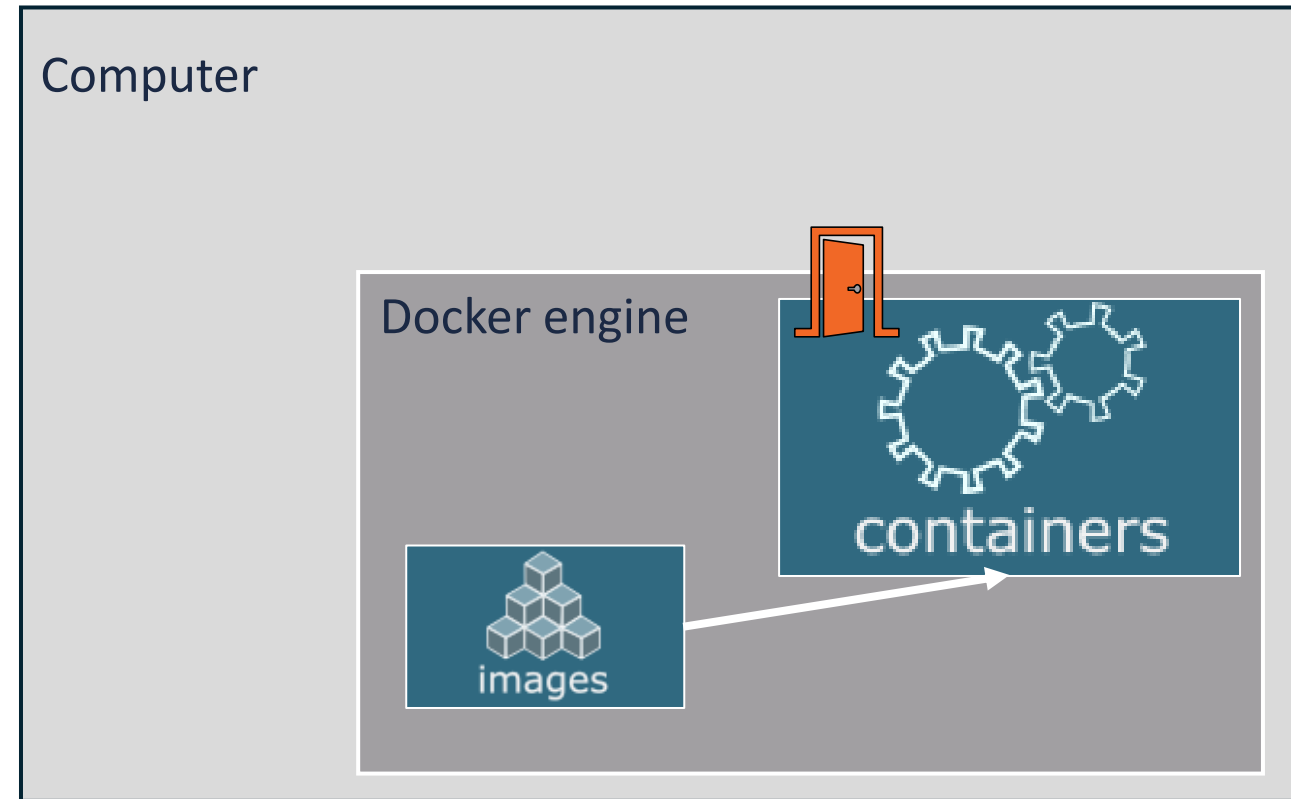
- Establish communication with webserver

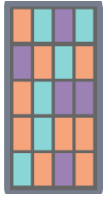
```
$ docker run --detach --name webserver nginx
```

```
$ curl localhost:80
```

- Container **X** external environment
- exec : execute inside the container
  - Open the door for host communication

```
$ docker exec webserver curl localhost:80
```



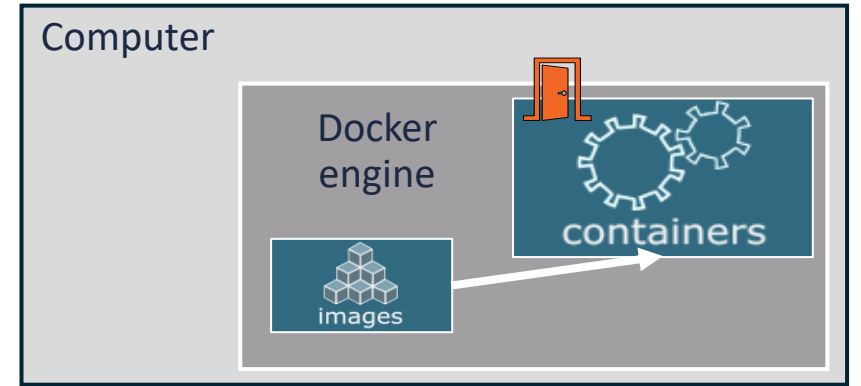


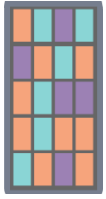
# Ports

- To keep the door open

```
$ docker run --detach --name webserver --publish 80:80 nginx
```

```
$ curl localhost:80
```





# Ports

- To keep the door open



```
$ docker run --detach --name webserver --publish 80:80 nginx
```

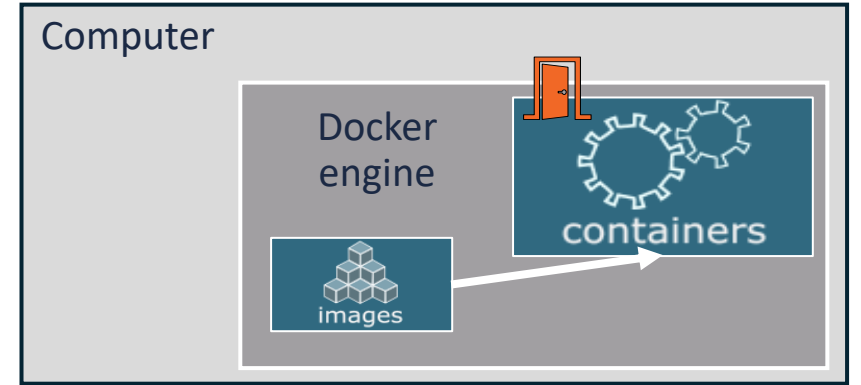
```
$ curl localhost:80
```



```
$ docker run --detach --name webserver -p 8080:80 nginx
```

```
$ curl localhost: ????
```

```
$ docker exec webserver curl localhost: ???
```

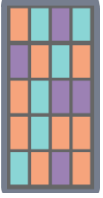


FYI

**--publish**

**=**

**-p**



# Practice time:

- What happens? Why?

```
$ docker run --detach --name webserver --publish 80:80 nginx
```

```
$ curl localhost:80
```

- What should you use? Why ?

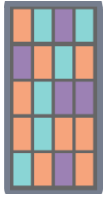
```
$ docker run --detach --name webserver -p 8080:80 nginx
```

```
$ curl localhost: ????
```

```
$ docker exec webserver curl localhost: ???
```

**Remember to remove these  
containers**

**docker rm -f <name>**

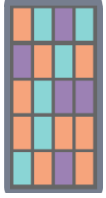


# Inspect

- Check a recipe
  - How others do
  - Potential security issues

```
$ docker inspect <image_name\image_ID>
```





# Practice time:

- Find trimmomatic in docker hub
  - dceoy/trimmomatic
- Pull and inspect

```
FROM ubuntu:latest

ENV DEBIAN_FRONTEND noninteractive

ADD https://github.com/timflutre/trimmomatic/archive/master.tar.gz /tmp/trimmomatic.tar.gz

RUN set -e \
 && ln -sf bash /bin/sh

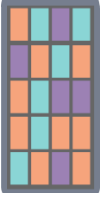
RUN set -e \
 && apt-get -y update \
 && apt-get -y dist-upgrade \
 && apt-get -y install --no-install-recommends --no-install-suggests \
 default-jdk make \
 && apt-get -y autoremove \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

RUN set -e \
 && tar xvf /tmp/trimmomatic.tar.gz -C /opt --remove-files \
 && mv /opt/trimmomatic-* /opt/trimmomatic \
 && cd /opt/trimmomatic \
 && make

RUN set -e \
 && mkdir /opt/trimmomatic/bin \
 && echo '#!/usr/bin/env bash' > /opt/trimmomatic/bin/trimmomatic \
 && echo 'java -jar /opt/trimmomatic/classes/trimmomatic.jar ${@}' \
 >> /opt/trimmomatic/bin/trimmomatic \
 && chmod +x /opt/trimmomatic/bin/trimmomatic

ENV PATH /opt/trimmomatic/bin:${PATH}

ENTRYPOINT ["/usr/bin/java", "-jar", "/opt/trimmomatic/classes/trimmomatic.jar"]
```

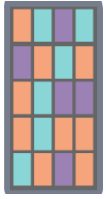


## ACTIVITY TIME: 4.3 + challenge

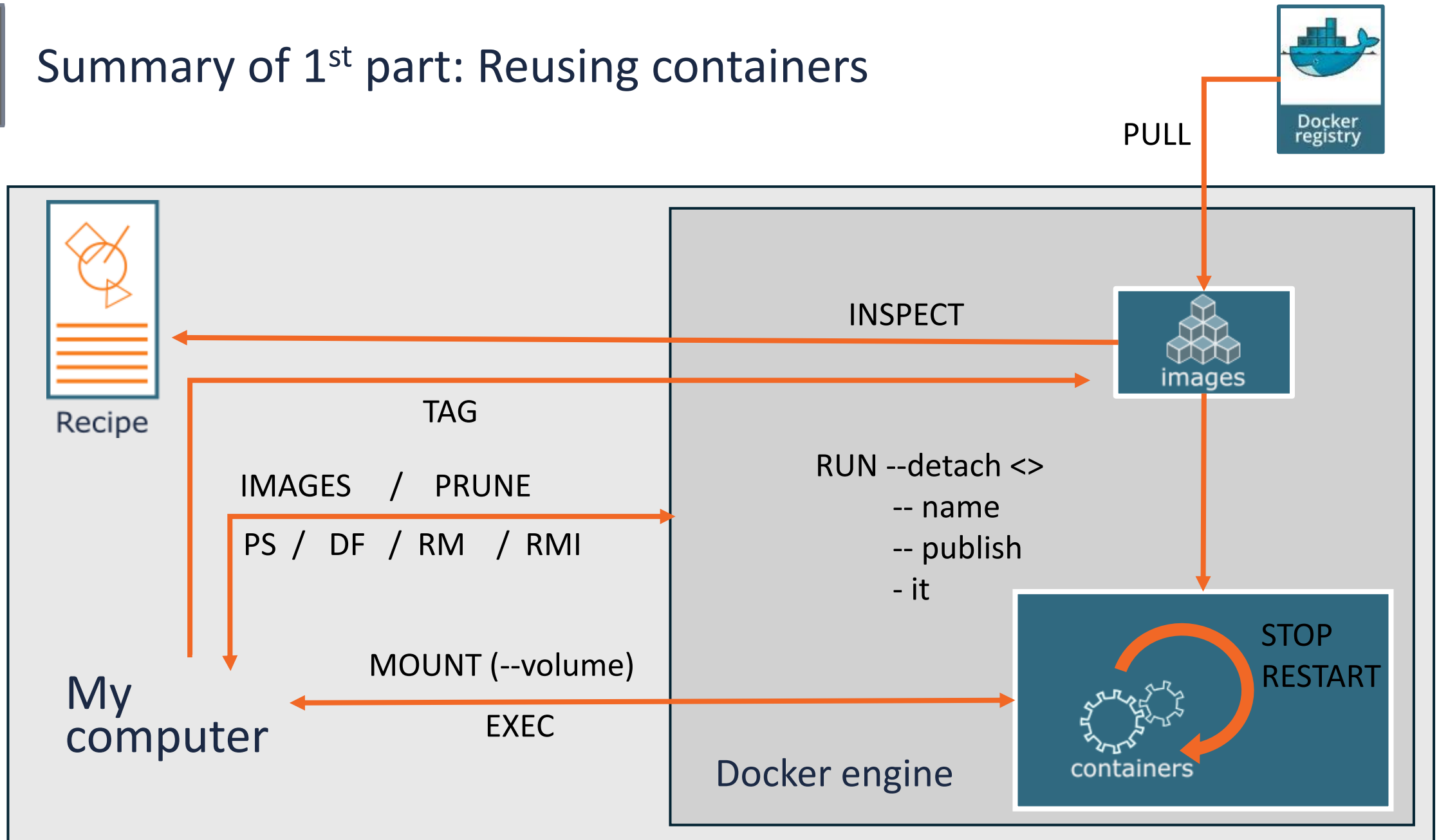
- Inspect the image  
biocontainers/fastqc:0.11.9\_cv7
- Extract the working directory (WorkingDir) using  
grep.

HINT:

```
cmd | grep "keyword"
```



## Summary of 1<sup>st</sup> part: Reusing containers





An aerial photograph of a large port facility. In the foreground and middle ground, there are massive stacks of colorful shipping containers in various colors like orange, blue, red, and green. Several large yellow gantry cranes are positioned along the waterfront, used for loading and unloading ships. The port is situated next to a body of water, with a long pier extending into the sea. The sky is clear and blue.

# Building your own docker image



## Recipes

- The default recipe is called Dockerfile

```
FROM ubuntu:18.04
```

```
RUN apt update && apt -y upgrade
```

```
RUN apt install -y wget
```



Recipe





## Practice time: Building images

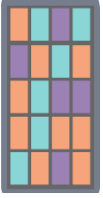
- Create a Dockerfile with the content below in a folder of your preference and save it.

```
FROM ubuntu:18.04
```

```
RUN apt update && apt -y upgrade
```

```
RUN apt install -y wget
```

- How many layers should be created?



## Practice time: Building images

- Let's test it and build the image with  
docker build .
- How many layers should be created?

docker history <id>





## Practice time: Building images

- Create a Dockerfile with the content below in a folder of your preference and save it.

```
FROM ubuntu:18.04
```

```
LABEL org.opencontainers.image.authors="training@vib.be"
```

```
WORKDIR /
```

```
RUN apt update && apt -y upgrade
```

```
RUN apt install -y wget
```

```
ENTRYPOINT ["/usr/bin/wget"]
```

```
CMD ["https://cdn-images-
1.medium.com/max/1600/1*_NQN6_YnxS29m8vFzWYlEg.png"]
```

| Layers                                                      | Cache? |
|-------------------------------------------------------------|--------|
| FROM ubuntu:latest                                          | ✓      |
| RUN apt-get update \<br>&& apt-get install build-essentials | ✓      |
| COPY main.c Makefile /src/                                  | ✗      |
| WORKDIR /src                                                | ✗      |
| RUN make build                                              | ✗      |

Build the image with and without caching.



Recipes



Recipe



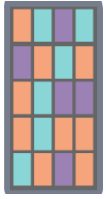
## More advanced image building

- Different ways to build images.
- Know your base system and their packages. Popular ones:
  - Debian
  - CentOS
  - Alpine
  - Conda. Anaconda, Conda-forge, Bioconda, etc.

| command    | what does it do?                                                                                                 |
|------------|------------------------------------------------------------------------------------------------------------------|
| LABEL      | Who is maintaining the container image                                                                           |
| WORKDIR    | all subsequent actions will be executed in that working directory.                                               |
| COPY       | lets you copy a local file or directory from your host (the machine from which you are building the image)       |
| ADD        | same, but ADD works also for URLs, and for .tar archives that will be automatically extracted upon being copied. |
| ARG        | available only while the image is built                                                                          |
| ENV        | available for the future running containers                                                                      |
| ENTRYPOINT | The ENTRYPOINT specifies a command that will always be executed when the container starts.                       |
| CMD        | The CMD specifies arguments that will be fed to the ENTRYPOINT.                                                  |

# What do you expect your container to do?

Makes easier to maintain the container.



## Recipes



Recipe



## One tool, one image or some tools, one image

- Different ways to build images.
- start from packages e.g. [pip/PyPI](#), [CPAN](#), or [CRAN](#)
- use versions for tools and images
- reduce size as much as possible
- keep data outside the image/container
- check the license
- make your container discoverable e.g. biocontainers, quay.io, docker hub

•Published: November 10, 2020  
•<https://doi.org/10.1371/journal.pcbi.1008316>

# Ten simple rules for writing Dockerfiles for reproducible data science

•Daniel Nüst ,  
•Vanessa Sochat,  
•Ben Marwick,  
•Stephen J. Eglén,  
•Tim Head,  
•Tony Hirst,  
•Benjamin D. Evans

Ten Simple Rules for Writing Dockerfiles for Reproducible Data Science

1

Use available tools

2

Build upon existing images

3

Format for clarity

4

Document within the Dockerfile

5

Specify software versions

6

Use version control

7

Mount datasets at run time

8

Make the image one-click runnable

9

Order the instructions

10

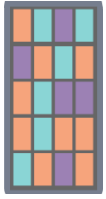
Regularly use and rebuild containers

© Benjamin D. Evans, 2020. This work is released under the CC-BY 4.0 license <https://creativecommons.org/licenses/by/4.0/>



Recipe





## ACTIVITY TIME:

[https://github.com/vibbits/containers-workshop/blob/main/exercises/02\\_building\\_dockerfiles.md](https://github.com/vibbits/containers-workshop/blob/main/exercises/02_building_dockerfiles.md)

- Exercise 5.2
  - You need to pull jupyter/scipy-notebook:python-3.11.5
  - Once you have it, run the script from outside the container  
***codereppy\_min\_batch.py***

Reach out to your neighbour(s) in case you need help.

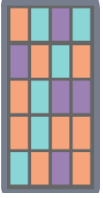
```
docker run --detach --rm -v ./data:/home/jovyan/ jupyter/scipy-notebook:python-3.11.5 python ./data/codereppy_min_batch.py
```

```
docker run --detach --rm -w /data -v ./data:/data jupyter/scipy-notebook:python-3.11.5 python ./data/
```

```
docker run --detach --rm -w /data -v ./data:/data jupyter/scipy-notebook:python-3.11.5 python codereppy_min_batch.py
```

```
docker run -it --rm -w /data -v ./data:/data jupyter/scipy-notebook:python-3.11.5 bash || python codereppy_min_batch.py
```

```
docker run -it -u root -w /data -v ./data:/data jupyter/scipy-notebook:python-3.11.5 bash || python codereppy_min_batch.py
```



## ACTIVITY TIME:

[https://github.com/vibbits/containers-workshop/blob/main/exercises/02\\_building\\_dockerfiles.md](https://github.com/vibbits/containers-workshop/blob/main/exercises/02_building_dockerfiles.md)

- Exercise 6
  - You “found” a Dockerfile on github – now in the docker folder
    - Dockerfile.deploy-cloud
  - Try to build the image
  - Run the Docker image – what does the container do?
  - Test the container
  - You need the Python script test\_query.py which is in the data folder

Reach out to your neighbour(s) in case you need help.





## Recipes



Recipe



## One tool, one image or some tools, one image

- Different ways to build images.
- start from packages e.g. [pip/PyPI](#), [CPAN](#), or [CRAN](#)
- use versions for tools and images
- reduce size as much as possible
- keep data outside the image/container
- check the license
- make your container discoverable e.g. biocontainers, quay.io, docker hub



# Short intro to HPC

## preparing for Apptainer



# Ugent : Ondemand - <https://login.hpc.ugent.be>

HPG UGent

Files

Jobs

Clusters


Interactive Apps

My Interactive Sessions

Clusters

>\_ RHEL8 Login node Shell Access

>\_ RHEL9 Login node Shell Access



OnDemand provides an integrated, sin

UGent TIER1 clusters

| Cluster name | Memory (GiB) | Disk space        | GPU                       |
|--------------|--------------|-------------------|---------------------------|
| swalot       | 116          | 1 TB              | -                         |
| skitty       | 177          | 1 TB + 240 GB SSD | -                         |
| victini      | 88           | 1 TB + 240 GB SSD | -                         |
| joltik       | 256          | 800 GB SSD        | 4 NVIDIA V100             |
| doduo        | 250          | 180 GB SSD        | -                         |
| accelgor     | 500          | 180 GB SSD        | 4 NVIDIA A100             |
| donphan **   | 738          | 1.6 TB NVME       | 1 shared NVIDIA Ampere A2 |
| gallade      | 940          | 1.5 TB NVME       | -                         |





# VSC - UGent

## Filesystems specifics

| Filesystem name                             | Intended usage                                                   |
|---------------------------------------------|------------------------------------------------------------------|
| \$VSC_HOME                                  | Home directory, Not the entry point to the system, same as Tier2 |
| \$VSC_SCRATCH                               | Entry point to the system                                        |
| \$VSC_DATA                                  | Long-term storage of large data files                            |
| /dodrio/scratch/projects/starting_2023_001/ | Temporary fast storage of 'live' data for calculations           |

Usually where you arrive when login in  
Not much space  
Not for running analysis

More space  
Where you run analysis (input/output)

Long term storage  
Archive also  
Large files

\\*\* Storage space for a group of users (Virtual Organisation or VO for short) can



# UGent TIER1

| Filesystem name                             | Intended usage                                                   | Total storage space | Personal storage space                        | VO storage space ** |
|---------------------------------------------|------------------------------------------------------------------|---------------------|-----------------------------------------------|---------------------|
| \$VSC_HOME                                  | Home directory, Not the entry point to the system, same as Tier2 | ?                   | 3GB (fixed)                                   | ✗                   |
| \$VSC_SCRATCH                               | Entry point to the system                                        | ?                   | 3GB (fixed)                                   | ✗                   |
| \$VSC_DATA                                  | Long-term storage of large data files                            | ?                   | Depend of you account(Leuven/Gent, see above) | ✗                   |
| /dodrio/scratch/projects/starting_2023_001/ | Temporary fast storage of 'live' data for calculations           | ?                   | 10TB                                          | ?                   |

\\*\* Storage space for a group of users (Virtual Organisation or VO for short) can be increased significantly on request.

•Source : [https://docs.vscentrum.be/gent/tier1\\_hortense.html#system-specific-aspects](https://docs.vscentrum.be/gent/tier1_hortense.html#system-specific-aspects)



# UGent TIER2

| Filesystem name                                                                                                          | Intended usage                                                                                        | Total storage space | Personal storage space | VO storage space (*) |
|--------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|---------------------|------------------------|----------------------|
| \$VSC_HOME                                                                                                               | Home directory, entry point to the system                                                             | 51 TB               | 3GB (fixed)            | ✗                    |
| \$VSC_DATA                                                                                                               | Long-term storage of large data files                                                                 | 1.8 PB              | 25GB (fixed)           | 250GB                |
| \$VSC_SCRATCH                                                                                                            | Temporary fast storage of 'live' data for calculations                                                | 1.9 PB              | 25GB (fixed)           | 250GB                |
| \$VSC_SCRATCH_ARCANINE                                                                                                   | Temporary very fast storage of 'live' data for calculations (recommended for very I/O-intensive jobs) | 70 TB               | (none)                 | upon request         |
| (*) Storage space for a group of users (Virtual Organisation or VO for short) can be increased significantly on request. |                                                                                                       |                     |                        |                      |

•Source : [https://docs.vscenrum.be/en/latest/gent/tier2\\_hardware.html?highlight=VSC\\_DATA#shared-storage](https://docs.vscenrum.be/en/latest/gent/tier2_hardware.html?highlight=VSC_DATA#shared-storage)



# KULeuven TIER2

## Filesystems specifics

| Filesystem name | Intended usage                                         | Total storage space | Personal storage space | VO storage space (*) |
|-----------------|--------------------------------------------------------|---------------------|------------------------|----------------------|
| \$VSC_HOME      | Home directory, entry point to the system              | ?                   | 3GB (fixed)            | ✗                    |
| \$VSC_DATA      | Long-term storage of large data files                  | ?                   | 75GB (fixed)           | ✗                    |
| \$VSC_SCRATCH   | Temporary fast storage of 'live' data for calculations | ?                   | 500GB                  | ?                    |

•Source [https://docs.vscentrum.be/en/latest/leuven/tier2\\_hardware/kuleuven\\_storage.html?highlight=VSC\\_DATA#ku-leuven-storage](https://docs.vscentrum.be/en/latest/leuven/tier2_hardware/kuleuven_storage.html?highlight=VSC_DATA#ku-leuven-storage)



# KULeuven – TIER2 clusters

## Clusters specifics at UGent - VSC

---

On top of the filesystem, each clusters will have different computational powers, therefore, depending on your needs, you can choose the one that most suits you.

| Cluster name | Memory (GiB) | Disk space        | GPU                       |
|--------------|--------------|-------------------|---------------------------|
| swalot       | 116          | 1 TB              | -                         |
| skitty       | 177          | 1 TB + 240 GB SSD | -                         |
| victini      | 88           | 1 TB + 240 GB SSD | -                         |
| joltik       | 256          | 800 GB SSD        | 4 NVIDIA V100             |
| doduo        | 250          | 180 GB SSD        | -                         |
| accelgor     | 500          | 180 GB SSD        | 4 NVIDIA A100             |
| donphan **   | 738          | 1.6 TB NVME       | 1 shared NVIDIA Ampere A2 |
| gallade      | 940          | 1.5 TB NVME       | -                         |

\*\* [debugging cluster \(Used for debugging and training\)](#)

Using KULeuven ondemand:

<https://ondemand.hpc.kuleuven.be>





## VSC - UGent

### Submission command

- `qsub submission_script.sh`
- `sbacth submission_script.sh`

### Commands in the HPC

- `module avail` : list all available modules
- `module spider <key word>` : search specific modules
- `module swap cluster/donphan` : Swap to specific cluster

```
#!/bin/bash
#SBATCH --job-name= Apptainer-donphan
#SBATCH --output= raboti_vistro.out
#SBATCH --ntasks= 1
#SBATCH --time= 1:00:00
#SBATCH --mem-per-cpu= 16G
#SBATCH --partition= cluster/dodrio/gpu_rome_a100
#SBATCH --gres= gpu:1
#SBATCH --cpus-per-task= 16
#SBATCH --error= raboti_vistro.%j.err
#SBATCH --account= starting_2023_001

module swap cluster/dodrio/gpu_rome_a100
module load scikit-image/0.19.3-foss-2022a
module load n2v/0.3.2-foss-2022a-CUDA-11.7.0

python
/dodrio/scratch/users/vsc33625/00_02_prediction_d
enoising_batch.py
```

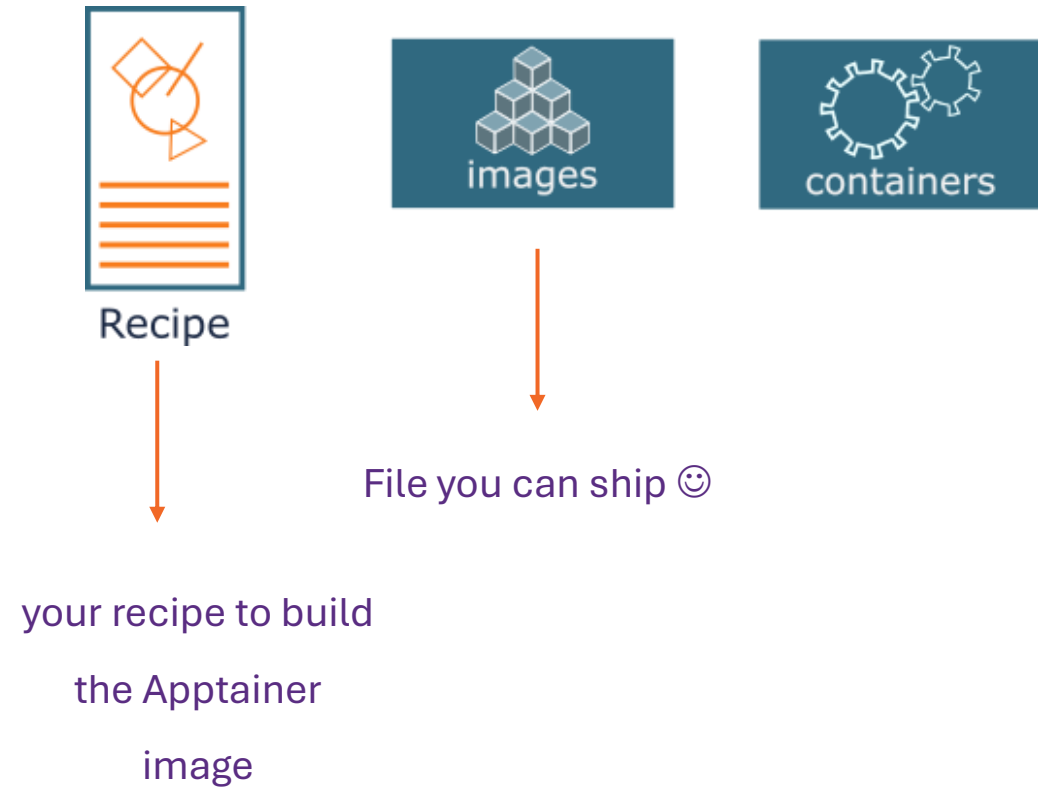


# Introduction to Apptainer for reproducible analysis



# How does it work?

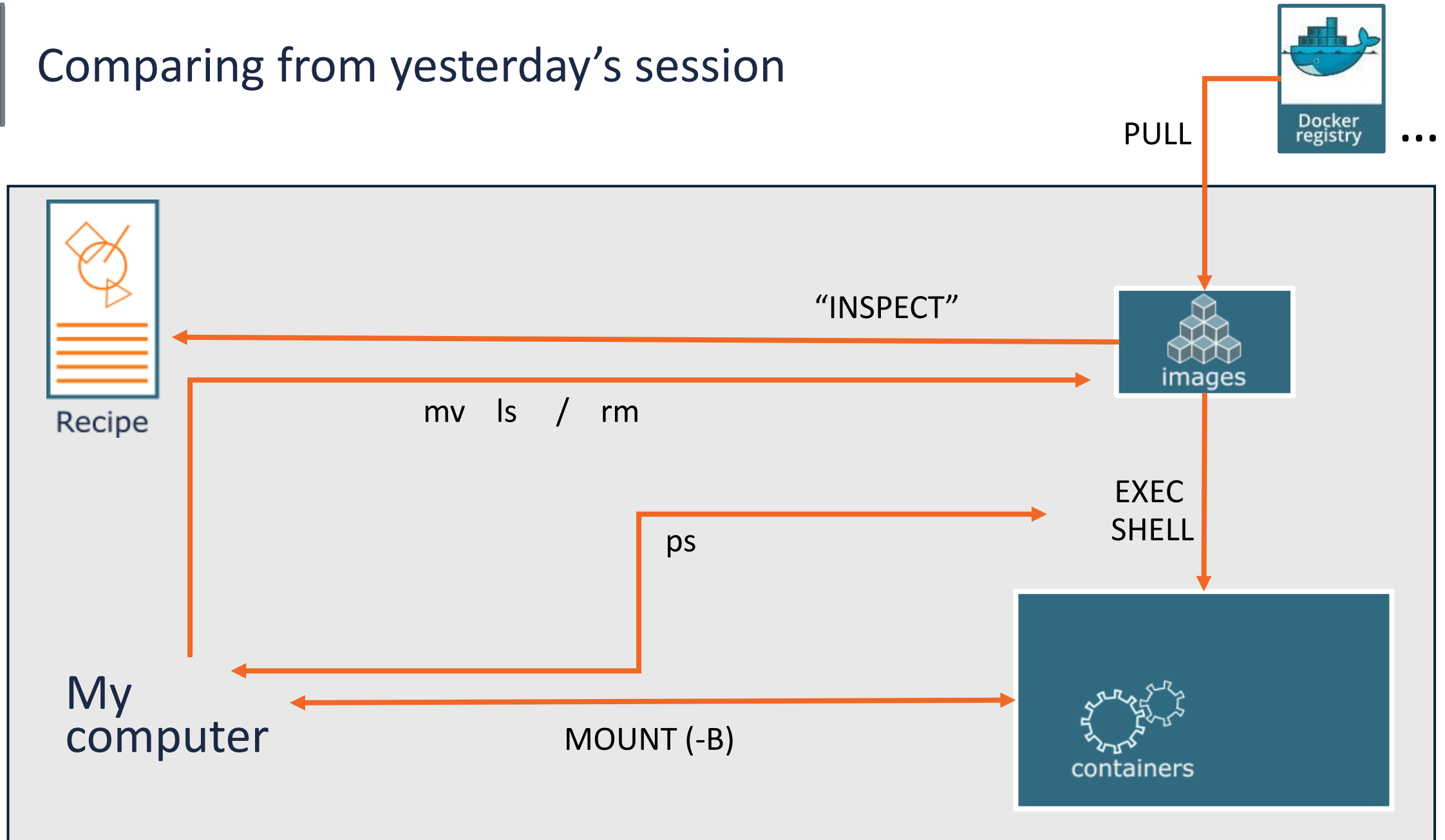
- Important concepts
  - **Text file:** Your recipe
  - **Apptainer image:** Static file
  - **Container:** Running image (functional)







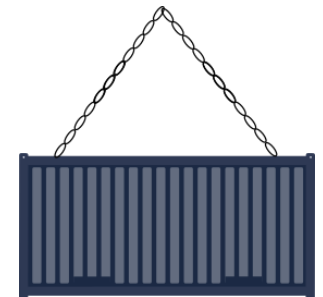
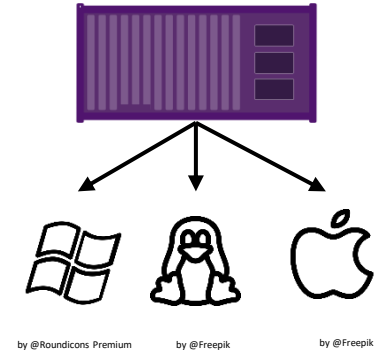
## Comparing from yesterday's session

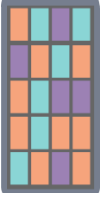




# Docker vs Apptainer

- **Strengths**
- No dependency of a daemon
- Can be run as a simple user
  - Avoid permission headaches and hacks
- Image/container is a file (or directory)
- More easily portable
- Two type of images
  - Read-only (production)
  - Writable (development, via sandbox)
- **Weaknesses**
- At the time of writing only good support in Linux
- For some features you need root account (or sudo)





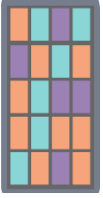
## How does it work?

- By default, Apptainer uses `$HOME/.apptainer/cache` as the location for the cache.

This will not work on the VSC.

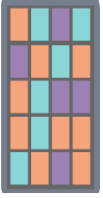
You can change the location of the cache by setting the `$APPTAINER_CACHEDIR` environment variable to the cache location you want to use.

- Please set the variable `$APPTAINER_CACHEDIR` to `$VSC_SCRATCH`.



# Practice time: 1

- **Exercise 1**
  - In the hello world container, try editing (for example using the editor vi which should be available in the container) the /rawr.sh file. What do you notice?
- **Exercise 2:**
  - In your home directory within the container shell, try and create a simple text file. Is it possible to do this? If so, why? If not, why not?! If you can successfully create a file, what happens to it when you exit the shell and the container shuts down?

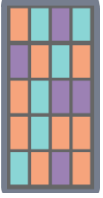


## Practice time: Downloading images

- `$ mkdir $VSC_DATA/apptainer-course`
- `$ cd $VSC_DATA/apptainer-course`
- `$ apptainer pull hello-world.sif shub://vsoch/hello-world`

```
$ apptainer pull --name fastqc-0.11.9--0.sif https://depot.galaxyproject.org/singularity/fastqc:0.11.9--0
$ file fastqc-0.11.9--0.cif
```





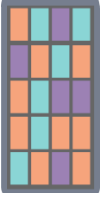
## Practice time: Binding folders

- singularity shell [-B /data/leuven/315/vsc315XX] hello-world.sif

```
Singularity> ls /data/leuven/315/vsc315XX
```

- \$ singularity shell -B /data/leuven/315/vsc315XX :/shared-data hello-world.sif

```
Singularity> ls /shared-data
```



## Practice time: Downloading images or pulling or building images

- Pulling images may take a while, so we need to run this as a job.

- `qsub pull-image.pbs`

- ...

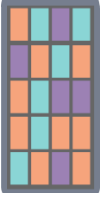
```
APPTAINER_CACHEDIR=$VSC_SCRATCH \
```

```
APPTAINER_TMPDIR=$VSC_SCRATCH \
```

```
apptainer build --fakeroot $VSC_SCRATCH/tensorflow-23.06-tf2-py3.sif \
```

```
docker://nvcr.io/nvidia/tensorflow:23.06-tf2-py3
```

```
...
```



## Practice time: And now really building images

- Pulling images may take a while, so we need to run this as a job.

- `qsub build-image.pbs`

- ...  
`APPTAINER_CACHEDIR=$VSC_SCRATCH \`  
`APPTAINER_TMPDIR=$VSC_SCRATCH \`  
`apptainer build --fakeroot $VSC_SCRATCH/test_image_ubuntu.sif \`  
`$VSC_SCRATCH/test_image_ubuntu.def`  
...