Getting started with containers

Managing containers and images

Two important concepts

Image ⟵ Container

Zutaten

Hauptgericht
Für 4 Personen

250 g	Brot, z. B. Weggli
2 dl	Bouillon
250 g	junger Spinat
2	Eier
5 EL	Paniermehl
50 g	Reibkäse, z.B. Greyerzer
	Salz
1	Schalotte
2	Knoblauchzehen
400 g	gemischte Pilze, z. B. Eierschwämmchen und Steinpilze
½ Bund	Thymian
50 g	Butter
	rosa Pfeffer
20 g	Brunnenkresse

So gehts

Zubereitung: ca. 40 Minuten
Ruhen lassen: ca. 30 Minuten
Gesamt: 1 Std. 10 Min.

1

Brot in Würfelchen schneiden und in Bouillon einweichen. Spinat waschen und tropfnass in eine Pfanne geben. Erhitzen, bis er zusammenfällt. Mit kaltem Wasser abschrecken. Spinat abgiessen und gut ausdrücken. Spinat und Eier mit einem Stabmixer pürieren. Mit Paniermehl und Käse zur Brotmasse geben und gut verkneten. Mit Salz abschmecken. Masse ca. 30 Minuten ruhen lassen. Aus der Masse mit nassen Händen Knödel à ca. 50 g formen.

FAST FERTIG

2

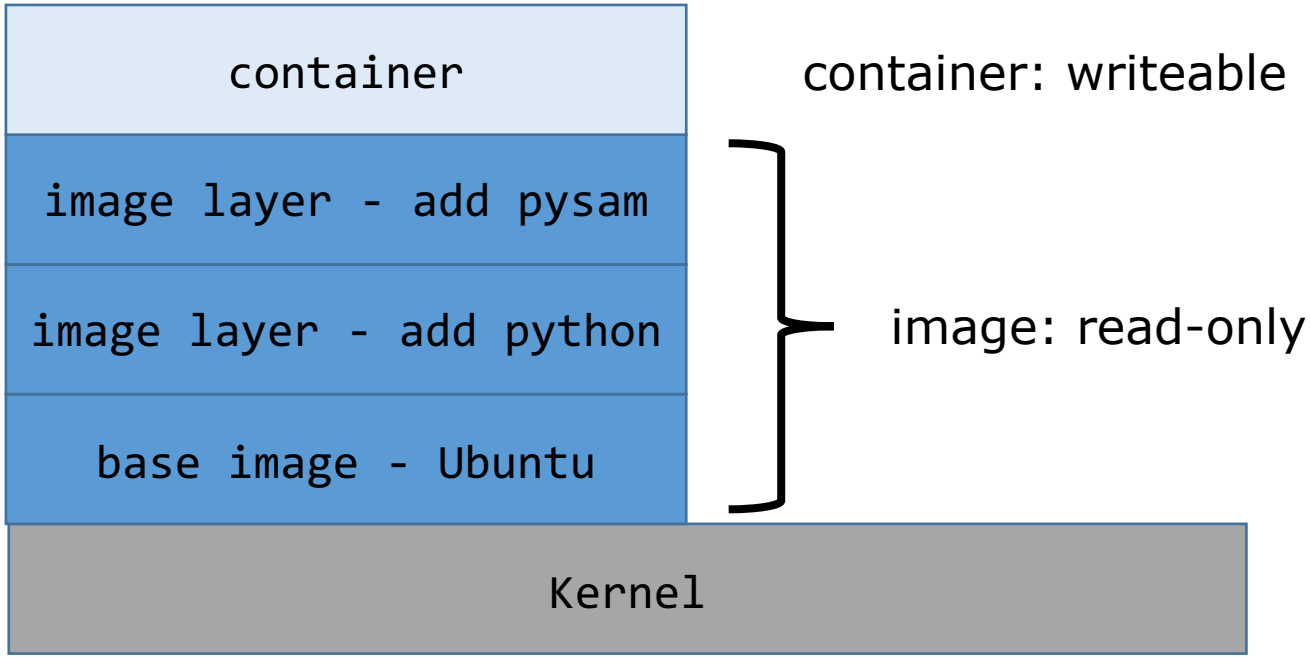
Inzwischen Schalotte und Knoblauch hacken. Pilze rüsten und nach Belieben klein schneiden. Kräuterblättchen abzupfen. Reichlich Wasser in einer weiten Pfanne aufkochen. Knödel portionenweise darin knapp unter dem Siedepunkt ca. 5 Minuten ziehen lassen, bis sie an der Oberfläche schwimmen und fest werden. Herausnehmen und abtropfen lassen. Butter in einer weiten, beschichteten Bratpfanne erhitzen. Schalotte, Knoblauch und Pilze darin bei mittlerer Hitze ca. 3 Minuten braten. Knödel dazugeben und kurz mitbraten. Mit Thymianblättchen und Pfeffer bestreuen. Mit Brunnenkresse servieren.

Rezept: Daniel Tinembart

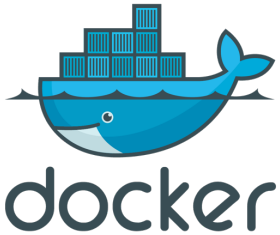
- read-only
- stored on longer term
- can be used as a base

- based on the image
- short-lived
- usually only minor adjustments

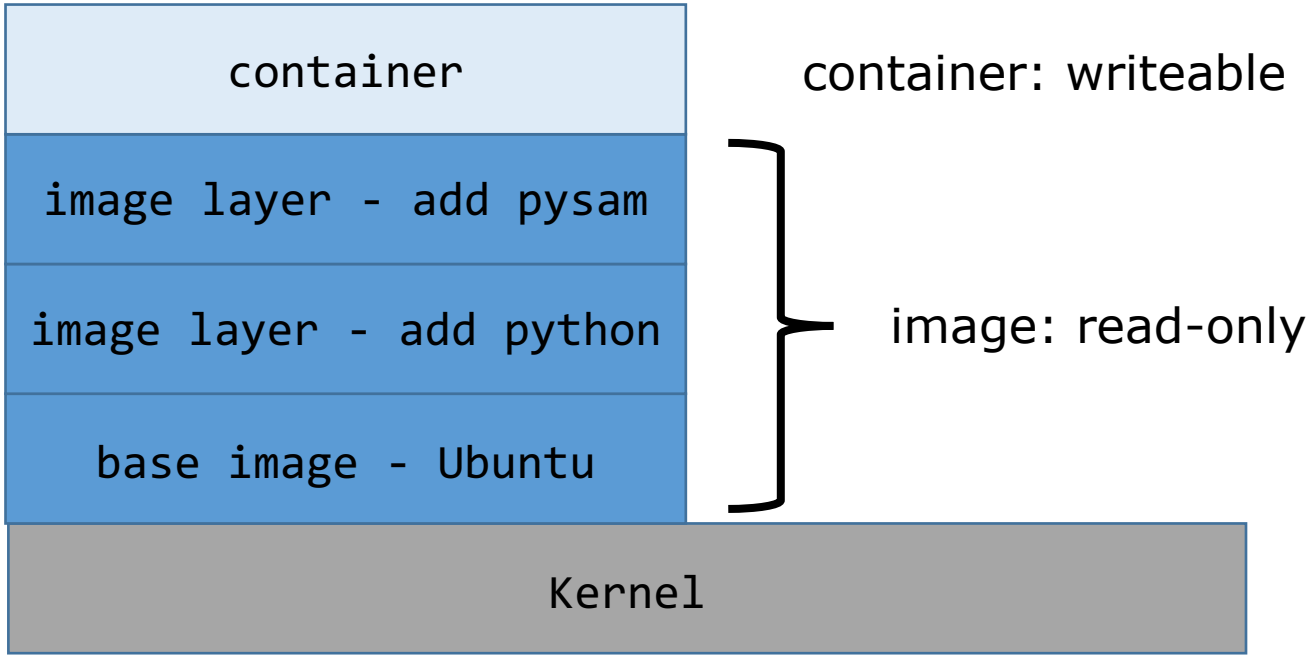
The concept of layers



Creating an image

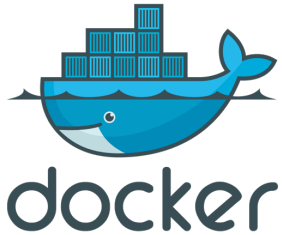


- From a Dockerfile
- From a container: `docker commit` (not reproducible)

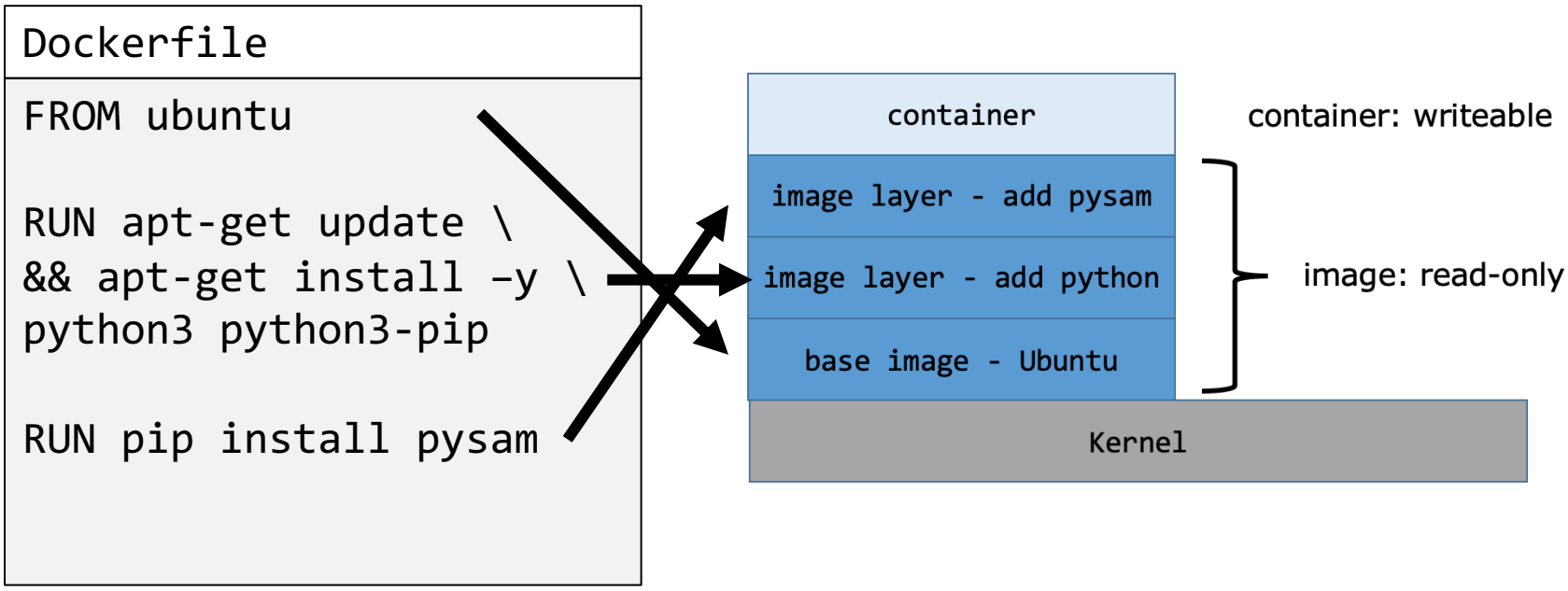


Quiz question 6

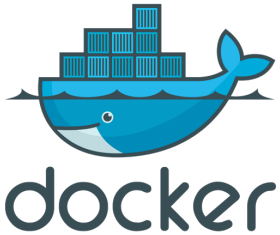
Dockerfiles



- Set of instructions on how to add layers to an image
- Build with `docker build`

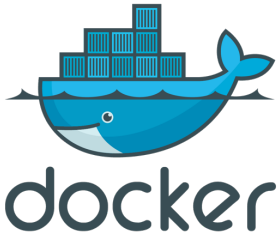


The docker engine



- Manages in a daemon process:
  - images
  - containers
- Layers are efficiently handled:
  - caching
  - re-use
- User interaction **not** through files -> through the CLI

The docker engine



Interaction through **command line interface** (or GUI), e.g:

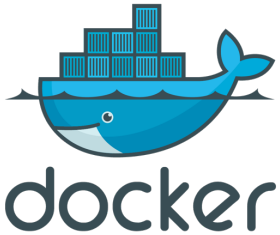
docker image ls

```
REPOSITORY                          TAG       IMAGE ID       CREATED         SIZE
lecture_basic                       latest    44465faa1426   43 hours ago    515MB
geertvangeest/adv_singlecell_2022   latest    fc616eb35cf6   2 days ago      2.97GB
own_script                          latest    35ee5b2f74c1   2 days ago      1.04GB
geertvangeest/ngs-variants-jupyter  2022.3    c3d451753035   2 weeks ago     3.54GB
```

docker container ls

```
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS       PORTS   NAMES
6420ae80cc6c   ubuntu    bash      6 seconds ago   Up 5 seconds         blissful_tharp
```

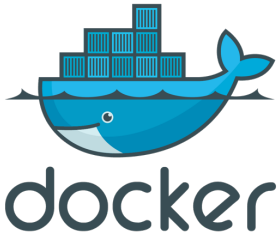
Sharing an image



- docker hub (open to the world)
  - Command: `docker push`
  - Alternatives: quay.io, gitlab and github container repositories, AWS/Google cloud/Azure …
- command `docker save`
- `Dockerfile`

Question 7

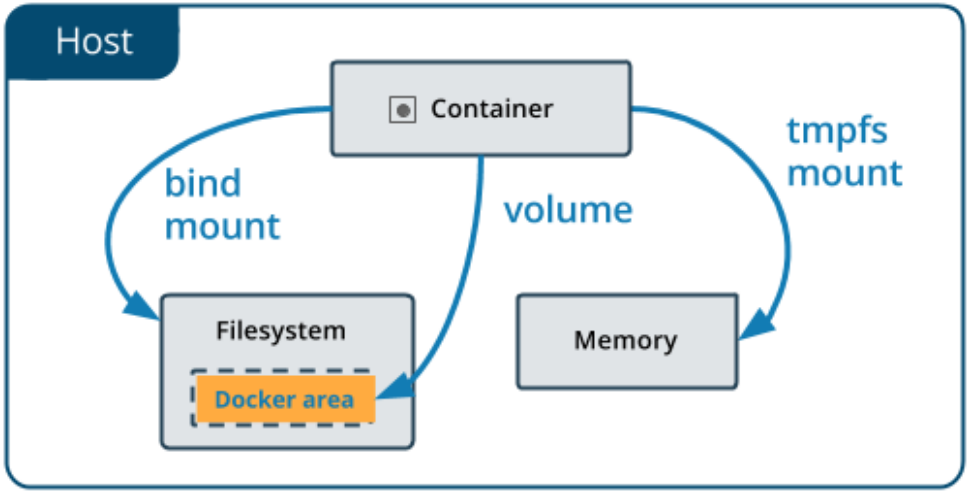
3 frequently used features



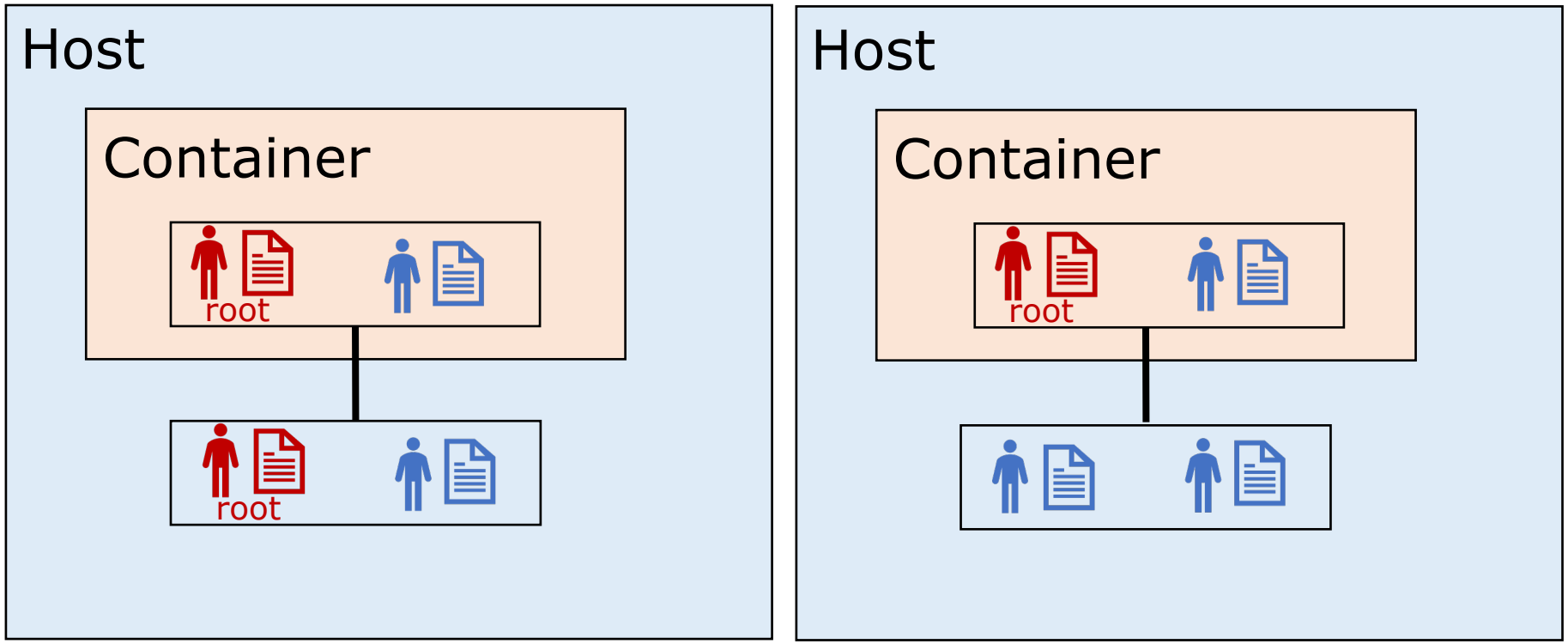
1. Mounting directories
2. Managing identities
3. Mapping ports

Mounting

- **Bind-mount:** Make a directory on the host available to the container
- **Volume:** Disk space reserved and managed by docker (isolated)



Identity

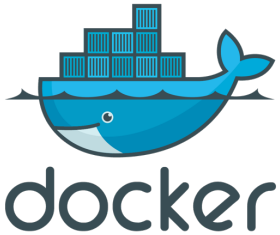


Linux

Other systems

```
docker run -u "$(id -u):$(id -g)"
```

Mapping ports



- Processes that display browser content:
  - Jupyter
  - Rstudio server
  - Any other web server
- These are published at [IP]:[PORT], so e.g: 127.0.0.1:8000
- Forward the port from the container to port on the host: `docker run -p 80:8000`
- Meaning: publish port 8000 in the container at port 80 on the host

Exercises

- Re-attaching to an exited container
- Creating an image with `docker commit`
- Non-interactive run
- Removing a container
- Pushing to docker hub
- Mounting a directory
- Managing permissions