Containers and their advantages

INTRODUCTION TO DOCKER

Tim Sangster

Software Engineer @ DataCamp

Prerequisites

INTERACTIVE COURSE

Introduction to Shell

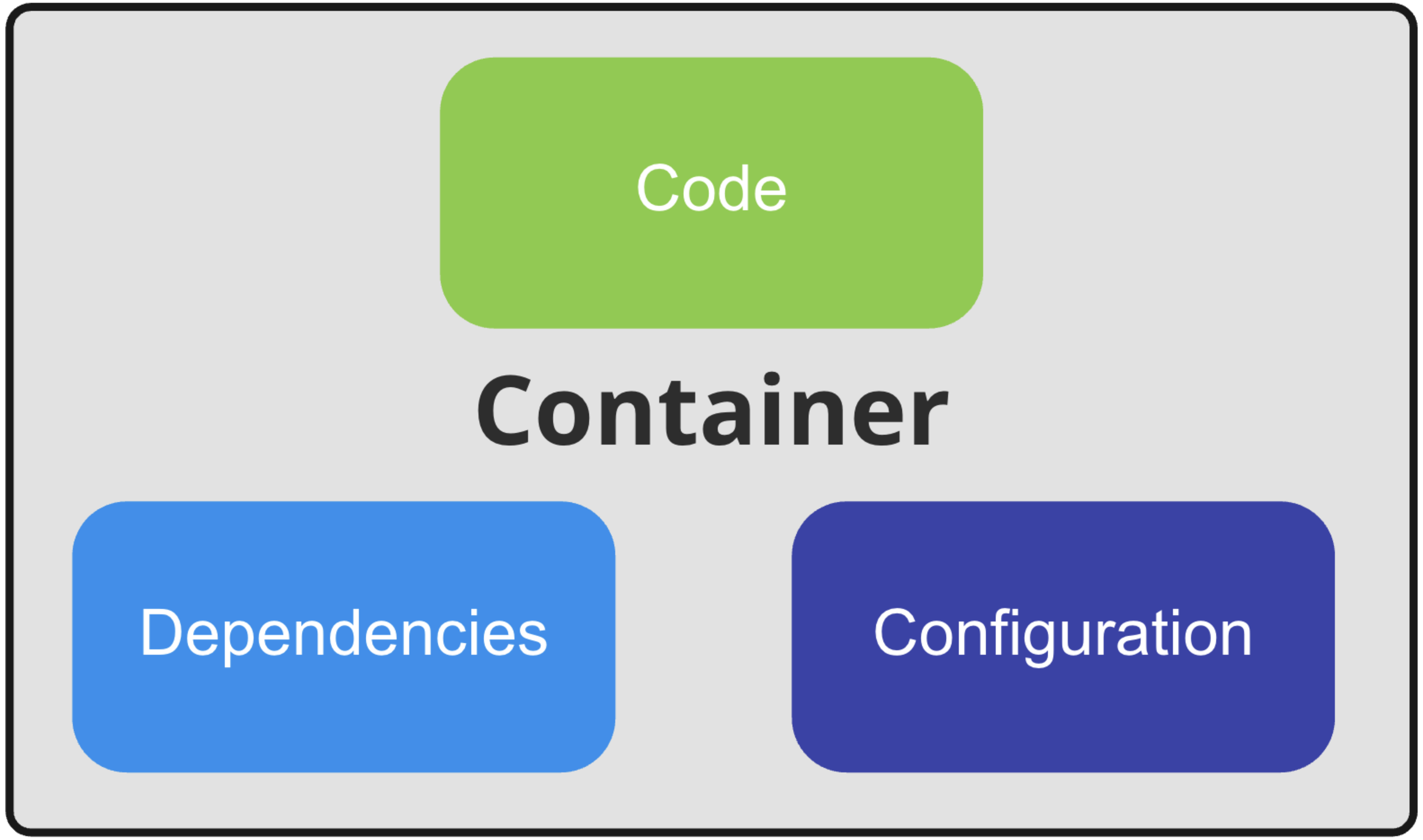
Please take DataCamp's **Introduction to Shell** before starting this course.

We will use:

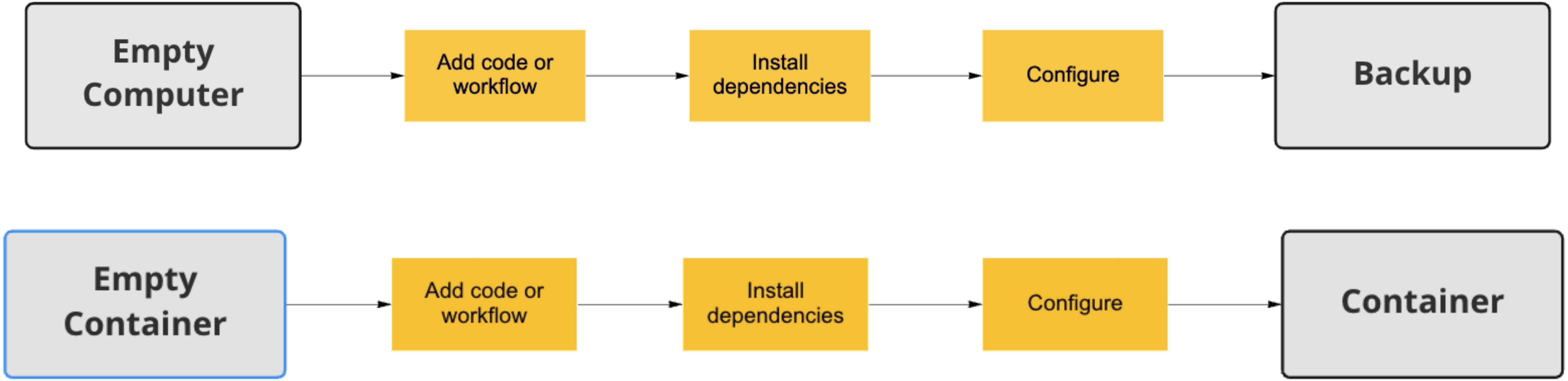
- `nano` to edit files.
- `ls` , `cd` , and `mkdir` to find our way in and manage the file system.

Containers

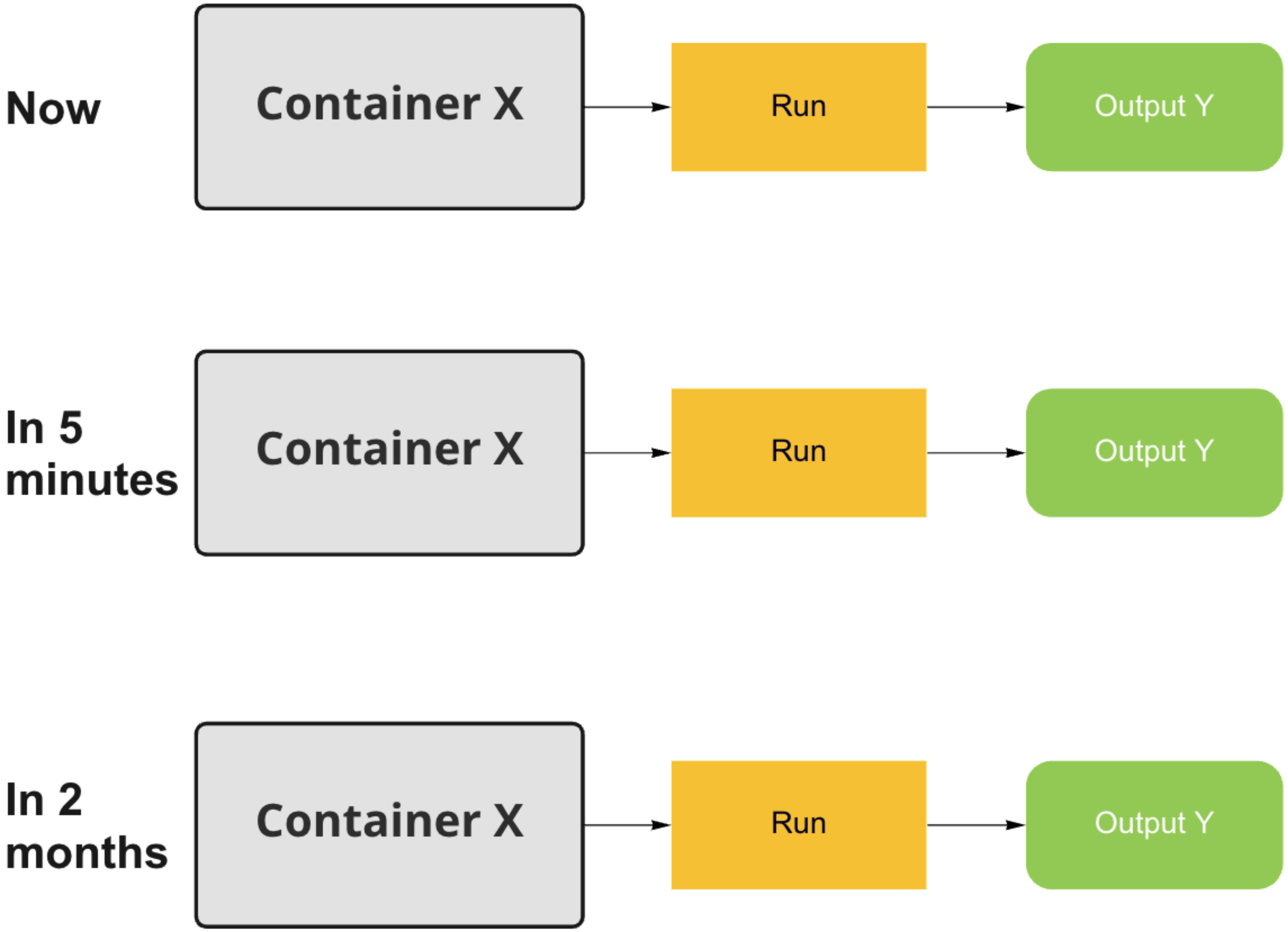
A portable computing environment



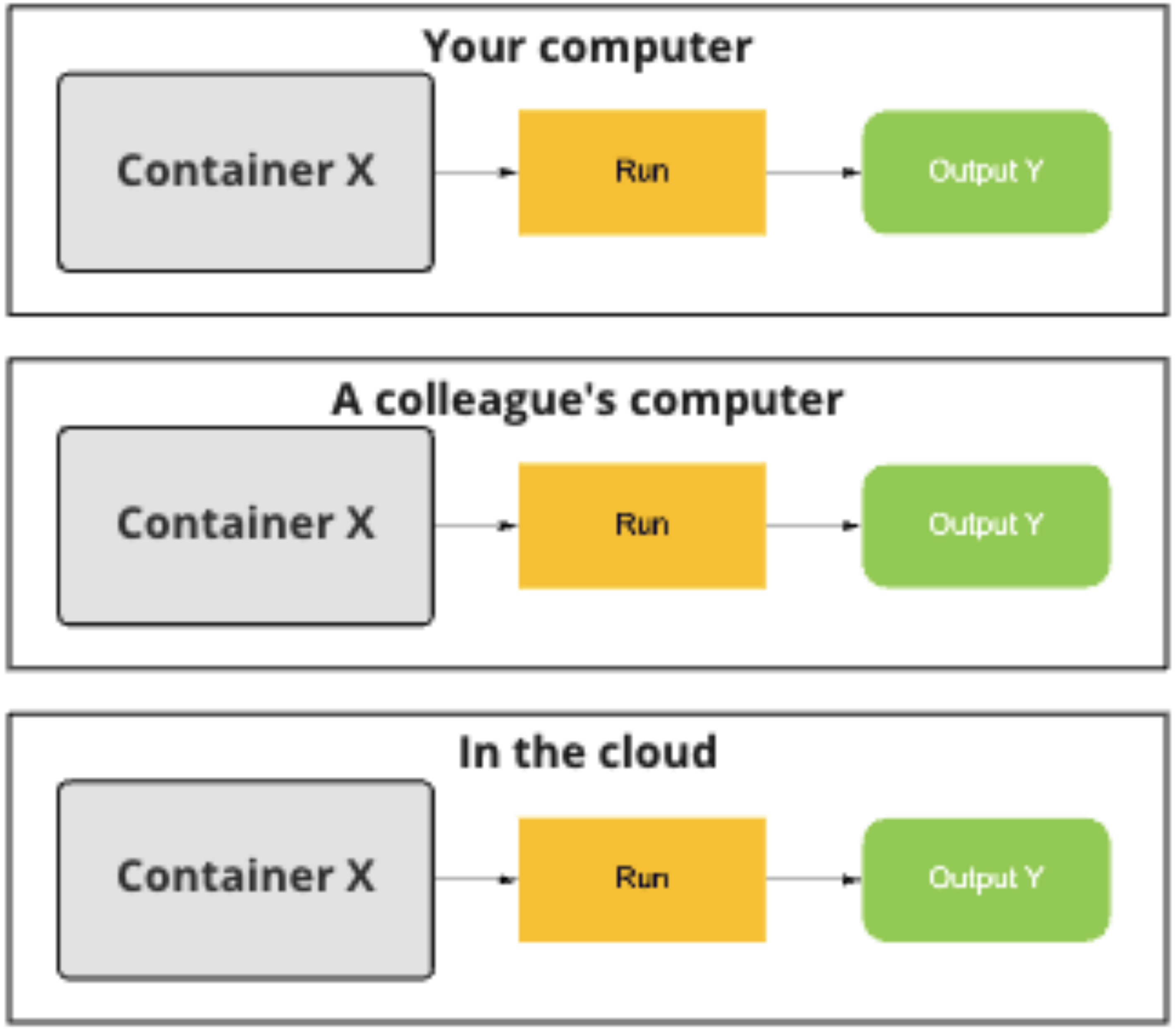
Making it less abstract



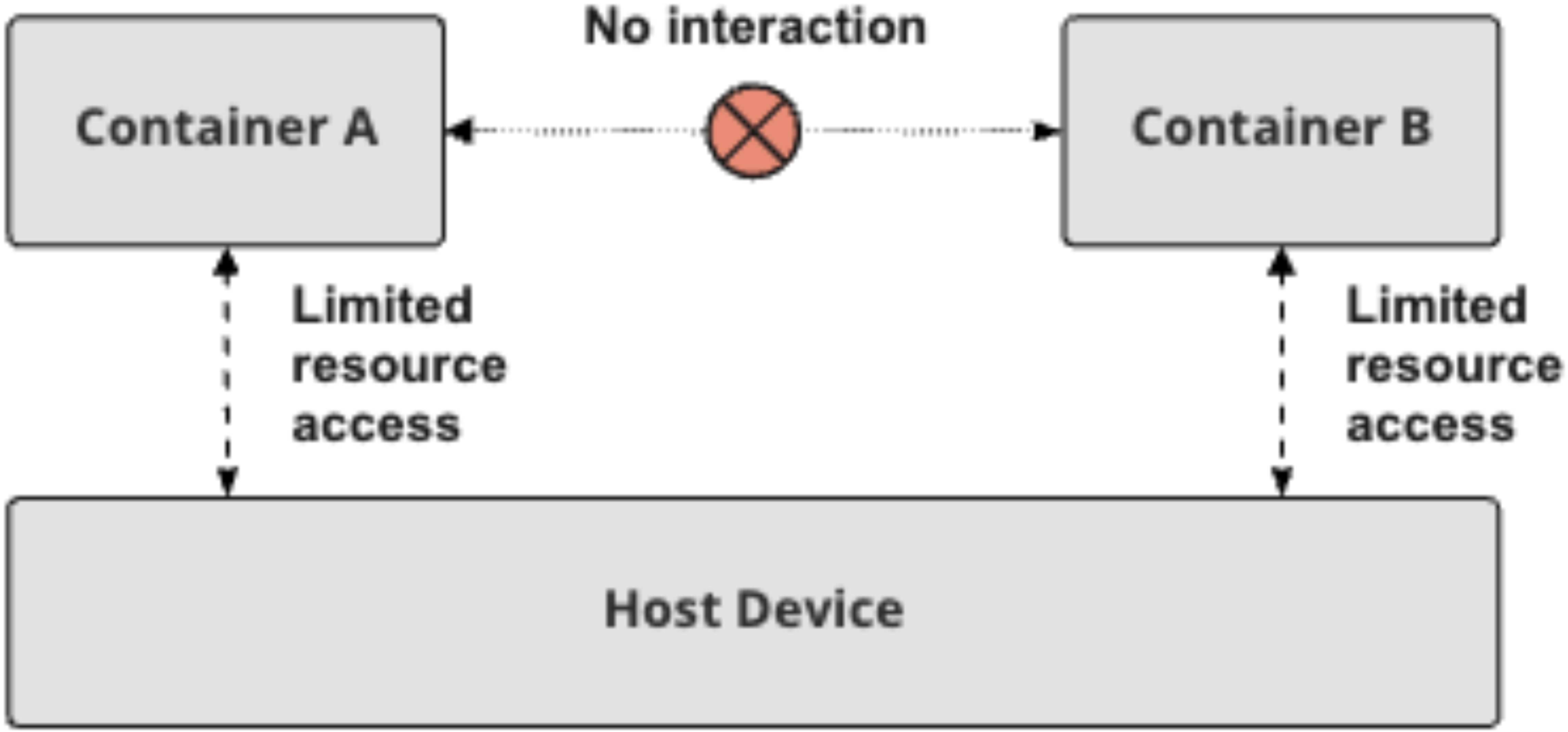
Containers run identically every time



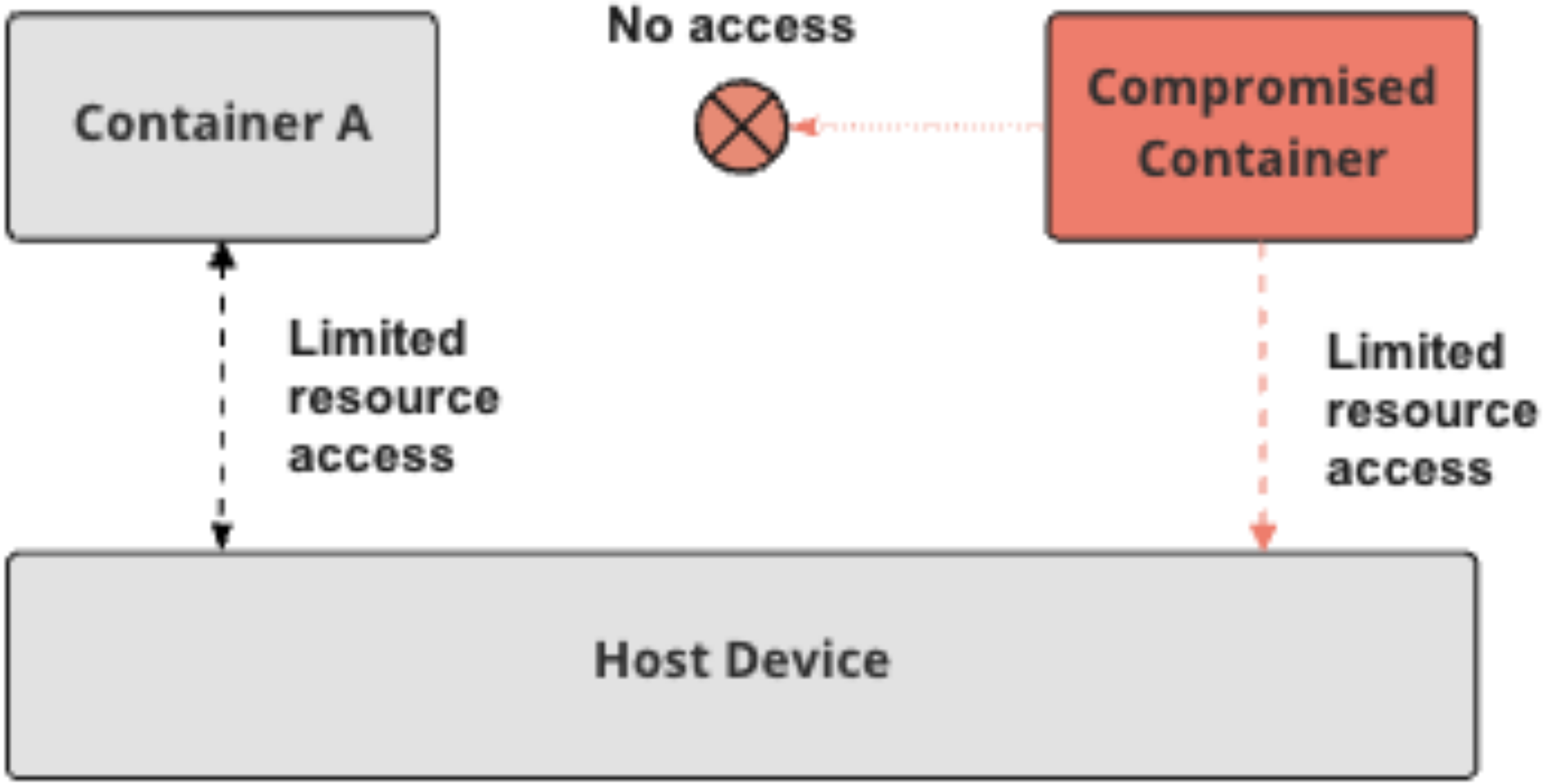
Containers run identically everywhere



Isolation


No interaction
Container A
Container B
Limited resource access
Limited resource access
Host Device

Containers provide security



Containers are lightweight

- Security
- Portability
- Reproducibility
- **Lightweight**
  - In comparison to running an application:
    - Outside of a container
    - Using a virtual machine

Containers and data science

- Automatically reproducible
- Dependencies are automatically included
- Datasets can be included
- Code will work on your colleagues machine
- Easier sharing than alternatives

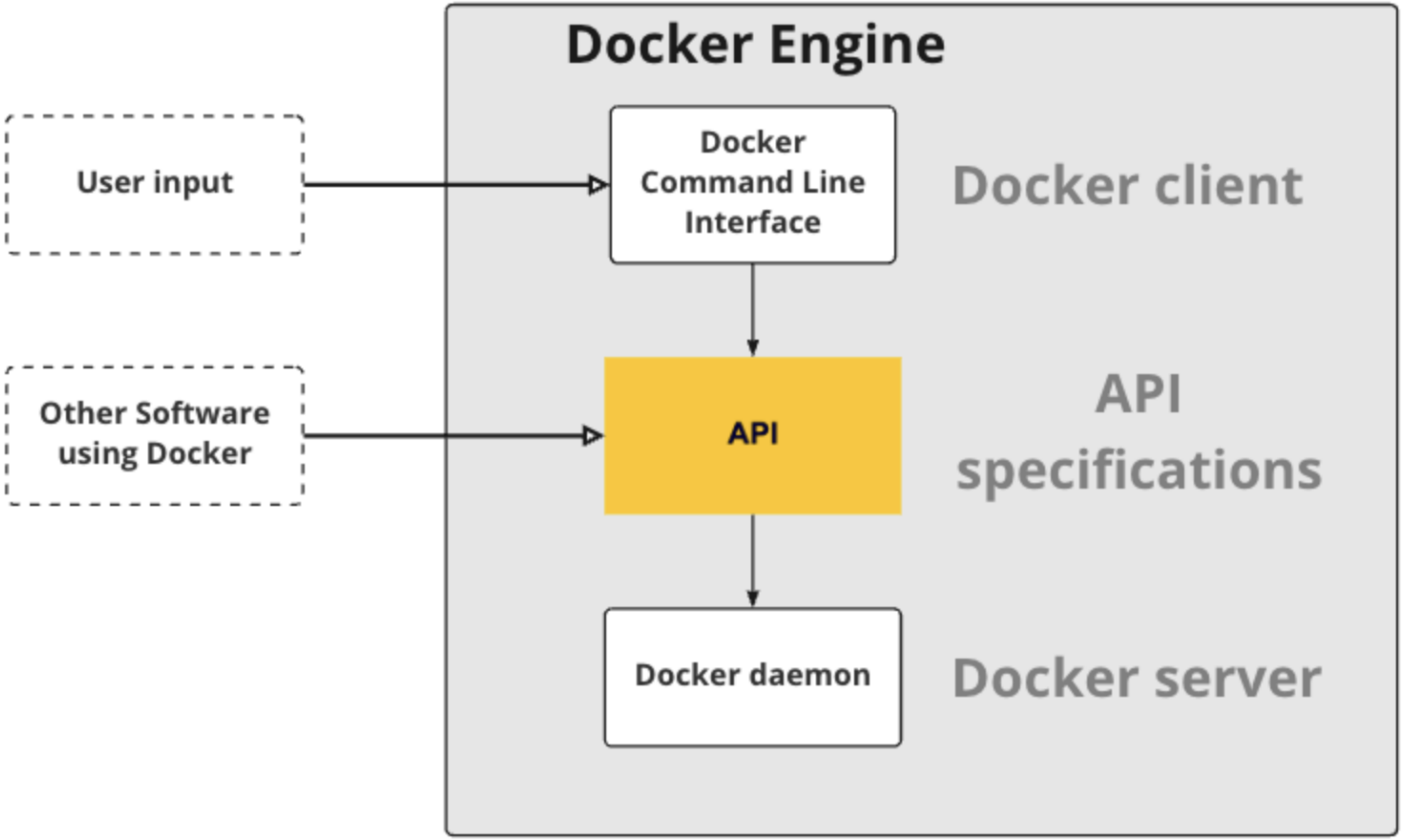
The Docker Engine

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

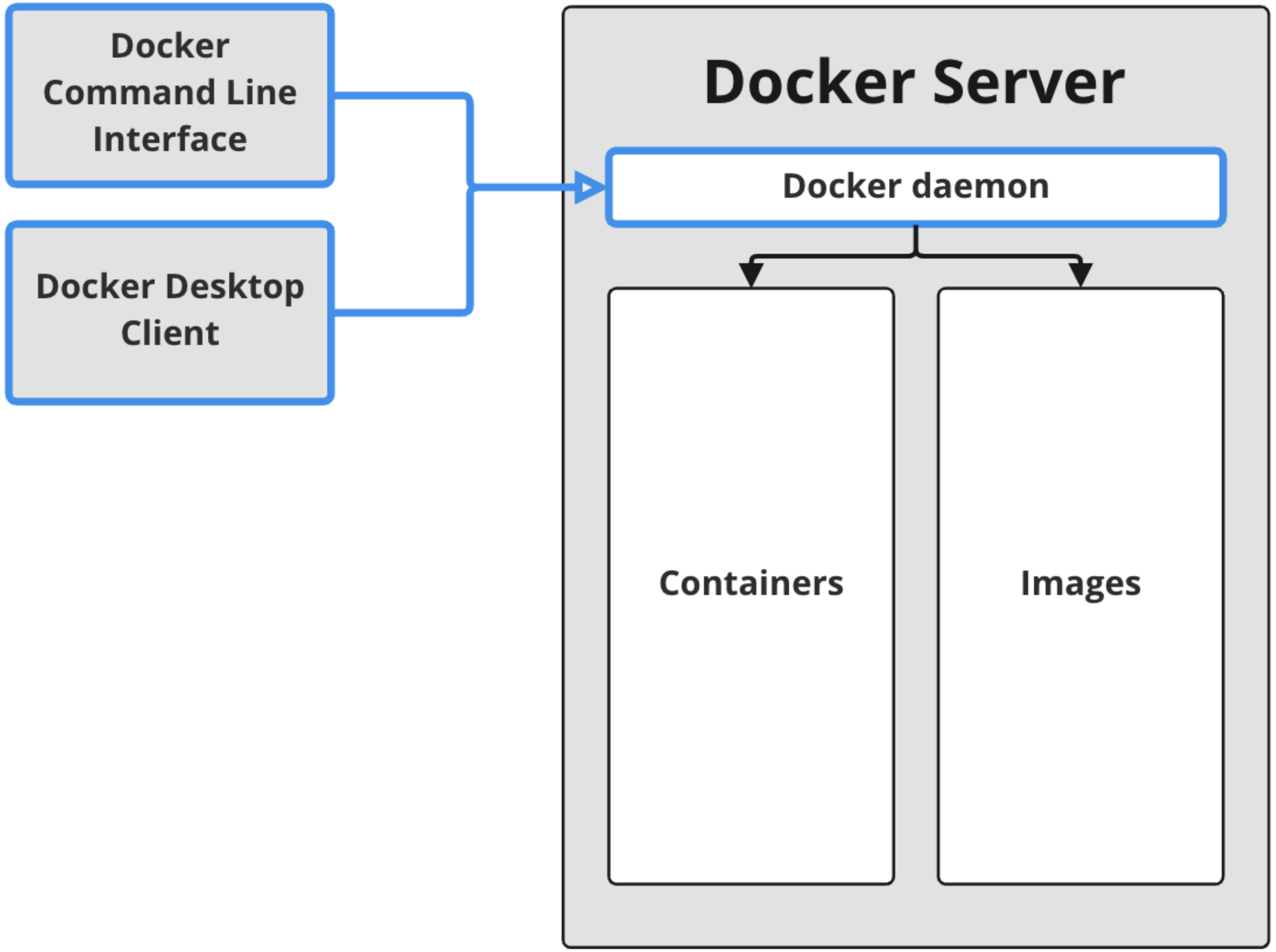
Docker ecosystem

Docker Engine



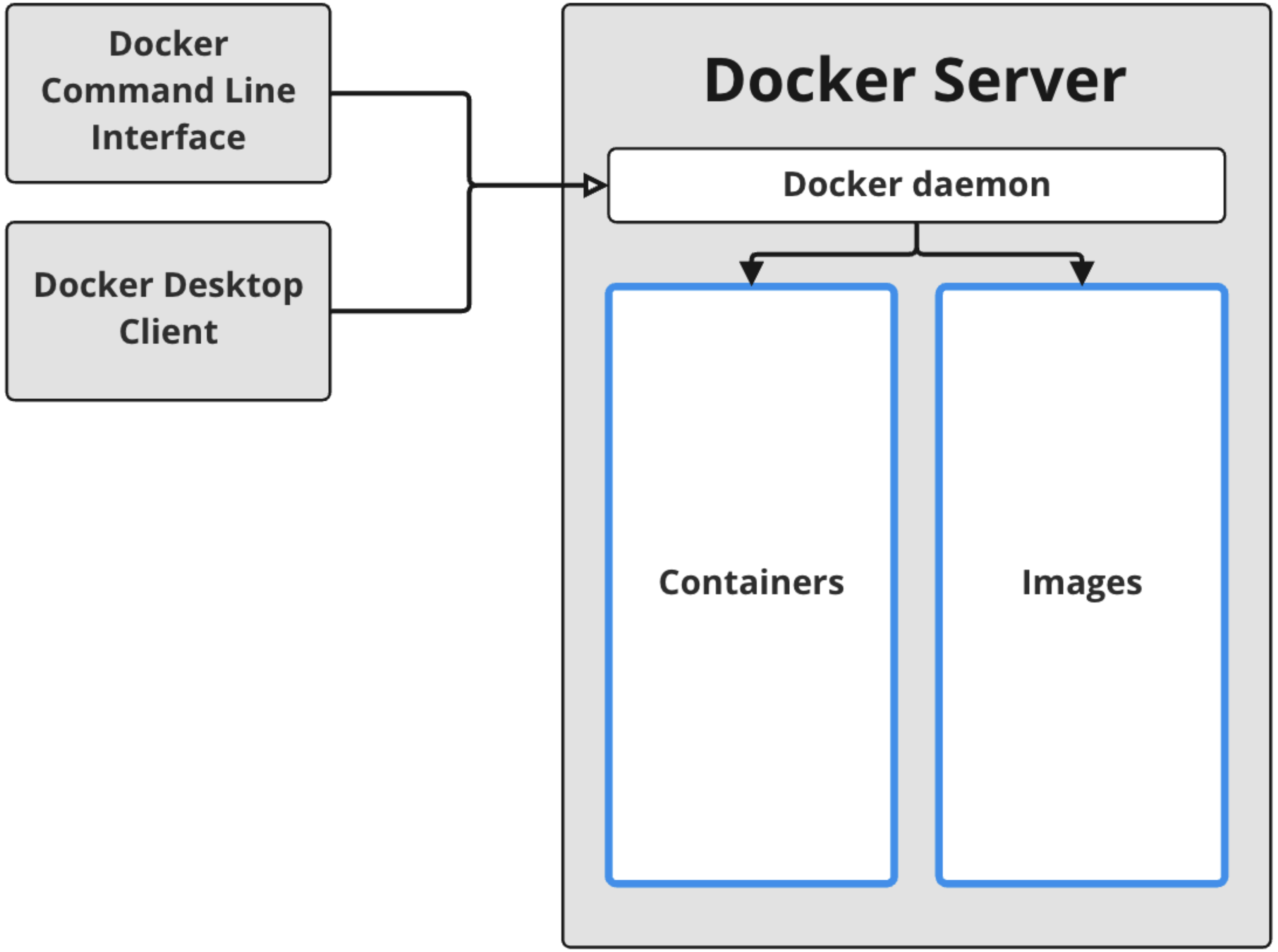
[1] https://docs.docker.com/engine/

The Docker daemon



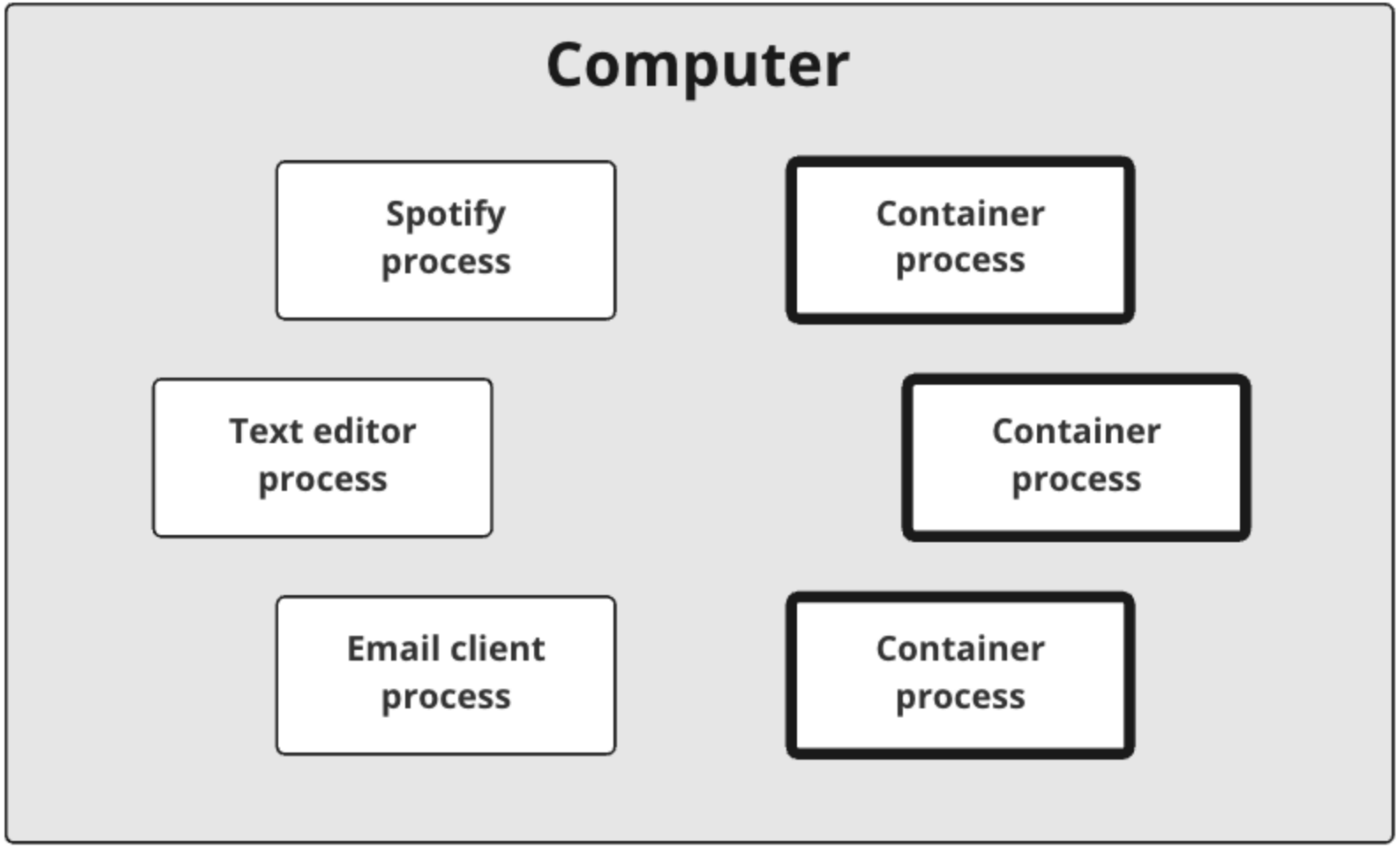
[1] https://docs.docker.com/engine/ [2] https://docs.docker.com/get-started/overview/#docker-architecture

Images and Containers

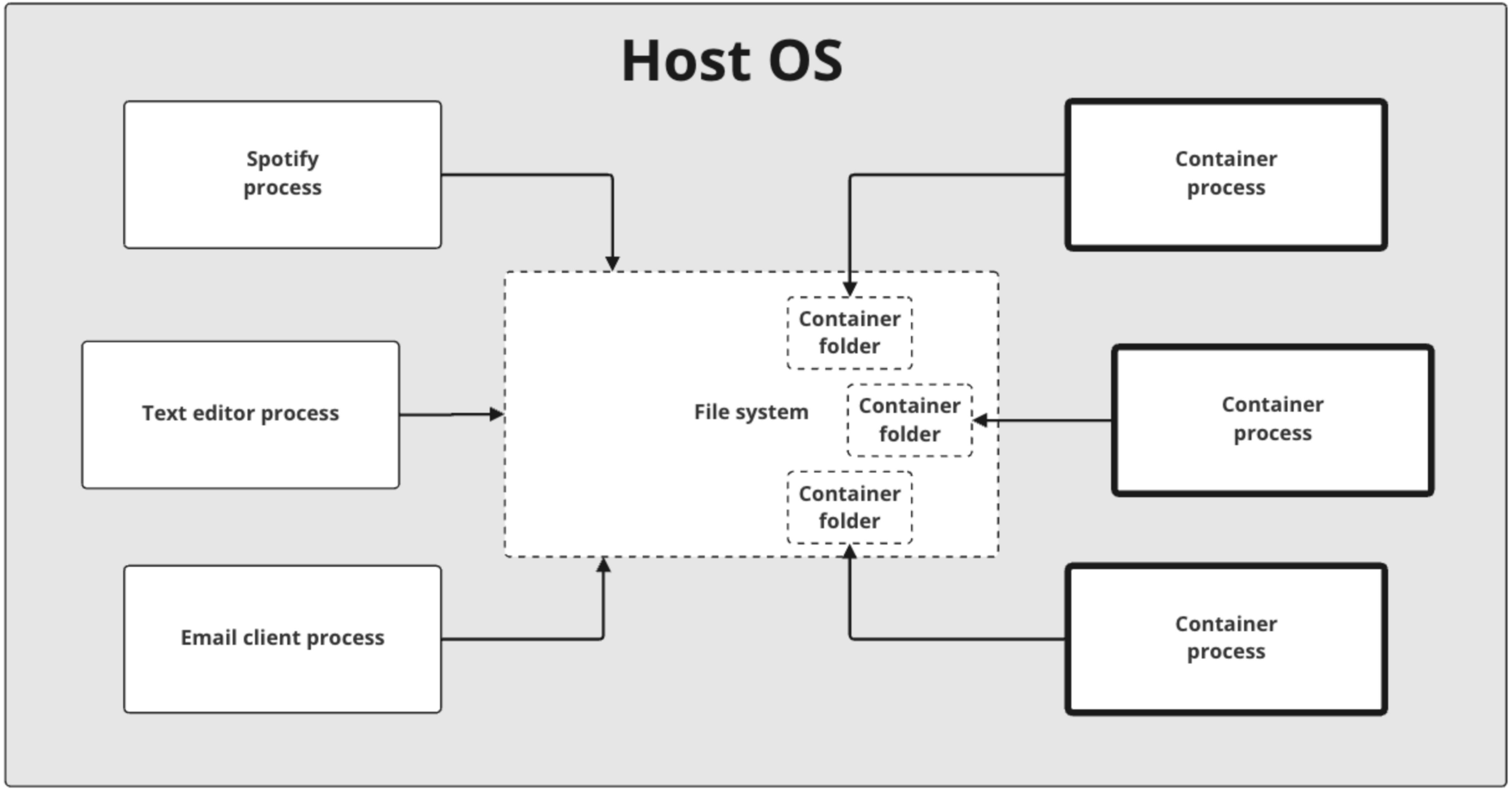


[1] https://docs.docker.com/engine/ [2] https://docs.docker.com/get-started/overview/#docker-architecture

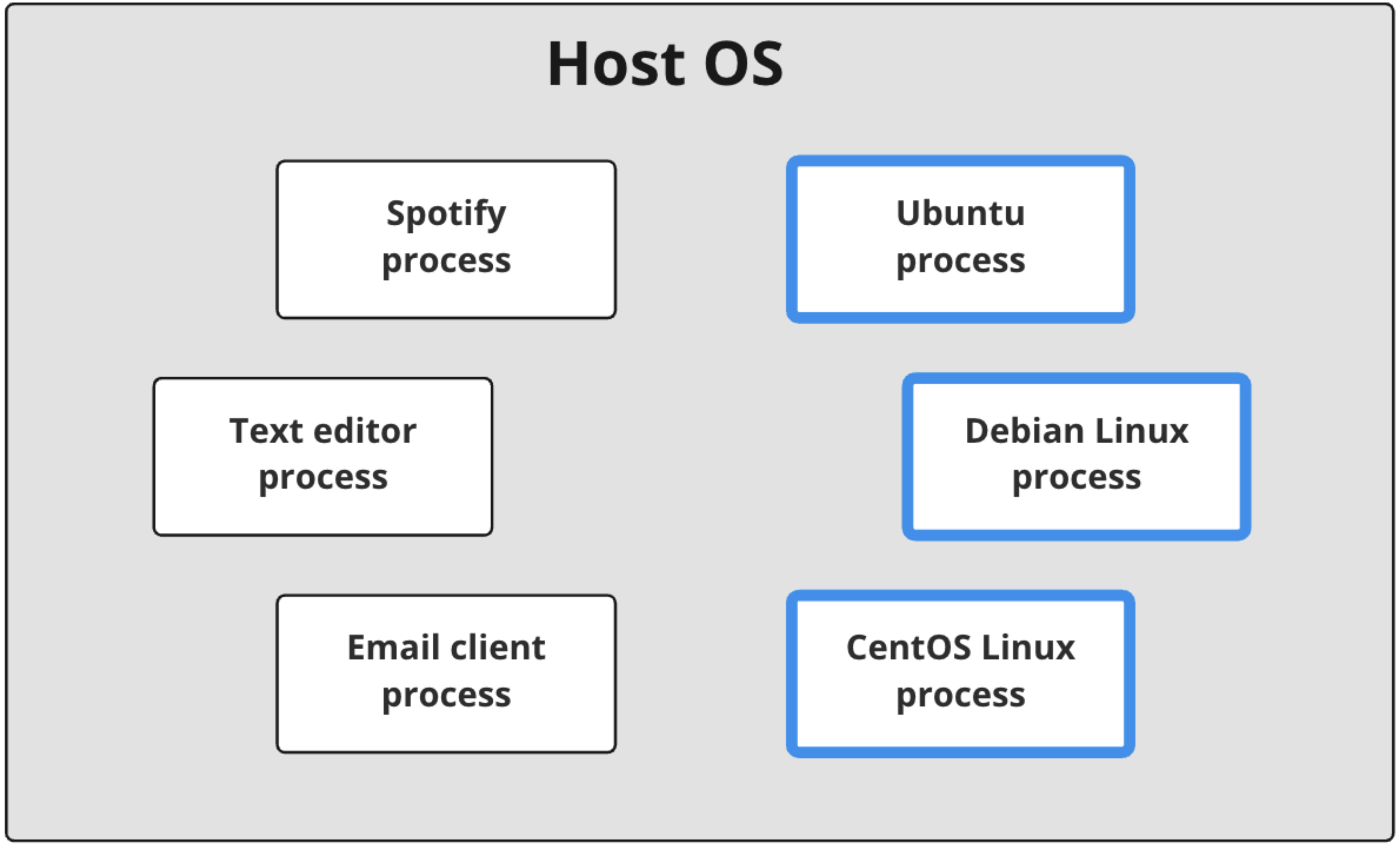
Containers are processes



Containers are processes



Containers are isolated processes

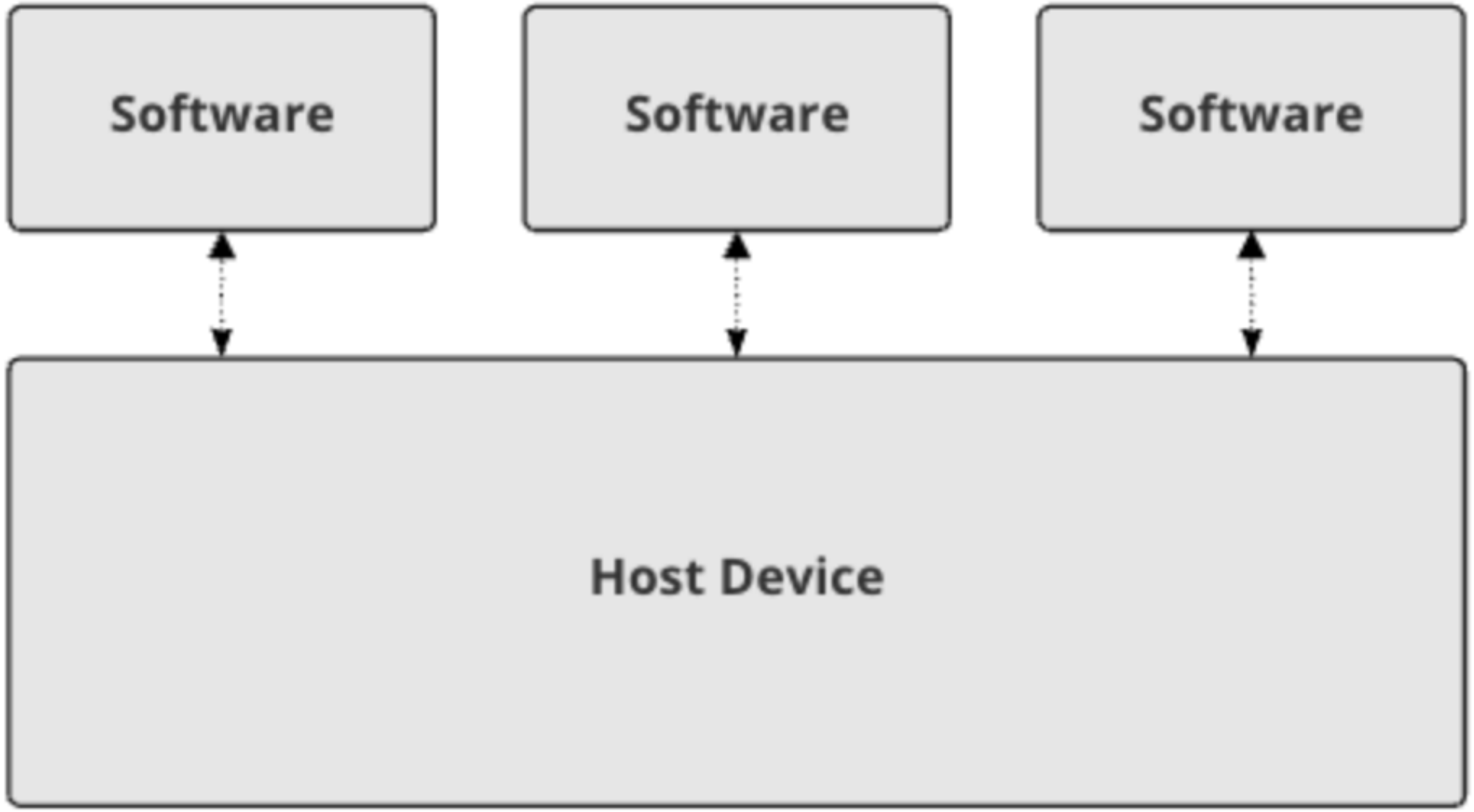


Containers vs. Virtual Machines

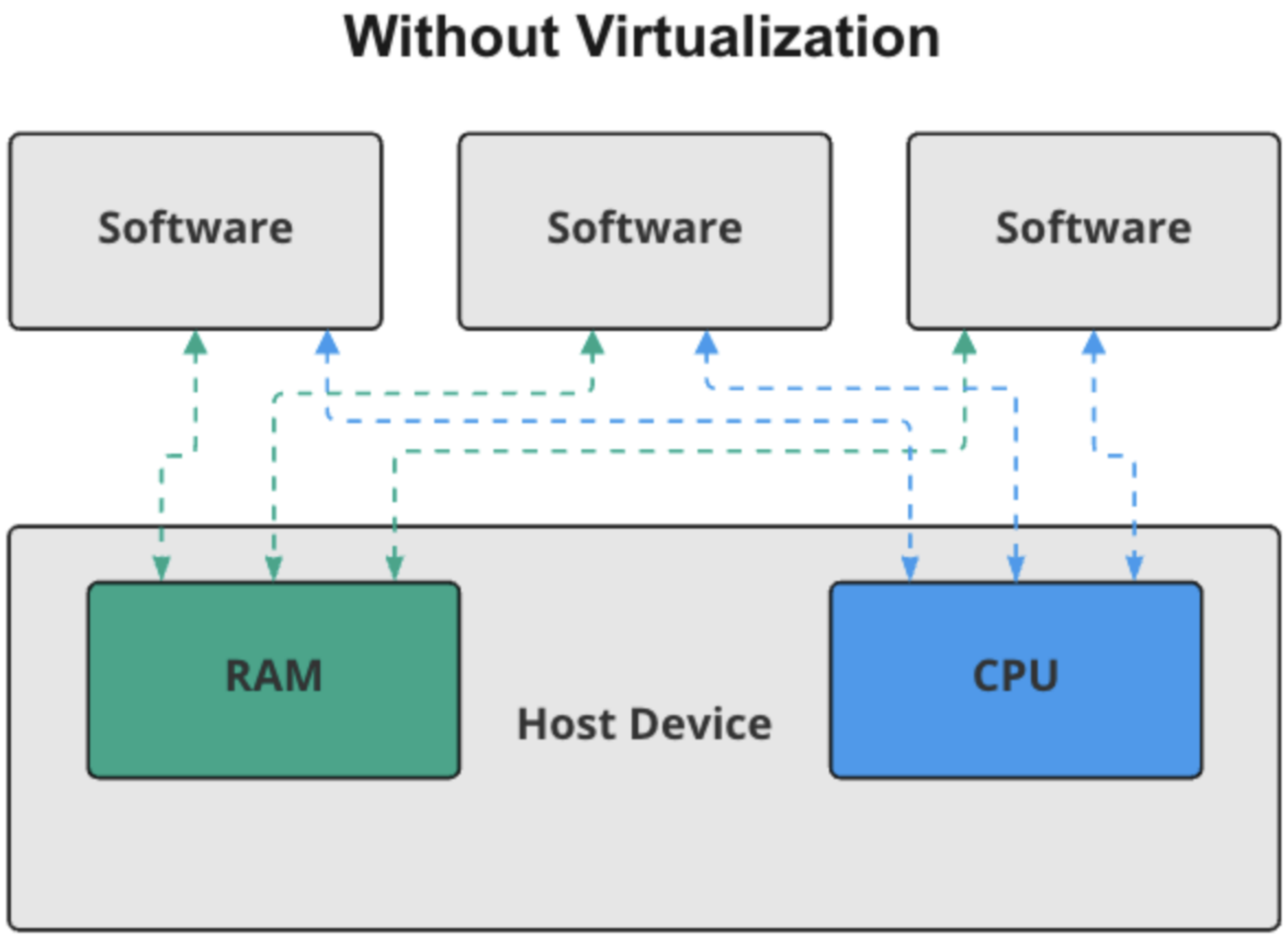
INTRODUCTION TO DOCKER

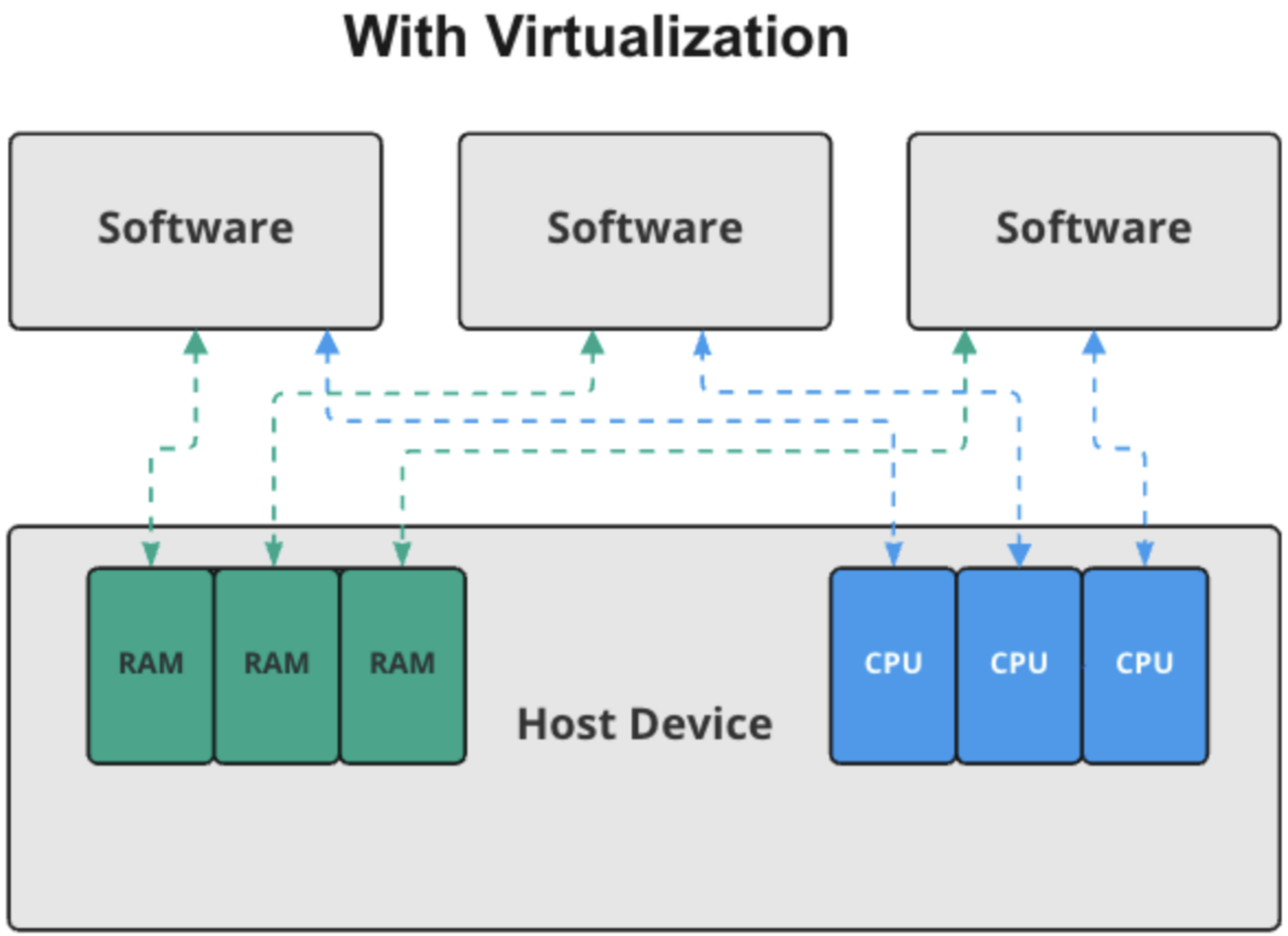
Tim Sangster
Software Engineer @ DataCamp

Containers and Virtual Machines

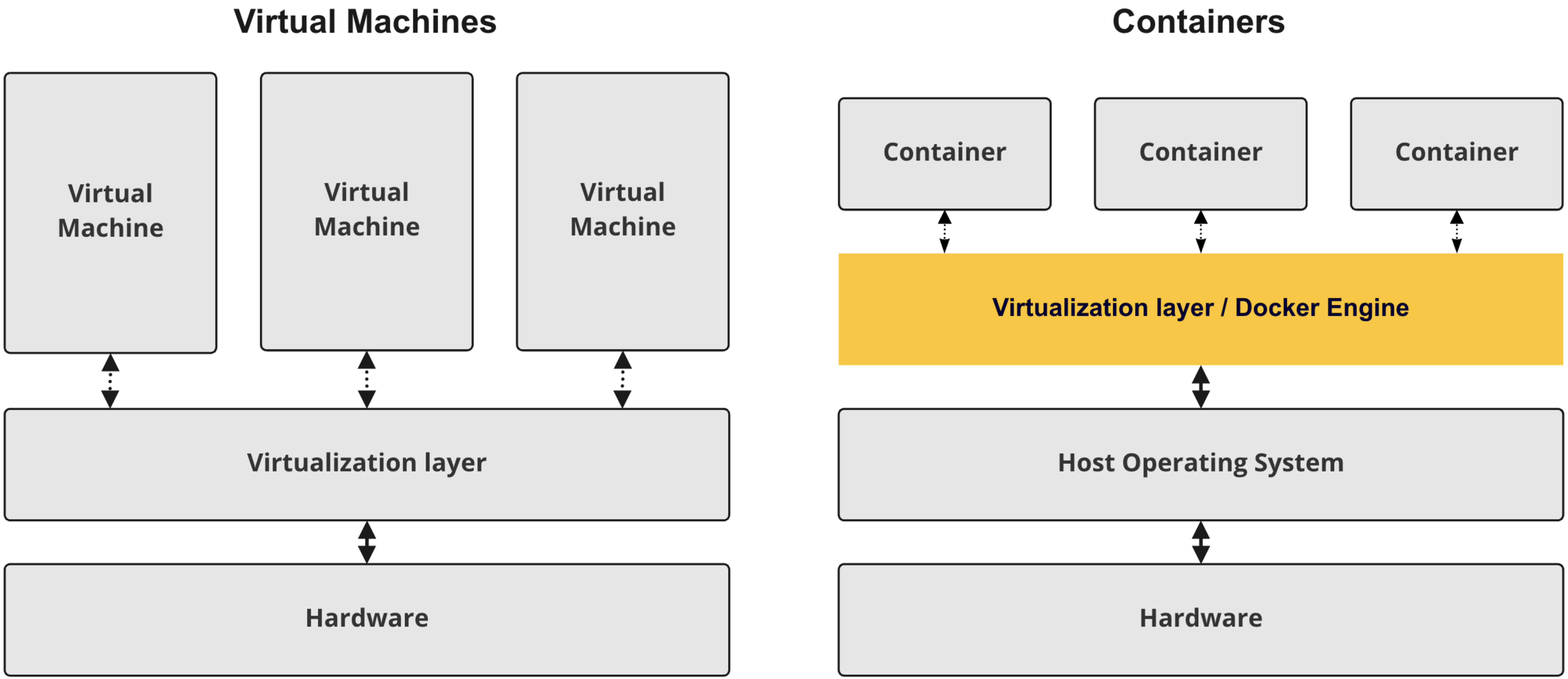


Resource Virtualization

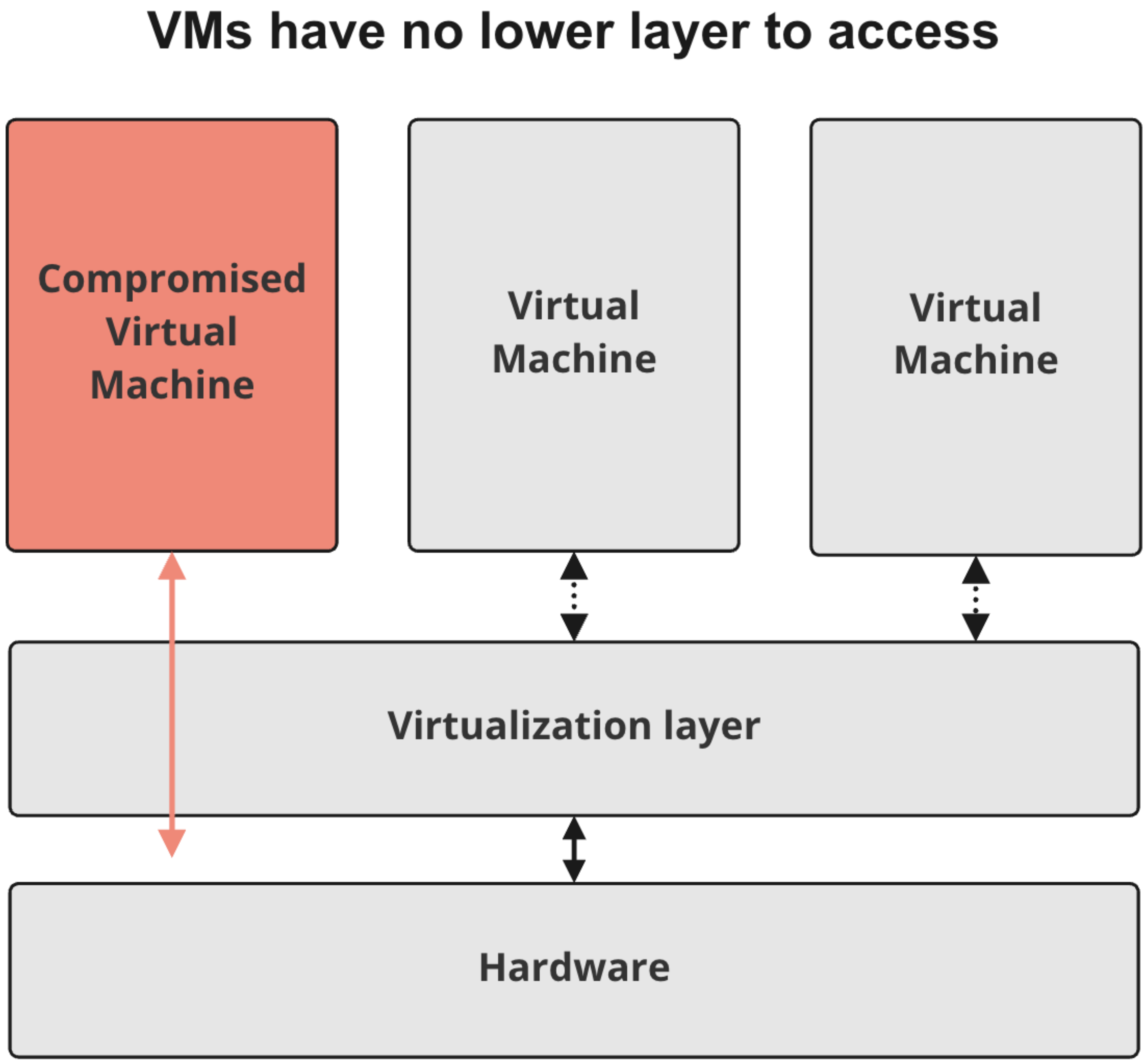


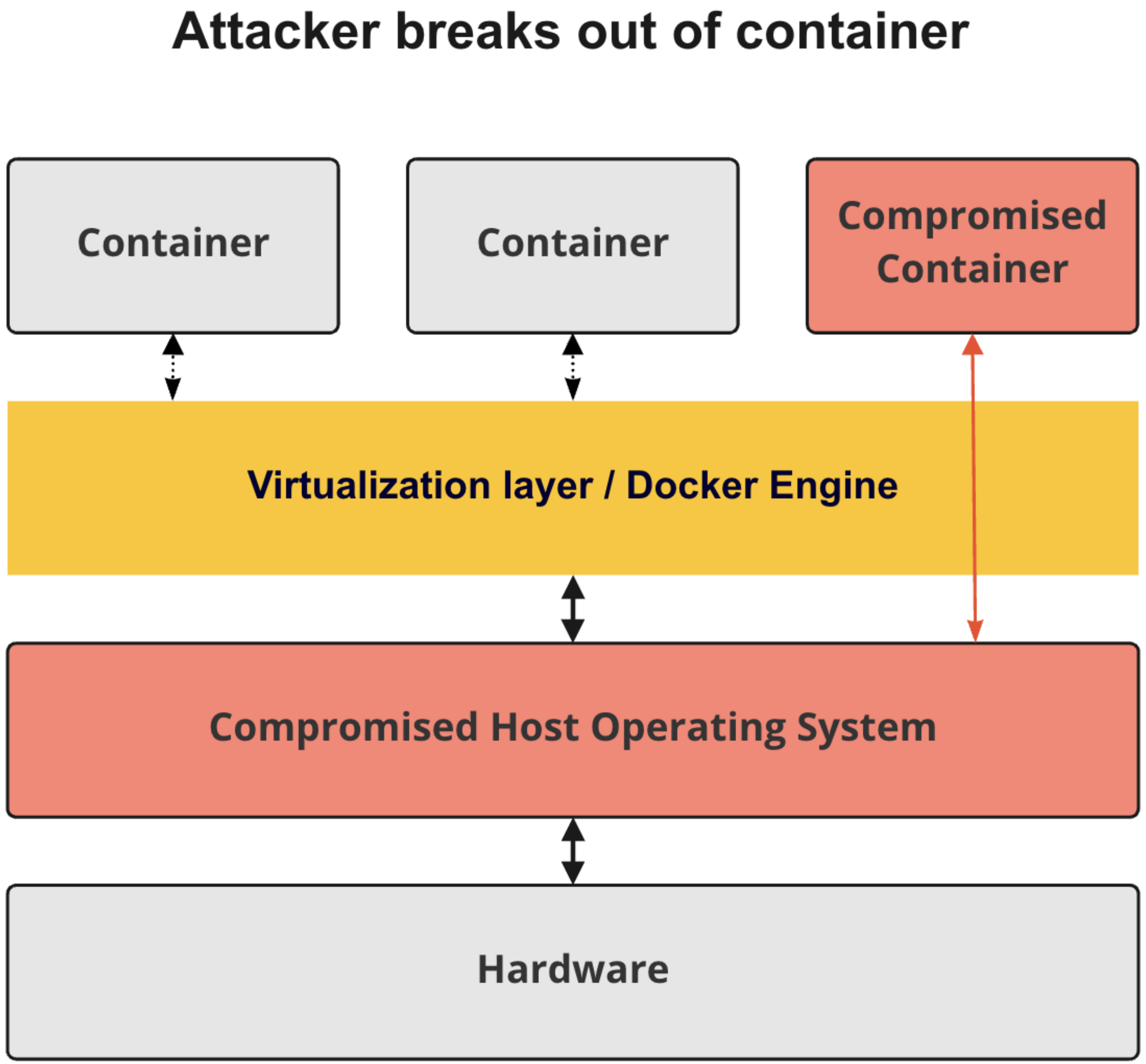


Containers vs Virtual Machines

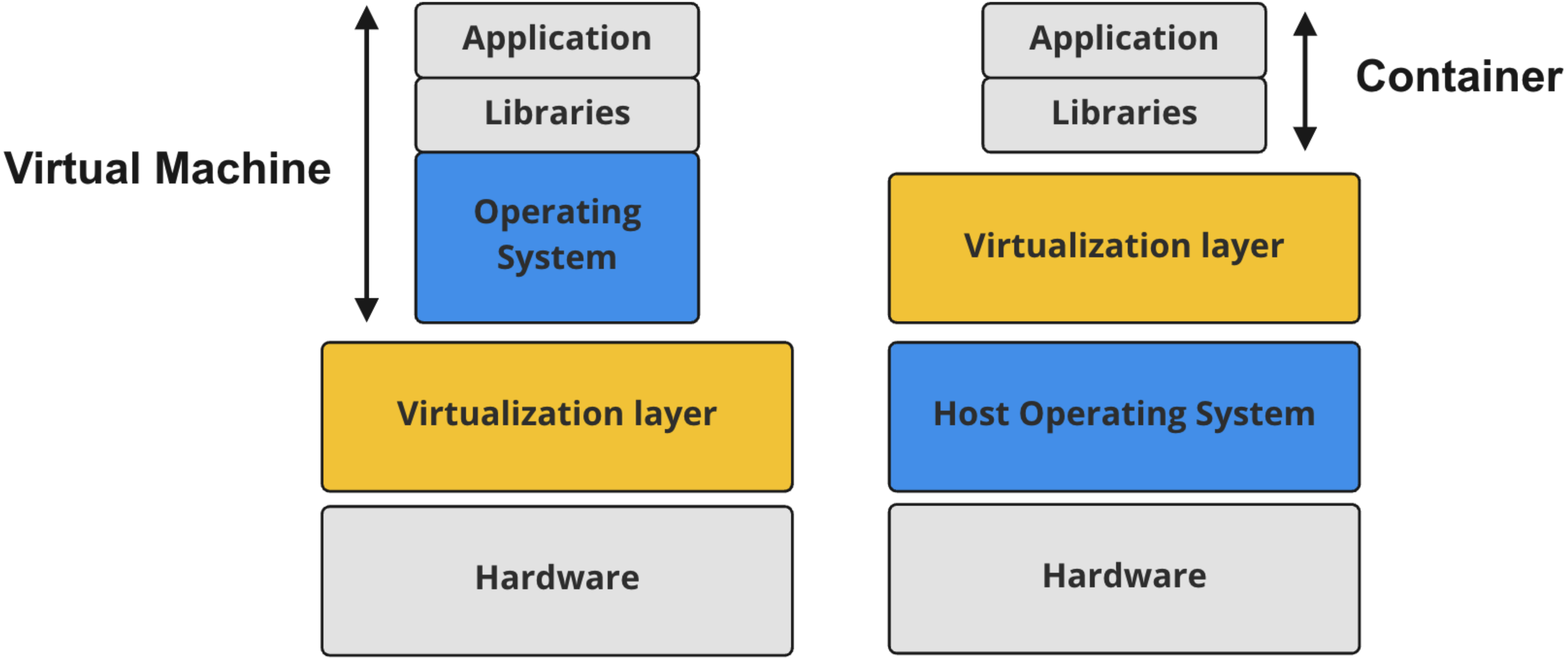


Security of Virtualization


VMs have no lower layer to access
Compromised Virtual Machine
Virtual Machine
Virtual Machine
Virtualization layer
Hardware


Attacker breaks out of container
Container
Container
Compromised Container
Virtualization layer / Docker Engine
Compromised Host Operating System
Hardware

Containers are lightweight



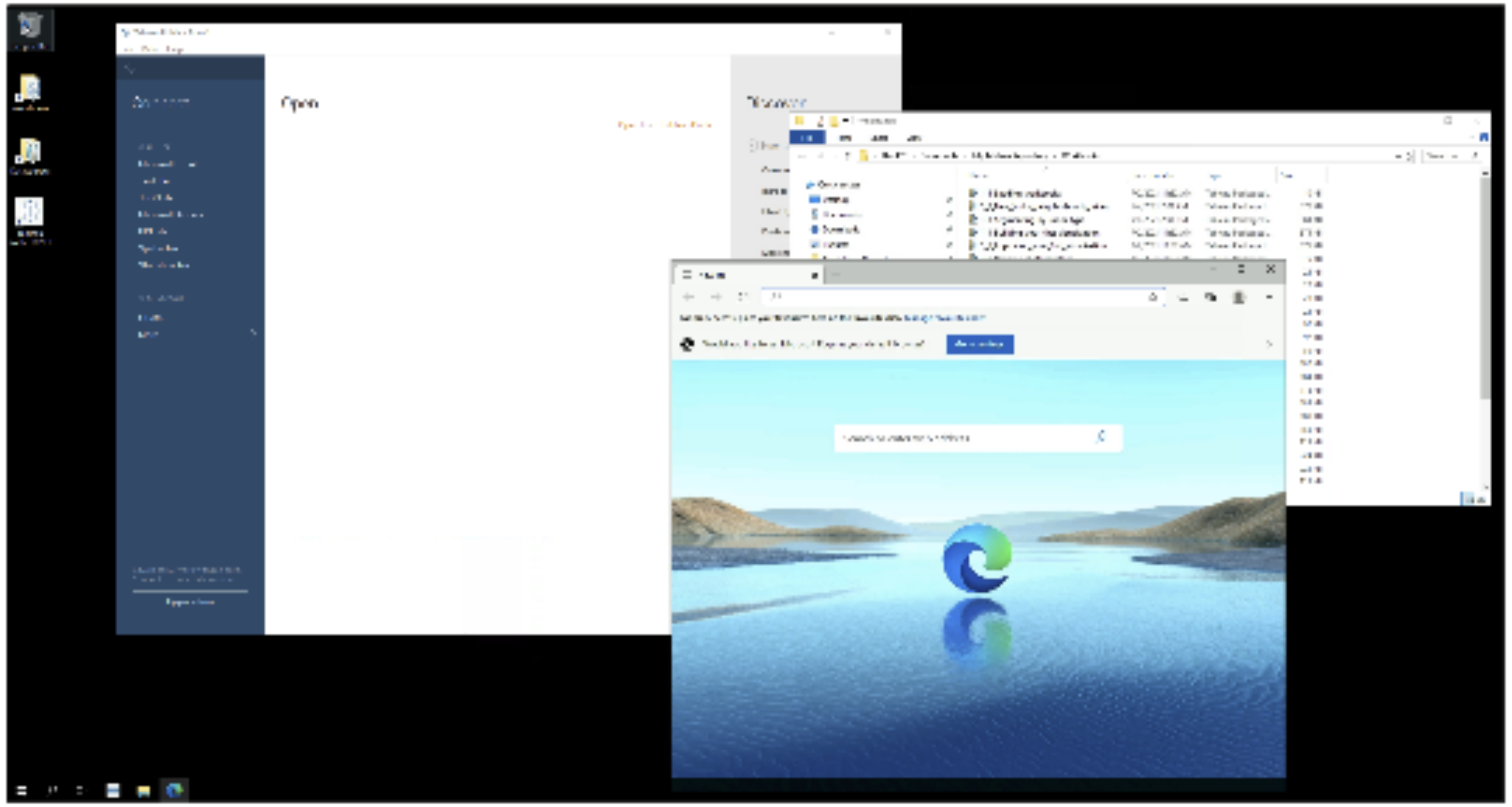
Advantages of containers

Because of their smaller size containers

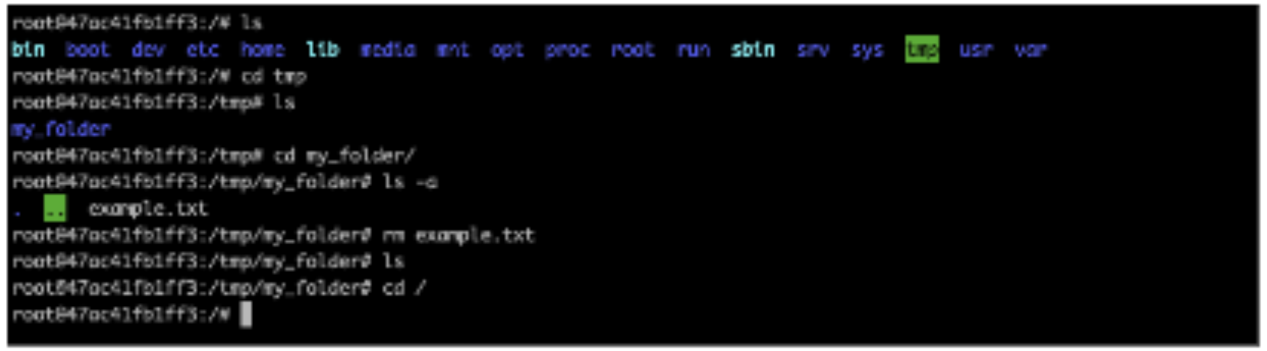
- Are faster to
  - Start
  - Stop
  - Distribute
  - To change or update
- Have a large ecosystem of pre-made containers

Advantages of Virtual Machines

Graphical User Interface (GUI)



Command Line Interface (CLI)



```
root@47ac41fb1ff3:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@47ac41fb1ff3:/# cd tmp
root@47ac41fb1ff3:/tmp# ls
my_folder
root@47ac41fb1ff3:/tmp# cd my_folder/
root@47ac41fb1ff3:/tmp/my_folder# ls -a
.  ..  example.txt
root@47ac41fb1ff3:/tmp/my_folder# rm example.txt
root@47ac41fb1ff3:/tmp/my_folder# ls
root@47ac41fb1ff3:/tmp/my_folder# cd /
root@47ac41fb1ff3:/# 
```

Running Docker containers

INTRODUCTION TO DOCKER

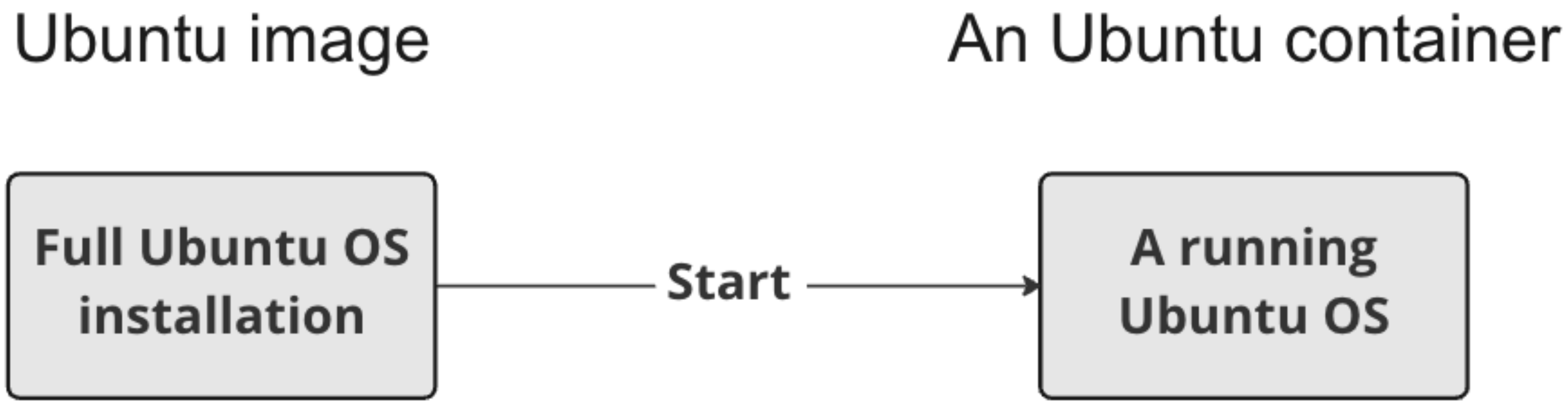
Tim Sangster
Software Engineer @ DataCamp

Prerequisite

Command	Usage
nano <file-name>	Opens <file-name> in the nano text editor
touch <file-name>	Creates an empty file with the specified name
echo "<text>"	Prints <text> to the console
<command> **>>** <file>	Pushes the output of <command> to the end of <file>
<command> -y	Automatically respond yes to all prompts from <command>

The Docker CLI

- Docker command line interface will send instructions to the Docker daemon.
- Every commands starts with `docker` .



Docker container output

```
docker run <image-name>
```

```
docker run hello-world
```

```
Hello from Docker!

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

Choosing Docker container output

```
docker run <image-name>
```

```
docker run ubuntu
```

```
repl@host:/# docker run ubuntu


repl@host:/#
```

An interactive Docker container

Adding `-it` to `docker run` will give us an interactive shell in the started container.

```
docker run -it <image-name>
```

```
docker run -it ubuntu
```

```
docker run -it ubuntu
repl@container:/#
```

```
repl@container:/# exit
exit
repl@host:/#
```

Running a container detached

Adding `-d` to `docker run` will run the container in the background, giving us back control of the shell.

```
docker run -d <image-name>
docker run -d postgres
```

```
repl@host:/# docker run -d postgres
4957362b5fb7019b56470a99f52218e698b85775af31da01958bab198a32b072
repl@host:/#
```

Listing and stopping running containers

```
docker ps
```

```
repl@host:/# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
4957362b5fb7   postgres   "docker-entrypoint.s…"   About a minute ago
STATUS              PORTS      NAMES
Up About a minute   5432/tcp   awesome_curie
```

```
docker stop <container-id>
```

```
repl@host:/# docker stop cf91547fd657
cf91547fd657
```

Summary of new commands

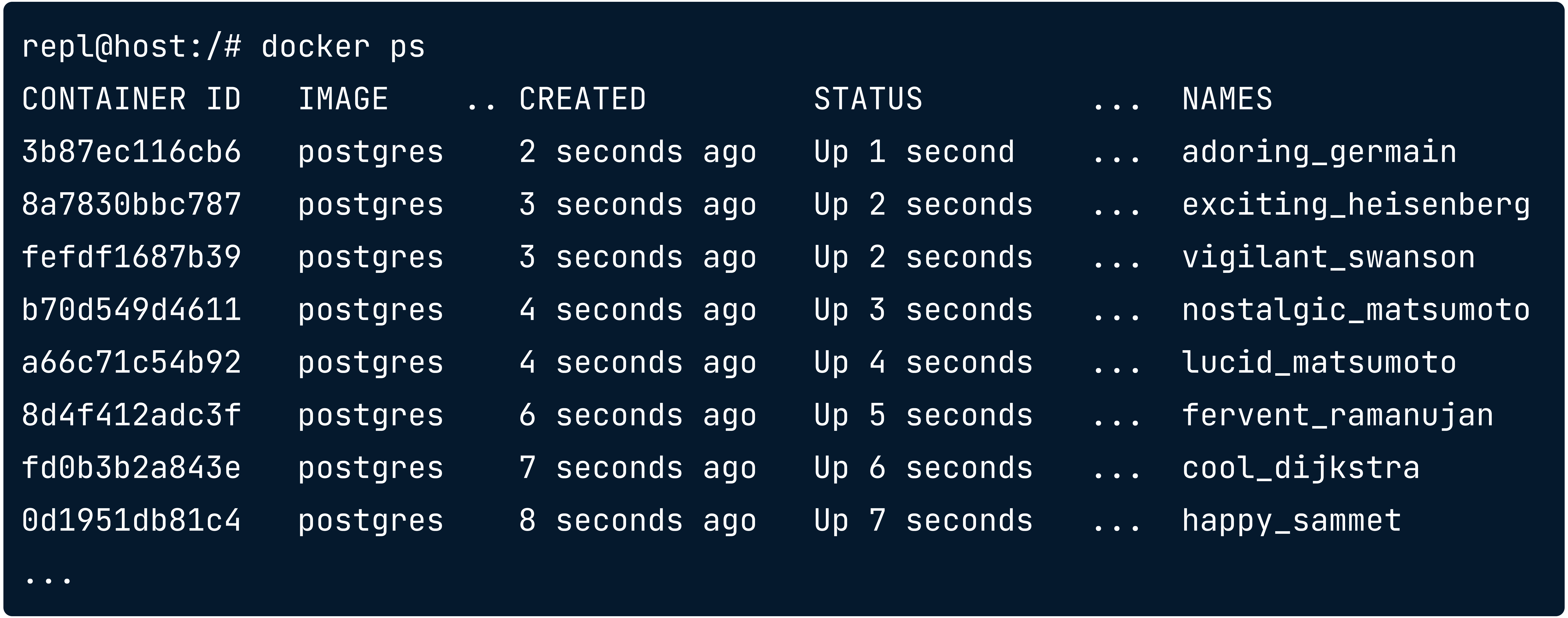
Usage	Command
Start a container	docker run <image-name>
Start an interactive container	docker run -it <image-name>
Start a detached container	docker run -d <image-name>
List running containers	docker ps
Stop a container	docker stop <container-id>

Working with Docker containers

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Listing containers



```
repl@host:/# docker ps
CONTAINER ID   IMAGE     .. CREATED          STATUS          ...  NAMES
3b87ec116cb6   postgres    2 seconds ago   Up 1 second     ...  adoring_germain
8a7830bbc787   postgres    3 seconds ago   Up 2 seconds    ...  exciting_heisenberg
fefdf1687b39   postgres    3 seconds ago   Up 2 seconds    ...  vigilant_swanson
b70d549d4611   postgres    4 seconds ago   Up 3 seconds    ...  nostalgic_matsumoto
a66c71c54b92   postgres    4 seconds ago   Up 4 seconds    ...  lucid_matsumoto
8d4f412adc3f   postgres    6 seconds ago   Up 5 seconds    ...  fervent_ramanujan
fd0b3b2a843e   postgres    7 seconds ago   Up 6 seconds    ...  cool_dijkstra
0d1951db81c4   postgres    8 seconds ago   Up 7 seconds    ...  happy_sammet
...
```

Named containers

```
docker run --name <container-name> <image-name>
```

```
repl@host:/# docker run --name db_pipeline_v1 postgres
repl@host:/# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
43aa37614330   postgres   "docker-entrypoint.s…"   About a minute ago
STATUS              PORTS      NAMES
Up About a minute   5432/tcp   db_pipeline_v1
```

```
docker stop <container-name>
```

```
repl@host:/# docker stop db_pipeline_v1
```

Filtering running containers

```
docker ps -f "name=<container-name>"
```

```
repl@host:/# docker ps -f "name=db_pipeline_v1"
CONTAINER ID   IMAGE      COMMAND                  CREATED
43aa37614330   postgres   "docker-entrypoint.s…"   About a minute ago
STATUS              PORTS      NAMES
Up About a minute   5432/tcp   db_pipeline_v1
```

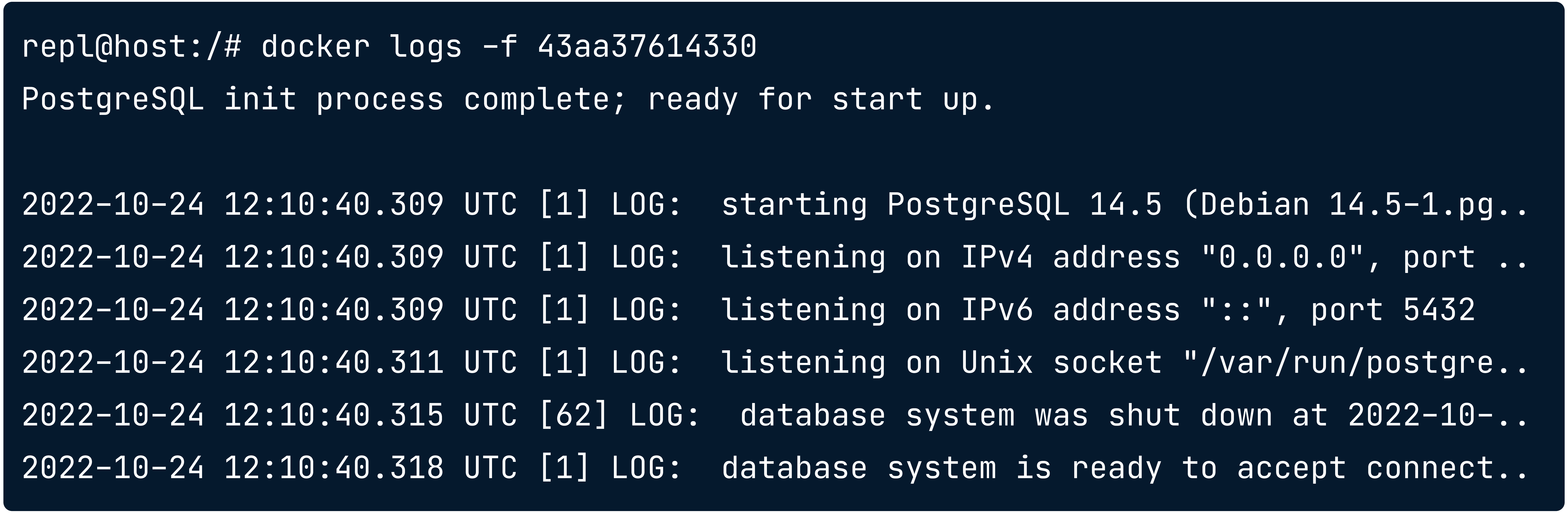
Container logs

```
docker logs <container-id>
```

```
repl@host:/# docker logs 43aa37614330
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".

PostgreSQL init process complete; ready for start up.

2022-10-24 12:10:40.318 UTC [1] LOG:  database system is ready to accept connect..
```

Live logs

```
docker logs -f <container-id>
```



```
repl@host:/# docker logs -f 43aa37614330
PostgreSQL init process complete; ready for start up.

2022-10-24 12:10:40.309 UTC [1] LOG:  starting PostgreSQL 14.5 (Debian 14.5-1.pg..
2022-10-24 12:10:40.309 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port ..
2022-10-24 12:10:40.309 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2022-10-24 12:10:40.311 UTC [1] LOG:  listening on Unix socket "/var/run/postgre..
2022-10-24 12:10:40.315 UTC [62] LOG:  database system was shut down at 2022-10-..
2022-10-24 12:10:40.318 UTC [1] LOG:  database system is ready to accept connect..
```

Cleaning up

```
docker container rm <container-id>
```

```
repl@host:/# docker stop 43aa37614330
43aa37614330
repl@host:/# docker container rm 43aa37614330
43aa37614330
```

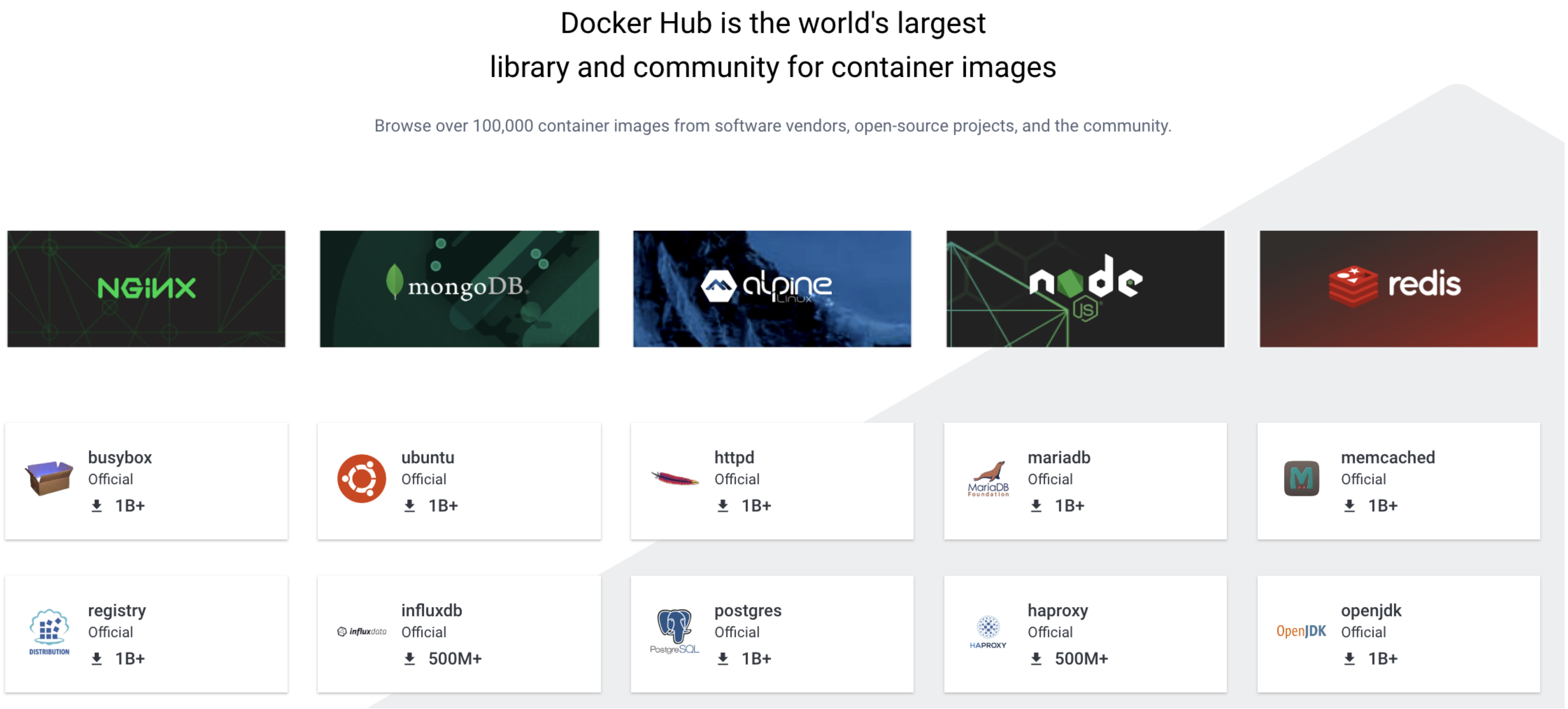
Summary of new commands

Usage	Command
Start container with a name	docker run --name <container-name> <image-name>
Filter running container on name	docker ps -f "name=<container-name>"
See existing logs for container	docker logs <container-id>
See live logs for container	docker logs -f <container-id>
Exit live log view of container	CTRL+C
Remove stopped container	docker container rm <container-id>

Managing local docker images

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp



Docker Hub is the world's largest
library and community for container images

Browse over 100,000 container images from software vendors, open-source projects, and the community.

busybox	Official	1B+
ubuntu	Official	1B+
httpd	Official	1B+
mariadb	Official	1B+
memcached	Official	1B+
registry	Official	1B+
influxdb	Official	500M+
postgres	Official	1B+
haproxy	Official	500M+
openjdk	Official	1B+

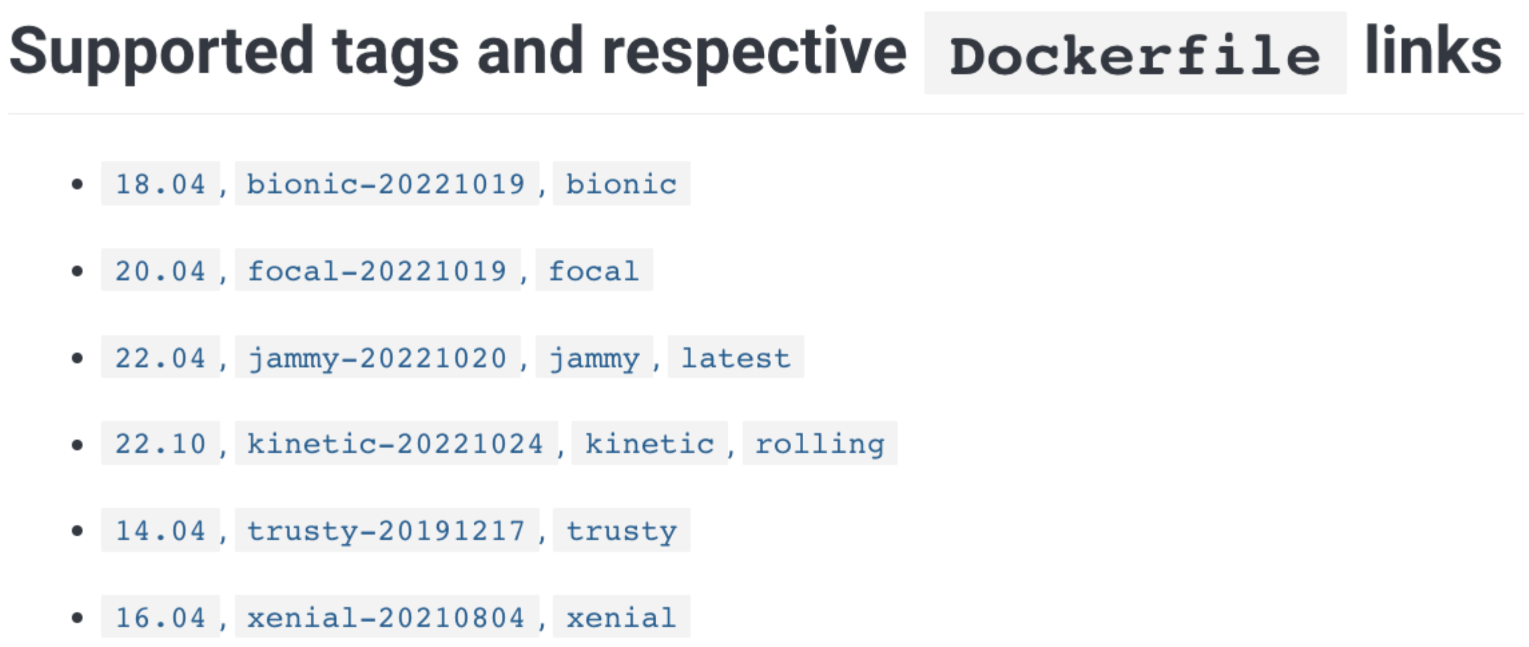
Pulling an image

```
docker pull <image-name>
```

```
docker pull postgres
docker pull ubuntu
```

```
repl@host:/# docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
7050e35b49f5: Pull complete
Digest: sha256:e18f0a777aefabe047a671ab3ec3eed05414477c951ab1a6f352a06974245fe7
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Image versions



Supported tags and respective `Dockerfile` links

- `18.04`, `bionic-20221019`, `bionic`
- `20.04`, `focal-20221019`, `focal`
- `22.04`, `jammy-20221020`, `jammy`, `latest`
- `22.10`, `kinetic-20221024`, `kinetic`, `rolling`
- `14.04`, `trusty-20191217`, `trusty`
- `16.04`, `xenial-20210804`, `xenial`

```
docker pull <image-name>:<image-version>
```

```
docker pull ubuntu:22.04
docker pull ubuntu:jammy
```

Listing images

```
docker images
```

```
repl@host:/# docker images
REPOSITORY    TAG               IMAGE ID       CREATED          SIZE
hello-world   latest            46331d942d63   7 months ago     9.14kB
ubuntu        bionic-20210723   7c0c6ae0b575   15 months ago    56.6MB
postgres      12.7              f076c2fa35f5   15 months ago    300MB
postgres      10.3              cbb7481ff9d5   4 years ago      232MB
...
```

Removing images

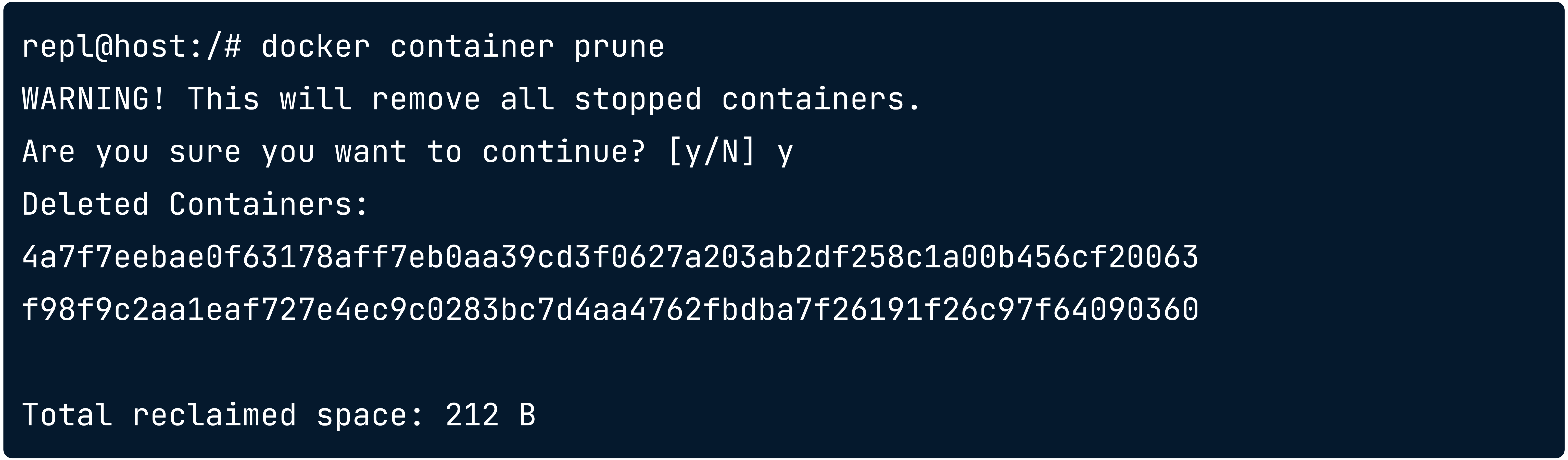
```
docker image rm <image-name>
```

```
repl@host:/# docker image rm hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:e18f0a777aefabe047a671ab3ec3eed05414477c951ab1a6f35..
Deleted: sha256:46331d942d6350436f64e614d75725f6de3bb5c63e266e236e04389820a234c4
Deleted: sha256:efb53921da3394806160641b72a2cbd34ca1a9a8345ac670a85a04ad3d0e3507
```

```
repl@host:/# docker image rm hello-world
Error response from daemon: conflict: unable to remove repository reference
"hello-world" (must force) - container 96a7b7b0c535 is using its
referenced image 46331d942d63
```

Cleaning up containers

```
docker container prune
```



```
repl@host:/# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
4a7f7eebae0f63178aff7eb0aa39cd3f0627a203ab2df258c1a00b456cf20063
f98f9c2aa1eaf727e4ec9c0283bc7d4aa4762fbdba7f26191f26c97f64090360

Total reclaimed space: 212 B
```

Cleaning up images

```
docker image prune -a
```

```
repl@host:/# docker image prune -a
WARNING! This will remove all images without at least one container associated t..
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: alpine:latest
untagged: alpine@sha256:3dcdb92d7432d56604d4545cbd324b14e647b313626d99b889d0626d..
deleted: sha256:4e38e38c8ce0b8d9041a9c4fefe786631d1416225e13b0bfe8cfa2321aec4bba
deleted: sha256:4fe15f8d0ae69e169824f25f1d4da3015a48feeeeebb265cd2e328e15c6a869f

Total reclaimed space: 16.43 MB
```

Dangling images

```
docker images
```

```
repl@host:/# docker images
REPOSITORY    TAG               IMAGE ID       CREATED         SIZE
testsql       latest            6c49f0cce145   7 months ago    3.73GB
<none>        <none>            a22b8450b88f   7 months ago    3.73GB
<none>        <none>            10dd2d03f59c   7 months ago    3.73GB
<none>        <none>            878bae40320b   7 months ago    3.73GB
<none>        <none>            4ea70583ba54   7 months ago    3.75GB
<none>        <none>            3c64576a3a7d   7 months ago    3.75GB
```

Summary of new commands

Usage	Command
Pull an image	docker pull <image-name>
Pull a specific version of an image	docker pull <image-name>:<image-version>
List all local images	docker images
Remove an image	docker image rm <image-name>
Remove all stopped containers	docker container prune
Remove all images	docker image prune -a

Distributing Docker Images

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Private Docker registries

- Unlike Docker official images there is no quality guarantee
- Name starts with the url of the private registry

```
dockerhub.myprivateregistry.com/classify_spam
```

```
docker pull dockerhub.myprivateregistry.com/classify_spam:v1
```

```
Using tag: v1
latest: Pulling from dockerhub.myprivateregistry.com
ed02c6ade914: Pull complete
Digest: sha256:b6b83d3c331794420340093eb706b6f152d9c1fa51b262d9bf34594887c2c7ac
Status: Downloaded newer image for dockerhub.myprivateregistry.com/classify_spam:v1
dockerhub.myprivateregistry.com/classify_spam:v1
```

Pushing to a registry

```
docker image push <image name>
```

Pushing to a specific registry --> name of the image needs to start with the registry url

```
docker tag classify_spam:v1 dockerhub.myprivateregistry.com/classify_spam:v1
```

```
docker image push dockerhub.myprivateregistry.com/classify_spam:v1
```

Authenticating against a registry

- Docker official images --> No authentication needed
- Private Docker repository --> Owner can choose

```
docker login dockerhub.myprivateregistry.com
```

```
user@pc ~ % docker login dockerhub.myprivateregistry.com
Username: student
Password:
Login succeeded
```

Docker images as files

Sending a Docker image to one or a few people? Send it as a file!

Save an image

```
docker save -o image.tar classify_spam:v1
```

Load an image

```
docker load -i image.tar
```

Summary of new commands

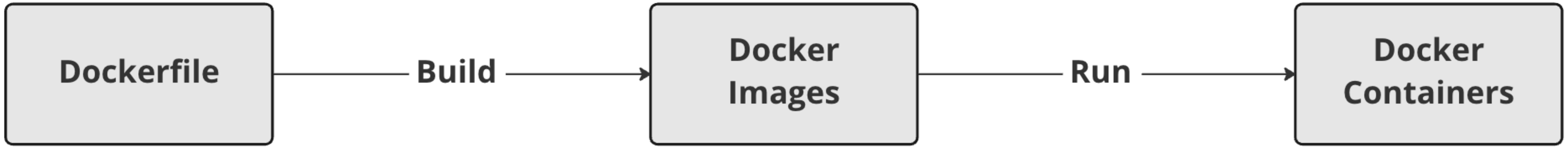
Usage	Command
Pull image from private registry	docker pull <private-registry-url>/<image-name>
Name an image	docker tag <old-name> <new-name>
Push an image	docker image push <image-name>
Login to private registry	docker login <private-registry-url>
Save image to file	docker save -o <file-name> <image-name>
Load image from file	docker load -i <file-name>

Creating your own Docker images

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Creating images with Dockerfiles



Text file with instructions to create an image

Blueprint to create a container

Running Image

Starting a Dockerfile

A Dockerfile always start from another image, specified using the FROM instruction.

```
FROM postgres
FROM ubuntu
FROM hello-world
FROM my-custom-data-pipeline
```

```
FROM postgres:15.0
FROM ubuntu:22.04
FROM hello-world:latest
FROM my-custom-data-pipeline:v1
```

Building a Dockerfile

Building a Dockerfile creates an image.

```
docker build /location/to/Dockerfile
docker build .
```

```
[+] Building 0.1s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 54B
...
 => CACHED [1/1] FROM docker.io/library/ubuntu
 => exporting to image
 => => exporting layers
 => => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
```

Naming our image

In practice we almost always give our images a name using the -t flag:

```
docker build -t first_image .
```

```
...
=> => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
=> => naming to docker.io/library/first_image
```

```
docker build -t first_image:v0 .
```

```
=> => writing image sha256:a67f41b1d127160a7647b6709b3789b1e954710d96df39ccaa21..
=> => naming to docker.io/library/first_image:v0
```

Customizing images

```
RUN <valid-shell-command>
```

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
```

Use the -y flag to avoid any prompts:

```
...
After this operation, 22.8 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Building a non-trivial Dockerfile

When building an image Docker actually runs commands after RUN

Docker running `RUN apt-get update` takes the same amount of time as us running it!

```
root@host:/# apt-get update
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease [270 kB]
...
Get:17 http://ports.ubuntu.com/ubuntu-ports jammy-security/restricted arm64 Pack..
Fetched 23.0 MB in 2s (12.3 MB/s)
Reading package lists... Done
```

Summary

Usage	Dockerfile Instruction
Start a Dockerfile from an image	FROM <image-name>
Add a shell command to image	RUN <valid-shell-command>
Make sure no user input is needed for the shell-command.	RUN apt-get install **-y** python3

Usage	Shell Command
Build image from Dockerfile	docker build /location/to/Dockerfile
Build image in current working directory	docker build **.**
Choose a name when building an image	docker build -t first_image .

Managing files in your image

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

COPYing files into an image

The COPY instruction copies files from our local machine into the image we're building:

```
COPY <src-path-on-host> <dest-path-on-image>
COPY /projects/pipeline_v3/pipeline.py /app/pipeline.py
```

```
docker build -t pipeline:v3 .
...
[4/4] COPY ./projects/pipeline_v3/pipeline.py /app/pipeline.py
```

If the destination path does not have a filename, the original filename is used:

```
COPY /projects/pipeline_v3/pipeline.py /app/
```

COPYing folders

Not specifying a filename in the src-path will copy all the file contents.

```
COPY <src-folder> <dest-folder>
COPY /projects/pipeline_v3/ /app/
```

`COPY /projects/pipeline_v3/ /app/` will copy everything under `pipeline_v3/` :

```
/projects/
    pipeline_v3/
        pipeline.py
        requirements.txt
        tests/
            test_pipeline.py
```

Copy files from a parent directory

```
/init.py
/projects/
    Dockerfile
    pipeline_v3/
        pipeline.py
```

If our current working directory is in the `projects/` folder.

We can't copy `init.py` into an image.

```
docker build -t pipeline:v3 .
 => ERROR [4/4] COPY ../init.py /     0.0s
failed to compute cache key: "../init.py" not found: not found
```

Downloading files

Instead of copying files from a local directory, files are often downloaded in the image build:

- Download a file

```
RUN curl <file-url> -o <destination>
```

- Unzip the file

```
RUN unzip <dest-folder>/<filename>.zip
```

- Remove the original zip file

```
RUN rm <copy_directory>/<filename>.zip
```

Downloading files efficiently

- Each instruction that downloads files adds to the total size of the image.
- Even if the files are later deleted.
- The solution is to download, unpack and remove files in a single instruction.

```
RUN curl <file_download_url> -o <destination_directory>/<filename>.zip \
&& unzip <destination_directory>/<filename>.zip -d <unzipped-directory> \
&& rm <destination_directory>/<filename>.zip
```

Summary

Usage	Dockerfile Instruction
Copy files from host to the image	COPY <src-path-on-host> <dest-path-on-image>
Copy a folder from host to the image	COPY <src-folder> <dest-folder>
We can't copy from a parent directory where we build a Dockerfile	~~COPY ../<file-in-parent-directory> /~~

Keep images small by downloading, unzipping, and cleaning up in a single RUN instruction:

```
RUN curl <file_download_url> -O <destination_directory> \
&& unzip <destination_directory>/<filename>.zip -d <unzipped-directory> \
&& rm <destination_directory>/<filename>.zip
```

Choosing a start command for your Docker image

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

What is a start command?

The hello-world image prints text and then stops.

```
docker run hello-world
```

```
Hello from Docker!

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

What is a start command?

An image with python could start python on startup.

```
docker run python3-sandbox
```

```
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
...
```

```
....
>>> exit()
repl@host:/#
```

Running a shell command at startup

```
CMD <shell-command>
```

The CMD instruction:

- Runs when the image is started.
- Does not increase the size of the image .
- Does not add any time to the build.
- If multiple exist, only the last will have an effect.

Typical usage

Starting an application to run a workflow or that accepts outside connections.

```
CMD python3 my_pipeline.py
```

```
CMD postgres
```

Starting a script that, in turn, starts multiple applications

```
CMD start.sh
```

```
CMD python3 start_pipeline.py
```

When will it stop?

- hello-world image -> After printing text
- A database image -> When the database exits

A more general image needs a more general start command.

- An Ubuntu image -> When the shell is closed

Overriding the default start command

Starting an image

```
docker run <image>
```

Starting an image with a custom start command

```
docker run <image> <shell-command>
```

Starting an image interactively with a custom start command

```
docker run -it <image> <shell-command>
```

```
docker run -it ubuntu bash
```

Summary

Usage	Dockerfile Instruction
Add a shell command run when a container is started from the image.	CMD <shell-command>

Usage	Shell Command
Override the CMD set in the image	docker run <image> <shell-command>
Override the CMD set in the image and run interactively	docker run -it <image> <shell-command>

Introduction to Docker layers and caching

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Docker build

Downloading and unzipping a file using the Docker instructions.

```
RUN curl http://example.com/example_folder.zip
RUN unzip example_folder.zip
```

Will change the file system and add:

```
/example_folder.zip
/example_folder/
    example_file1
    example_file2
```

It is these changes that are stored in the image.

Docker instructions are linked to File system changes

Each instruction in the Dockerfile is linked to the changes it made in the image file system.

```
FROM docker.io/library/ubuntu
 => Gives us a file system to start from with all files needed to run Ubuntu
```

```
COPY /pipeline/ /pipeline/
 => Creates the /pipeline/ folder
 => Copies multiple files in the /pipeline/ folder
```

```
RUN apt-get install -y python3
 => Add python3 to /var/lib/
```

Docker layers

- Docker layer: All changes caused by a single Dockerfile instruction.
- Docker image: All layers created during a build

--> Docker image: All changes to the file system by all Dockerfile instructions.

While building a Dockerfile, Docker tells us which layer it is working on:

```
=> [1/3] FROM docker.io/library/ubuntu
=> [2/3] RUN apt-get update
=> [3/3] RUN apt-get install -y python3
```

Docker caching

Consecutive builds are much faster because Docker re-uses layers that haven't changed.

Re-running a build:

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> CACHED [3/3] RUN apt-get install -y python3
```

Re-running a build but with changes:

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> [3/3] RUN apt-get install -y R
```

Understanding Docker caching

When layers are cached helps us understand why sometimes images don't change after a rebuild.

- Docker can't know when a new version of python3 is released.
- Docker will use cached layers because the instructions are identical to previous builds.

```
=> [1/3] FROM docker.io/library/ubuntu
=> CACHED [2/3] RUN apt-get update
=> CACHED [3/3] RUN apt-get install -y python3
```

Understanding Docker caching

Helps us write Dockerfiles that build faster because not all layers need to be rebuilt.

In the following Dockerfile all instructions need to be rebuild if the pipeline.py file is changed:

```
FROM ubuntu
COPY /app/pipeline.py /app/pipeline.py
RUN apt-get update
RUN apt-get install -y python3
```

```
 => [1/4] FROM docker.io/library/ubuntu
 => [2/4] COPY /app/pipeline.py /app/pipeline.py
 => [3/4] RUN apt-get update
 => [4/4] RUN apt-get install -y python3
```

Understanding Docker caching

Helps us write Dockerfiles that build faster because not all layers need to be rebuilt.

In the following Dockerfile, only the COPY instruction will need to be re-run.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
COPY /app/pipeline.py /app/pipeline.py
```

```
=> [1/4] FROM docker.io/library/ubuntu
=> CACHED [2/4] RUN apt-get update
=> CACHED [3/4] RUN apt-get install -y python3
=> [4/4] COPY /app/pipeline.py /app/pipeline.py
```

Changing users and working directory

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Dockerfile instruction interaction

FROM, RUN, and COPY interact through the file system.

```
COPY /projects/pipeline_v3/start.sh /app/start.sh
RUN /app/start.sh
```

Some influence other instructions directly:

- `WORKDIR` : Changes the working directory for all following instructions
- `USER` : Changes the user for all following instructions

WORKDIR - Changing the working directory

Starting all paths at the root of the file system:

```
COPY /projects/pipeline_v3/ /app/
```

Becomes cluttered when working with long paths:

```
COPY /projects/pipeline_v3/ /home/my_user_with_a_long_name/work/projects/app/
```

Alternatively, use WORKDIR:

```
WORKDIR /home/my_user_with_a_long_name/work/projects/
```

```
COPY /projects/pipeline_v3/ app/
```

RUN in the current working directory

Instead of using the full path for every command:

```
RUN /home/repl/projects/pipeline/init.sh
RUN /home/repl/projects/pipeline/start.sh
```

Set the WORKDIR:

```
WORKDIR /home/repl/projects/pipeline/
RUN ./init.sh
RUN ./start.sh
```

Changing the startup behavior with WORKDIR

Instead of using the full path:

```
CMD /home/repl/projects/pipeline/start.sh
```

Set the WORKDIR:

```
WORKDIR /home/repl/projects/pipeline/
CMD start.sh
```

Overriding command will also be run in WORKDIR:

```
docker run -it pipeline_image start.sh
```

Linux permissions

- Permissions are assigned to users.
- Root is a special user with all permissions.

Best practice

- Use root to create new users with permissions for specific tasks.
- Stop using root.

Changing the user in an image

Best practice: Don't run everything as root

Ubuntu -> root by default

```
FROM ubuntu           --> Root user by default
RUN apt-get update    --> Run as root
```

USER Dockerfile instruction:

```
FROM ubuntu           --> Root user by default
USER repl             --> Changes the user to repl
RUN apt-get update    --> Run as repl
```

Changing the user in a container

Dockerfile setting the user to repl:

```
FROM ubuntu           --> Root user by default
USER repl             --> Changes the user to repl
RUN apt-get update    --> Run as repl
```

Will also start containers with the repl user:

```
docker run -it ubuntu bash
repl@container: whoami
repl
```

Summary

Usage	Dockerfile Instruction
Change the current working directory	WORKDIR <path>
Change the current user	USER <user-name>

Variables in Dockerfiles

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Variables with the ARG instruction

Create variables in a Dockerfile

```
ARG <var_name>=<var_value>
```

For example `ARG path=/home/repl`

To use in the Dockerfile

```
$path
```

For example `COPY /local/path $path`

Use-cases for the ARG instruction

Setting the Python version

```
FROM ubuntu
ARG python_version=3.9.7-1+bionic1
RUN apt-get install python3=$python_version
RUN apt-get install python3-dev=$python_version
```

Configuring a folder

```
FROM ubuntu
ARG project_folder=/projects/pipeline_v3
COPY /local/project/files $project_folder
COPY /local/project/test_files $project_folder/tests
```

Setting ARG variables at build time

```
FROM ubuntu
ARG project_folder /projects/pipeline_v3
COPY /local/project/files $project_folder
COPY /local/project/test_files $project_folder/tests
```

Setting a variable in the build command

```
docker build --build-arg project_folder=/repl/pipeline .
```

ARG is overwritten, and files end up in:

```
COPY /local/project/files /repl/pipeline
COPY /local/project/test_files /repl/pipeline/tests
```

Variables with ENV

Create variables in a Dockerfile

```
ENV <var_name>=<var_value>
```

For example `ENV DB_USER=pipeline_user`

To use in the Dockerfile or at runtime

```
$DB_USER
```

For example `CMD psql -U $DB_USER`

Use-cases for the ENV instruction

Setting a directory to be used at runtime

```
ENV DATA_DIR=/usr/loca/var/postgres
```

```
ENV MODE production
```

Setting or replacing a variable at runtime

```
docker run --env <key>=<value> <image-name>
```

```
docker run --env POSTGRES_USER=test_db --env POSTGRES_PASSWORD=test_db postgres
```

[1] https://hub.docker.com/_/postgres

Secrets in variables are not secure

`docker history <image-name>`

```
ARG DB_PASSWORD=example_password
```

Will show in `docker history` :

```
IMAGE          CREATED        CREATED BY                          SIZE      ...
cd338027297f   2 months ago   ARG DB_PASSWORD=example_password    0B        ...
```

Summary

Usage	Dockerfile Instruction
Create a variable accessible only during the build	ARG <name>=<value>
Create a variable	ENV <name>=<value>

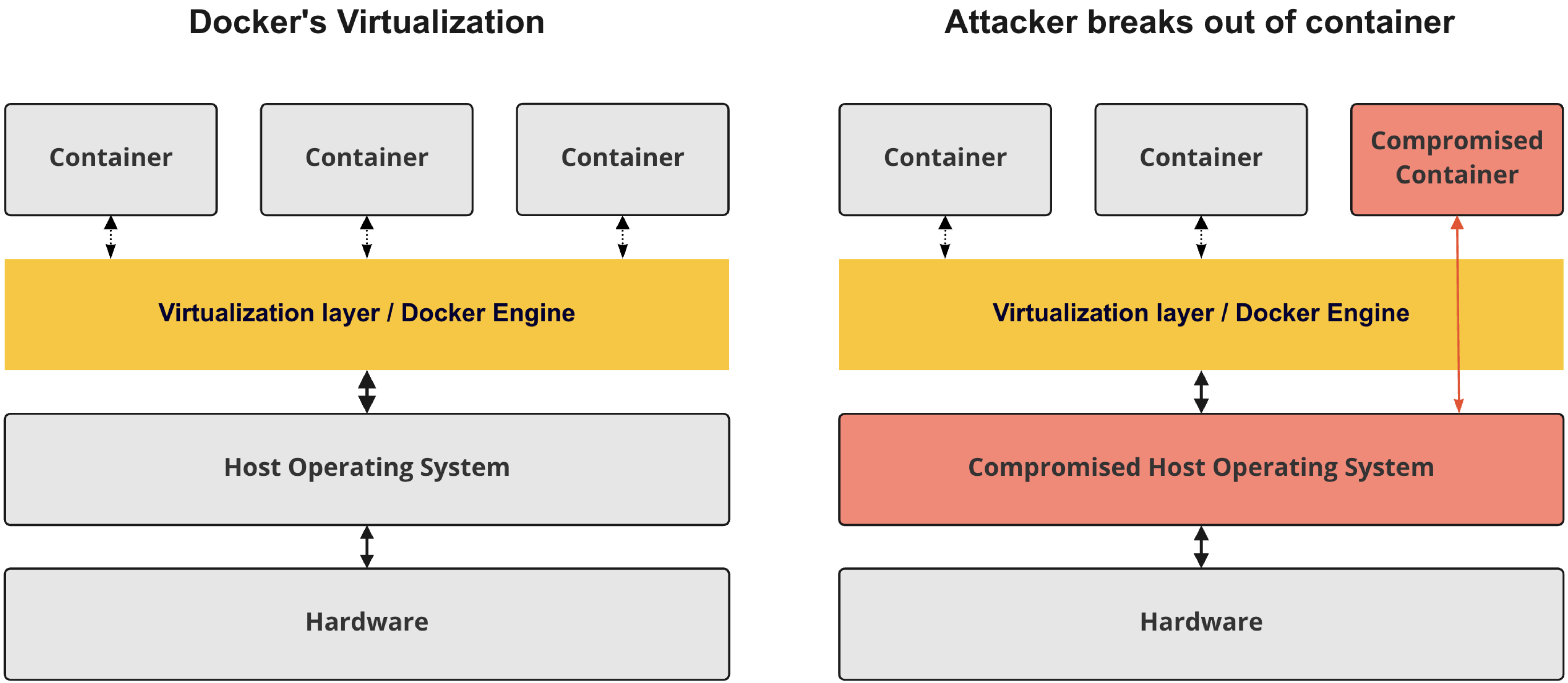
Usage	Shell Command
Override an ARG in docker build	docker build --build-arg <name>=<value>
Override an ENV in docker run	docker run --env <name>=<value> <image-name>
See the instructions used to create an image	docker history <image-name>

Creating Secure Docker Images

INTRODUCTION TO DOCKER

Tim Sangster
Software Engineer @ DataCamp

Inherent Security



Docker's Virtualization

Container

Container

Container

Virtualization layer / Docker Engine

Host Operating System

Hardware

Attacker breaks out of container

Container

Container

Compromised Container

Virtualization layer / Docker Engine

Compromised Host Operating System

Hardware

Making secure images

Attackers can exceptionally break out of a container.

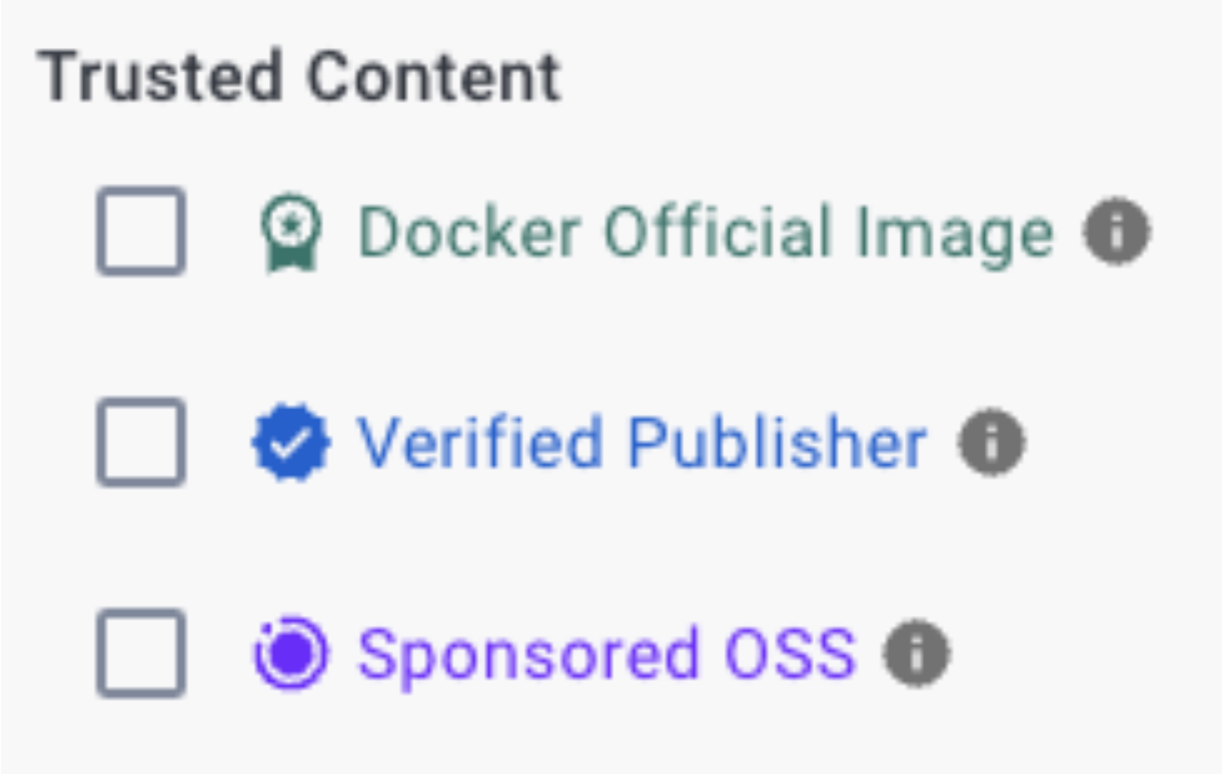
Additional security measures can lower this risk

Becomes especially important once exposing running containers to the Internet.

Images from a trusted source

Creating secure images -> Start with an image from a trusted source

Docker Hub filters:



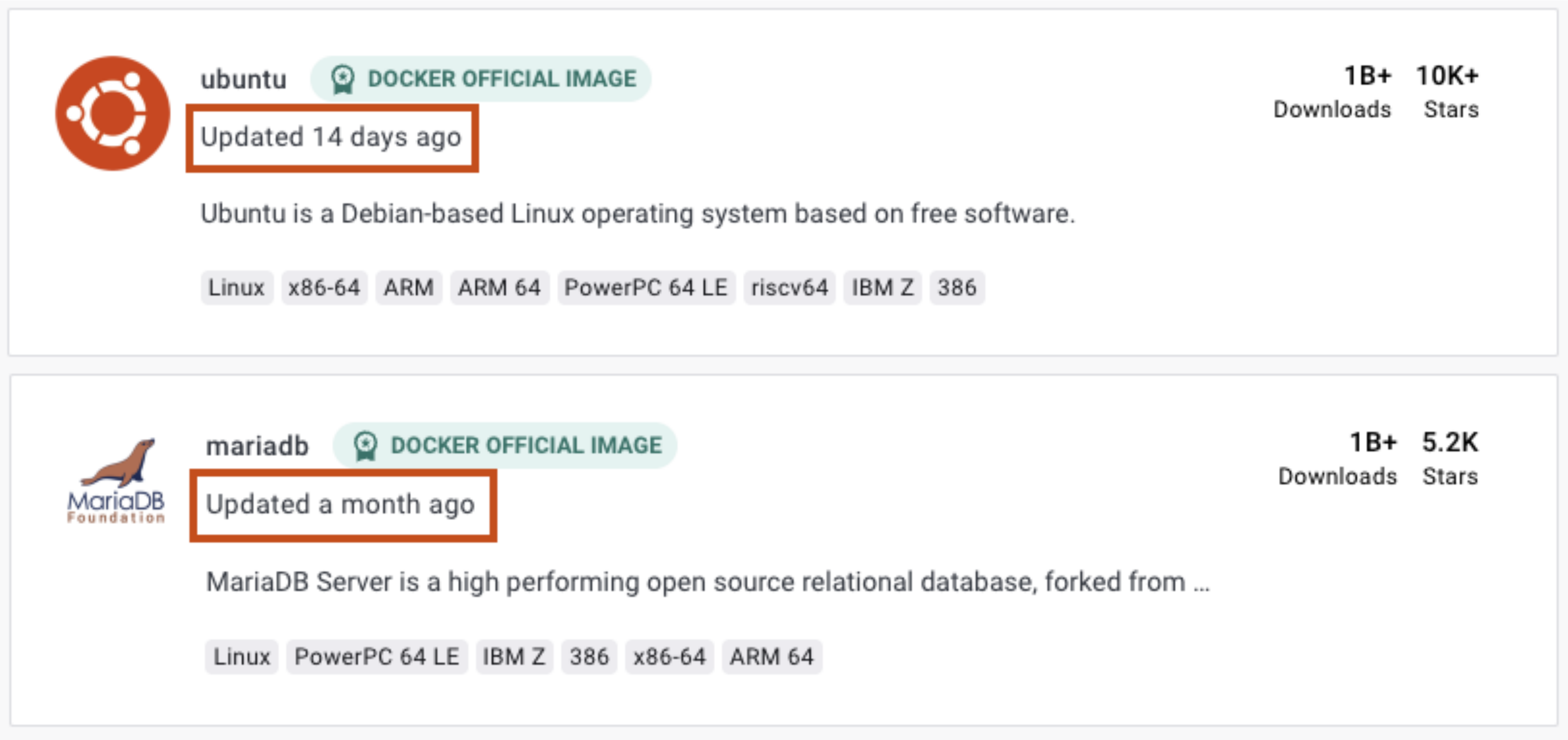
Trusted Content

□ Docker Official Image

□ Verified Publisher

□ Sponsored OSS

Keep software up-to-date



ubuntu DOCKER OFFICIAL IMAGE

Updated 14 days ago

1B+ Downloads 10K+ Stars

Ubuntu is a Debian-based Linux operating system based on free software.

Linux x86-64 ARM ARM 64 PowerPC 64 LE riscv64 IBM Z 386

mariadb DOCKER OFFICIAL IMAGE

Updated a month ago

1B+ Downloads 5.2K Stars

MariaDB Server is a high performing open source relational database, forked from ...

Linux PowerPC 64 LE IBM Z 386 x86-64 ARM 64

Keep images minimal

Adding unnecessary packages reduces security

Ubuntu with:

- Python2.7
- Python3.11
- Java default-jre
- Java openjdk-11
- Java openjdk-8
- Airflow
- Our pipeline application

Installing only essential packages improves security

Ubuntu with:

- Python3.11
- Our pipeline application

Don't run applications as root

Allowing root access to an image defeats keeping the image up-to-date and minimal.

Instead, make containers start as a user with fewer permissions:

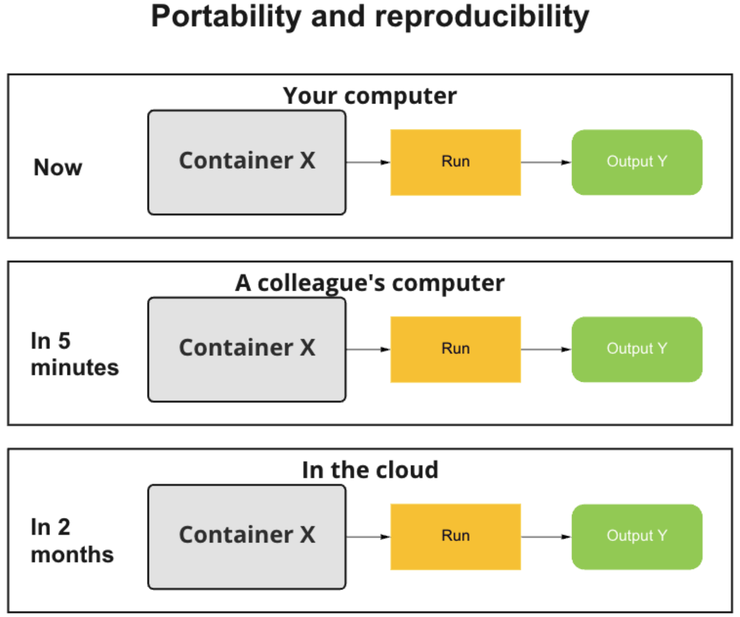
```
FROM ubuntu # User is set to root by default.
RUN apt-get update
RUN apt-get install python3
USER repl # We switch the user after installing what we need for our use-case.
CMD python3 pipeline.py
```

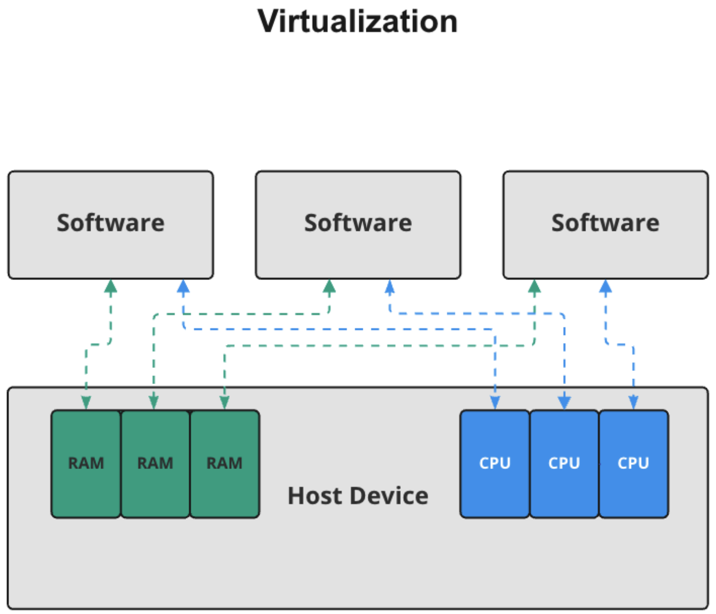
Wrap-up

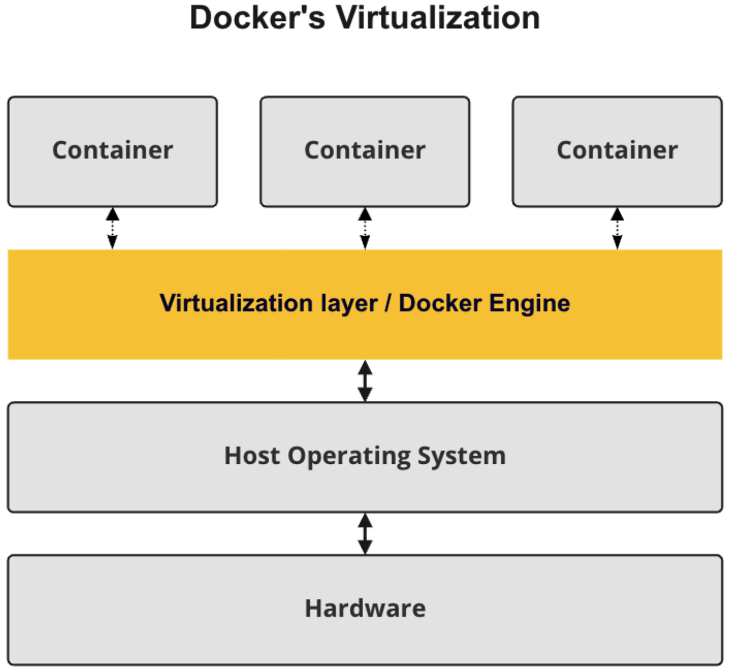
INTRODUCTION TO DOCKER

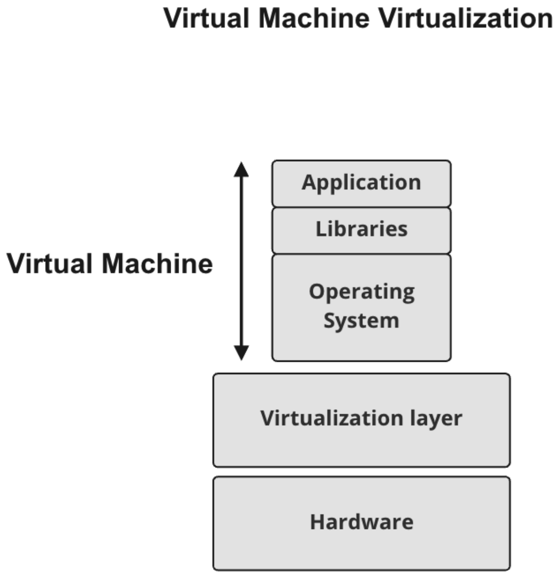
Tim Sangster
Software Engineer @ DataCamp

Chapter 1: The theoretical foundation









Chapter 2: The Docker CLI

Usage	Command
Start a container	docker run (--name <container-name>) (-it) (-d) <image-name>
List running containers	docker ps (-f "name=<container-name>")
Stop a container	docker stop <container-id>
See (live) logs for container	docker logs (-f) <container-id>
Remove stopped container	docker container rm <container-id>
Pull a specific version of an image	docker pull <image-name>:<image-version>
List all local images	docker images
Remove an image	docker image rm <image-name>

Chapter 3: Dockerfiles

```
FROM ubuntu
RUN apt-get update && apt-get install python3
COPY /projects/pipeline /app/
CMD /app/init.py
```

```
docker build -t my_pipeline .
 => [1/3] FROM docker.io/library/ubuntu
 => CACHED [2/3] RUN apt-get update && apt-get install python3
 => CACHED [3/3] COPY /projects/pipeline /app/
```

Chapter 4: Security and Customization

Usage	Dockerfile Instruction
Change the current working directory	WORKDIR <path>
Change the current user	USER <user-name>
Create a variable accessible only during the build	ARG <name>=<value>
Create a variable	ENV <name>=<value>

Usage	Shell Command
Override an ARG in docker build	docker build --build-arg <name>=<value>
Override an ENV in docker run	docker run --env <name>=<value> <image-name>
See the instructions used to create a image	docker history <image-name>

Chapter 4: Security and Customization

- Isolation provided by containers gives security but is not perfect.
- Use the "Trusted Content" images from the official Docker Hub registry
- Keep software on images up-to-date
- Only install the software you need for the current use case.
- Do not leave the user in images set to root.

What more is there to learn?

Dockerfile instructions

- `ENTRYPOINT`
- `HEALTHCHECK`
- `EXPOSE`
- ...

Multi stage builds

```
FROM ubuntu as stage1
RUN generate_data.py
...
FROM postgres as stage2
COPY --from=stage 1 /tmp /data
```