

Data?

Application (Code + Environment)

Written & provided by
you (= the developer)

Added to image and
container in build phase

“Fixed”: Can’t be changed
once image is built

Read-only, hence stored
in Images

Temporary App Data (e.g. entered user input)

Fetches / Produced in
running container

Stored in memory or
temporary files

Dynamic and changing,
but cleared regularly

Read + write,
temporary, hence stored
in Containers

Permanent App Data (e.g. user accounts)

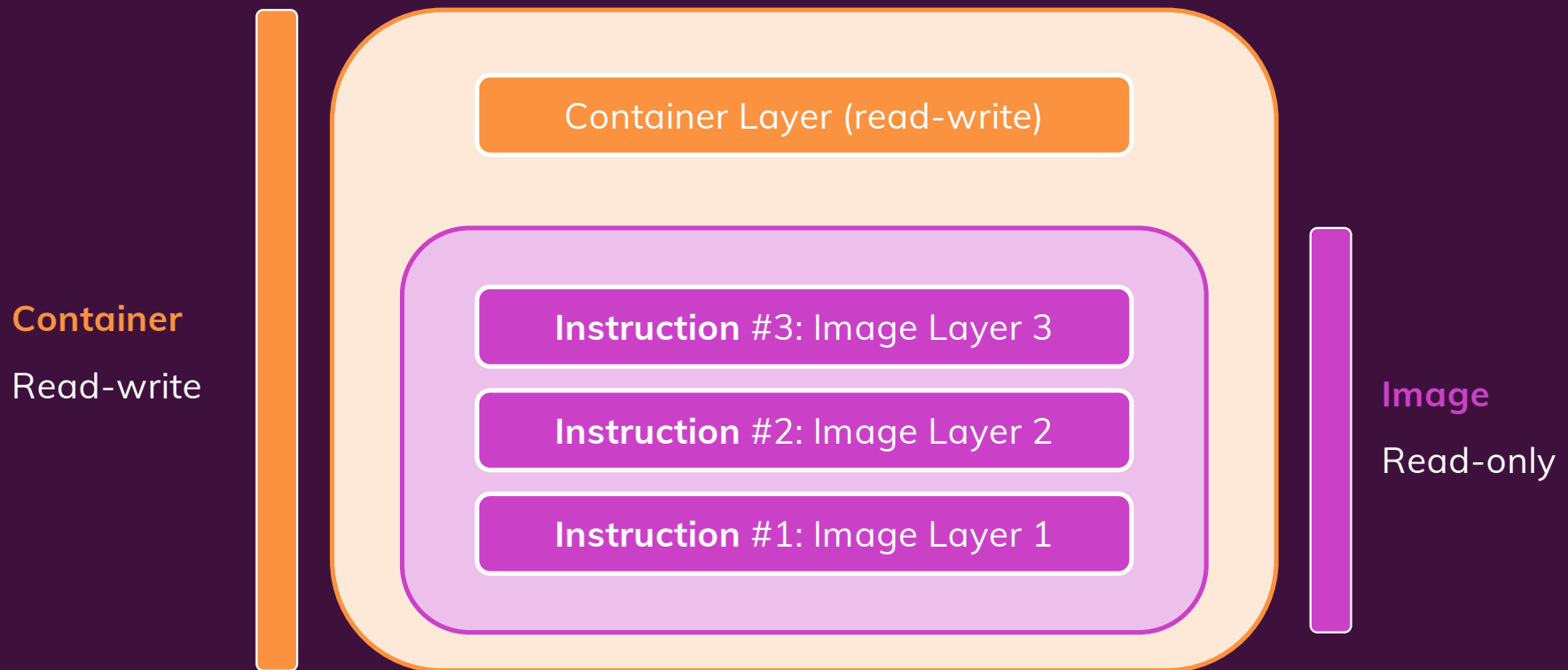
Fetches / Produced in
running container

Stored in files or a
database

Must not be lost if
container stops / restarts

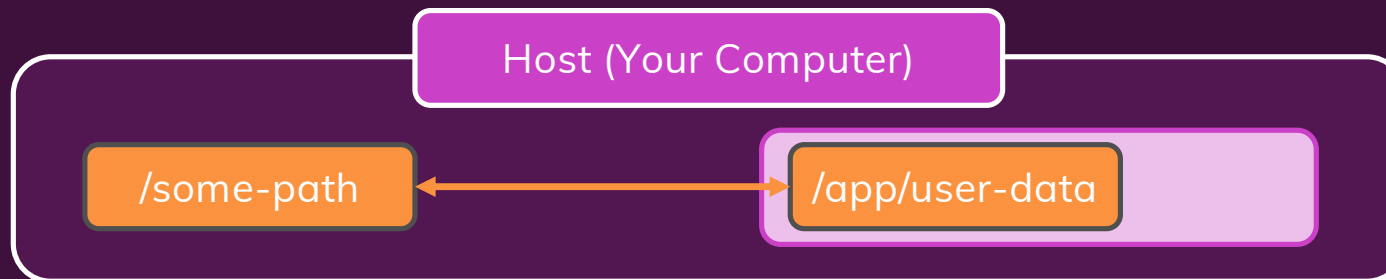
Read + write,
permanent, stored with
Containers & Volumes

A Container Is Based On An Image



Understanding Volumes

Volumes are **folders on your host machine** hard drive which are **mounted** ("made available", mapped) **into containers**



Volumes **persist if a container shuts down**. If a container (re-)starts and mounts a volume, any data inside of that volume is **available in the container**.

A container **can write data** into a volume **and read data** from it.

Two Types of External Data Storages

Volumes
(Managed by Docker)

Anonymous Volumes

Named Volumes

Docker sets up a folder / path on your host machine,
exact location is unknown to you (= dev).
Managed via *docker volume* commands.

A defined path in the container is mapped to the created volume / mount.
e.g. */some-path* on your *hosting machine* is mapped to */app/data*

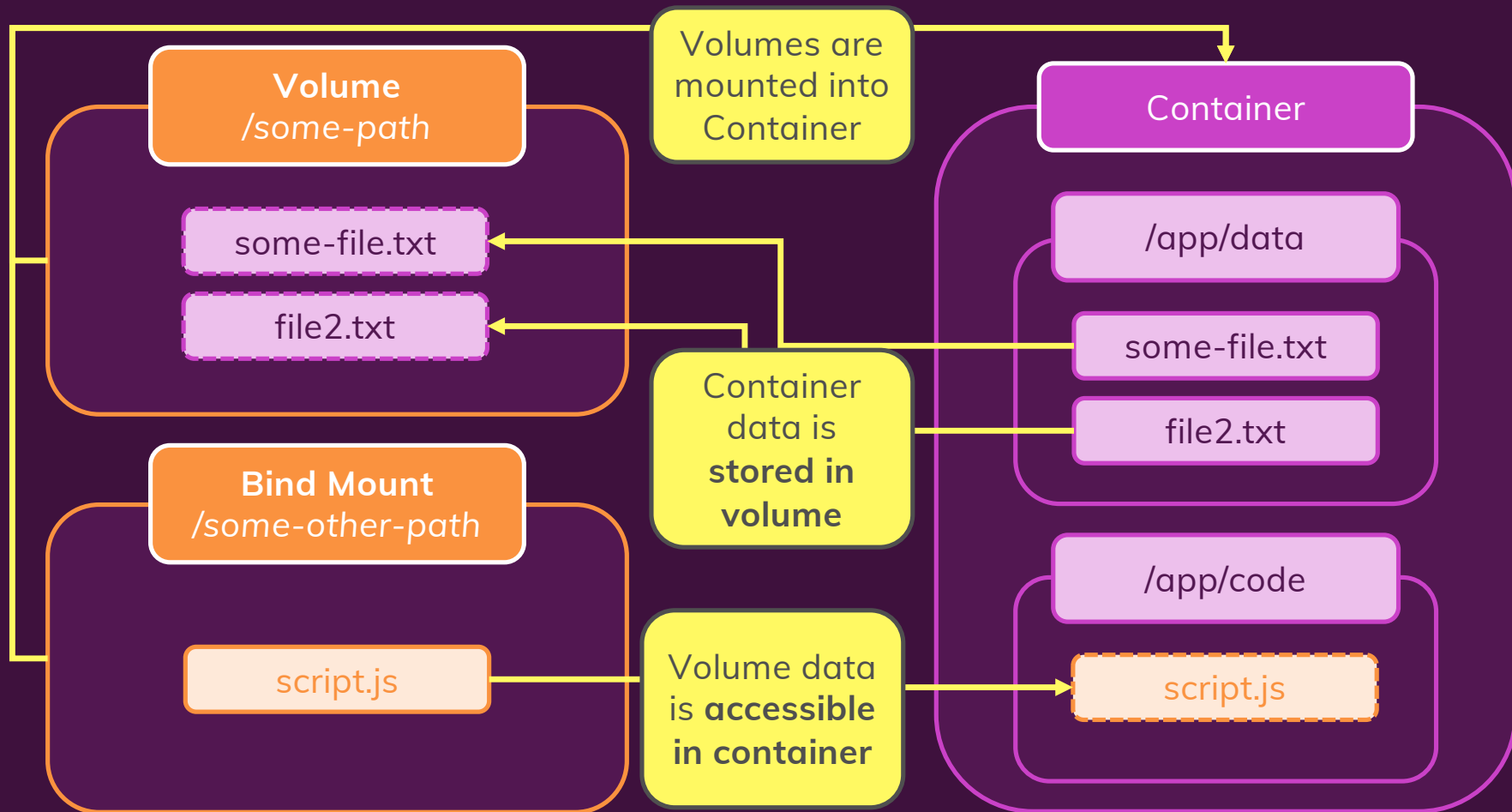
Great for data which
should be persistent
but which you don't
need to edit directly.

Bind Mounts
(Managed by you)

You define a folder / path on
your host machine.

Great for persistent, editable
(by you) data
(e.g. source code).

Understanding Container / Volume Interaction



Volumes & Bind Mounts – Quick Overview

```
docker run -v /app/data ...
```



Anonymous Volume

```
docker run -v data:/app/data ...
```



Named Volume

```
docker run -v /path/to/code:/app/code ...
```



Bind Mount

Volumes – Comparison

Anonymous Volumes

Created specifically for a single container

Survives container shutdown / restart unless --rm is used

Can not be shared across containers

Since it's anonymous, it can't be re-used (even on same image)

Named Volumes

Created in general – not tied to any specific container

Survives container shutdown / restart – removal via Docker CLI

Can be shared across containers

Can be re-used for same container (across restarts)

Bind Mounts

Location on host file system, not tied to any specific container

Survives container shutdown / restart – removal on host fs

Can be shared across containers

Can be re-used for same container (across restarts)

ARGuments & ENVironment Variables

Docker supports build-time ARGuments and runtime ENVironment variables

ARG

Available inside of Dockerfile, NOT accessible in CMD or any application code

Set on image build (**docker build**) via **--build-arg**

ENV

Available inside of Dockerfile & in application code

Set via ENV in Dockerfile or via **--env** on **docker run**

Module Summary

Containers can read + write data. **Volumes** can help with data storage, **Bind Mounts** can help with direct container interaction.

Containers can read + write data, but written **data is lost** if the container is removed

Volumes are folders on the host machine, managed by Docker, which are mounted into the Container

Named Volumes survive container removal and can therefore be used to store persistent data

Anonymous Volumes are attached to a container – they can be used to save (temporary) data inside the container

Bind Mounts are folders on the host machine which are specified by the user and mounted into containers – like **Named Volumes**

Build ARGuments and **Runtime ENVironment** variables can be used to make images and containers more **dynamic / configurable**

Read-Only, Read-Write & Volumes

Images

Read-only

Once created, you need to re-build them to change something

Application data (e.g. user data) is NOT stored in images

Containers

Read & Write

A running container can store data (e.g. incoming user data)

But: Data is lost when a container stops

Solution for persistent data: **Volumes**