Docker Core Concepts

Containers

Read-write layer on top of image

- Isolated
- Single-task-focused
- Shareable, reproducible
- Stateless (+ volumes)

Images

Created with instructions (layers)

- Blueprints for Containers
- Code + environment
- Read-only / does not run
- Can be built + shared

ACADE MIND

Key Commands

Build an image based on a **Dockerfile**

```
docker build -t NAME:TAG .
```

Name & versions of an image

Build context

Run a container based on a **remote or local Image**

```
docker run --name NAME --rm -d IMAGE
```

Container name

Remove once stopped

Detached mode

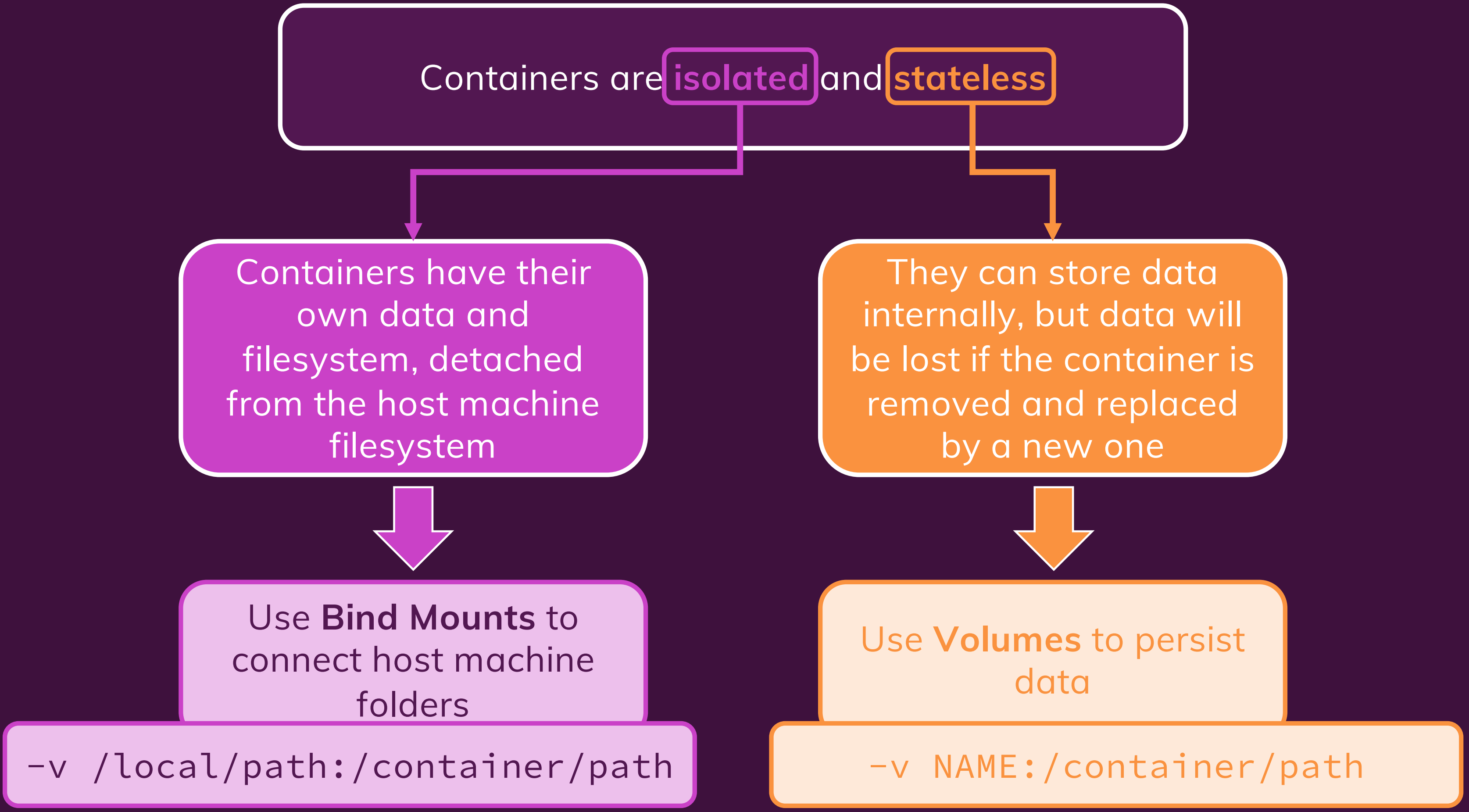
Share (push) an Image to a **Registry** (default: **DockerHub**)

```
docker push REPOSITORY/NAME:TAG
```

Fetch (pull) an Image from a **Registry** (default: **DockerHub**)

```
docker pull REPOSITORY/NAME:TAG
```

Docker Containers & Data



Containers are **isolated** and **stateless**

Containers have their own data and filesystem, detached from the host machine filesystem

They can store data internally, but data will be lost if the container is removed and replaced by a new one

Use **Bind Mounts** to connect host machine folders

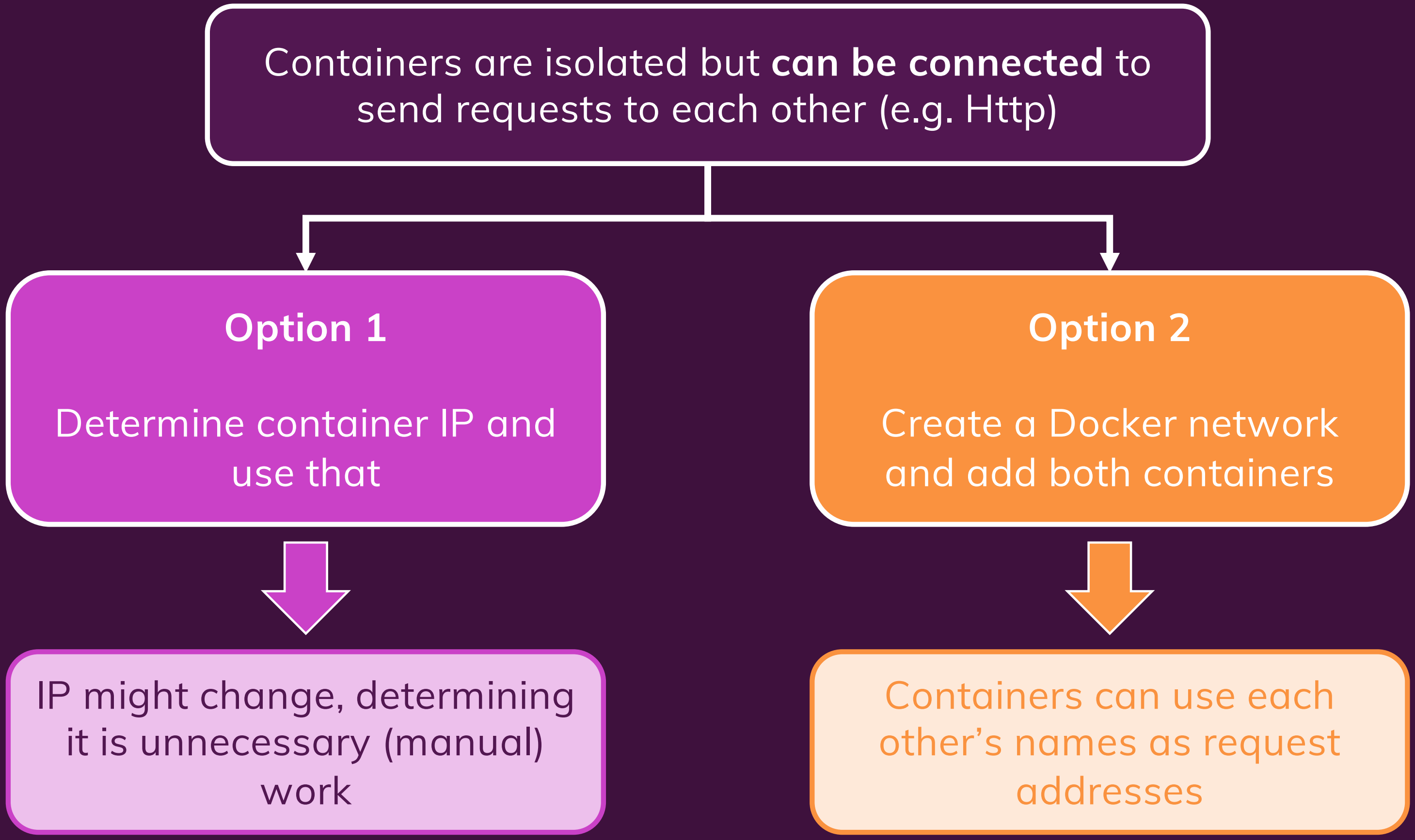
```
-v /local/path:/container/path
```

Use **Volumes** to persist data

```
-v NAME:/container/path
```

ACADE MIND

Docker Containers & Networks



Containers are isolated but **can be connected** to send requests to each other (e.g. Http)

Option 1

Determine container IP and use that

IP might change, determining it is unnecessary (manual) work

Option 2

Create a Docker network and add both containers

Containers can use each other's names as request addresses

ACADE MIND

Docker vs Docker Compose

Repeating long **`docker build`** and **`docker run`** commands gets annoying – especially when working **with multiple containers**

Docker Compose allows you to pre-define build and run configuration in a .yaml file

`docker-compose up`

Build missing images and start all containers

`docker-compose down`

Stop all started containers

ACADE MIND

Local Host (Development) vs Remote Host (Production)

Local Host / Development	Remote Host / Production
Isolated, encapsulated, reproducible development environments	Isolated, encapsulated, reproducible environments
No dependency or software clashes	Easy updates: Simply replace a running container with an updated one

Develop your application in the same environment you'll run it in after deployment

Deployment Is Optional!

It’s perfectly fine to use Docker (and Docker Compose) for local development!

- **Encapsulated** environments for different projects
- **No global installation** of tools
- **Easy to share** and re-produce

Deployment Considerations

Replace Bind Mounts with Volumes or COPY

Multiple containers might need multiple hosts

But they can also run on the same host (depends on application)

Multi-stage builds help with apps that need a build step

Control vs Ease-of-use

You can launch a remote server, install Docker and run your containers

Full control but you also need to manage everything

You can use a managed service instead

Less control and extra knowledge required but easier to use, less responsibility