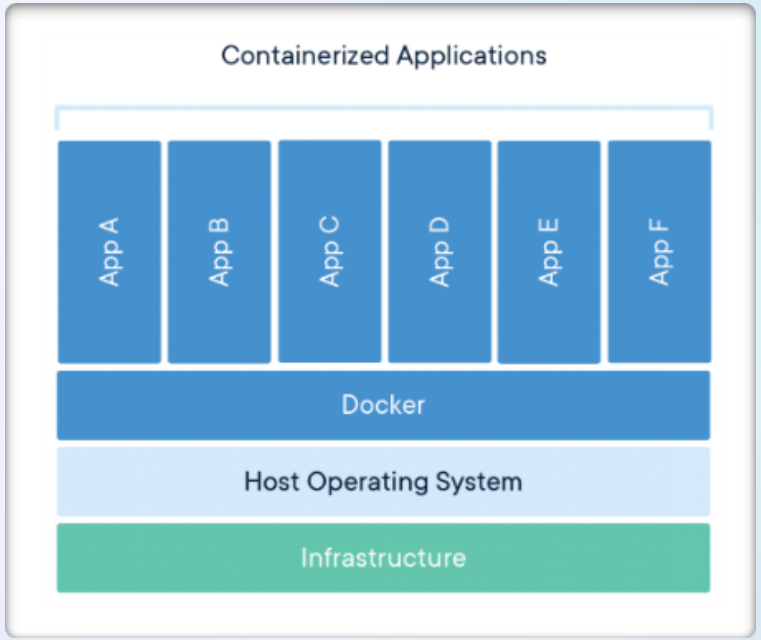
Docker basic usage

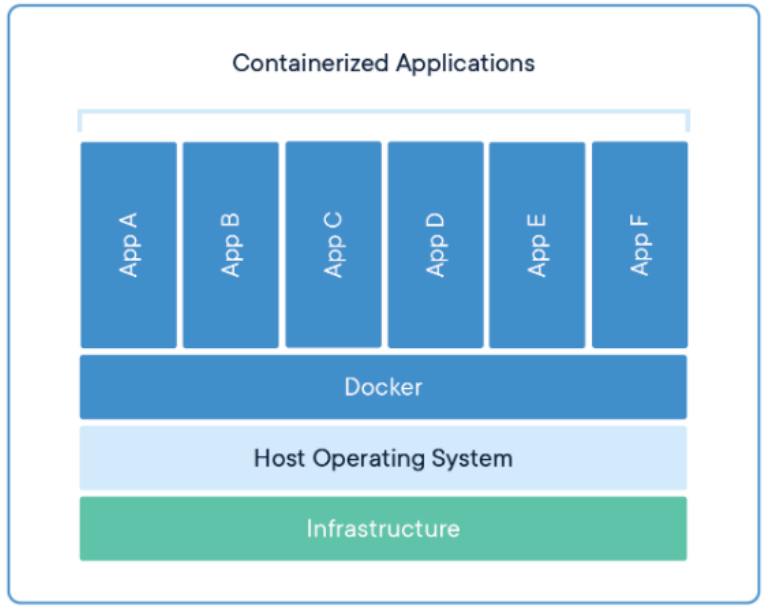
Containerization

Container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

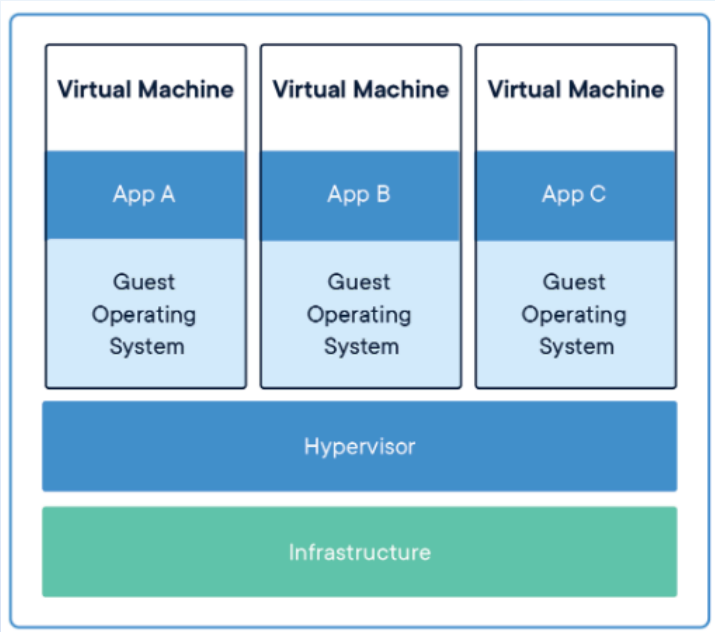
A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.



Containers vs. Virtual Machines



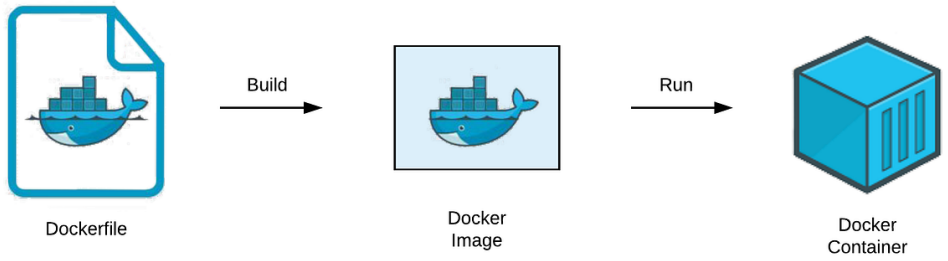
- abstraction at the app layer that packages code and dependencies together
- run multiple containers as isolated processes
- less space than VMs



- abstraction of physical hardware turning one server into many servers.
- each VM includes a full copy of an operating system, the application, necessary binaries and libraries. Takes up a lot of space
- can be slow to boot

CODEUS_

Dockerfile, Image, Container



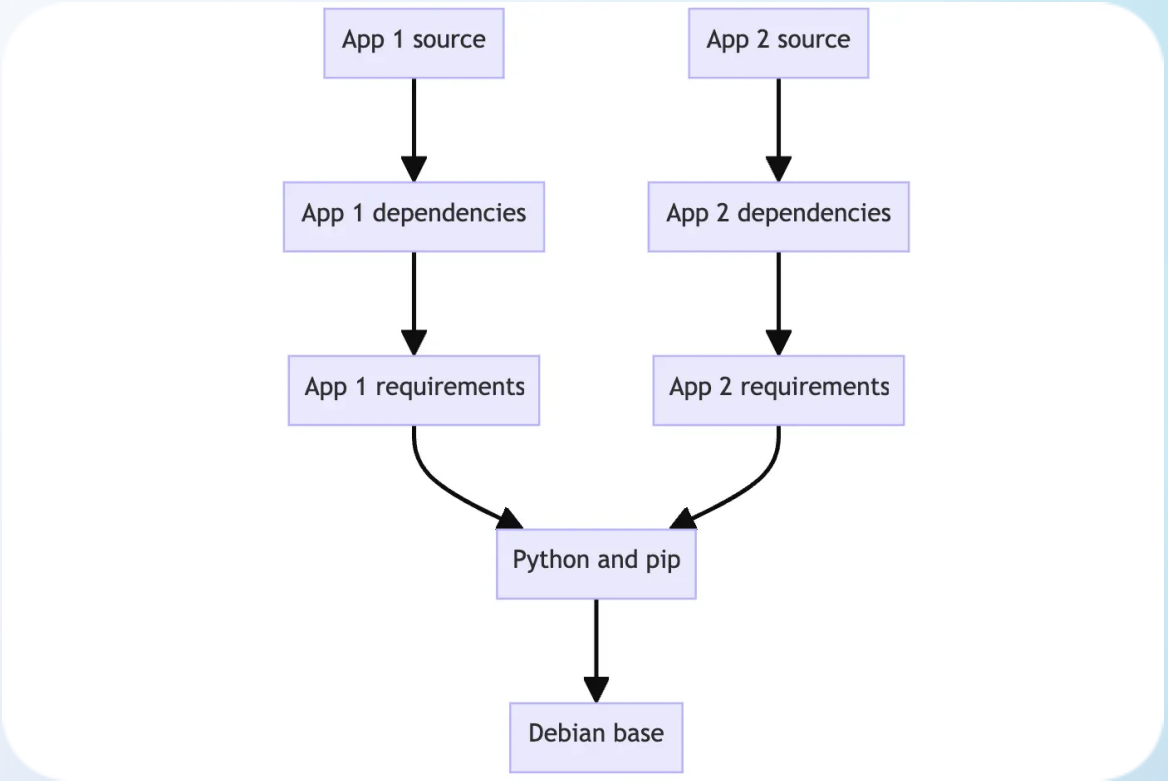
- Dockerfile contains a source code for a Docker image creation
- Docker image is a blueprint with instructions for a container creation. Like a recipe including code, libraries, environment needed to run your application

Image Layers

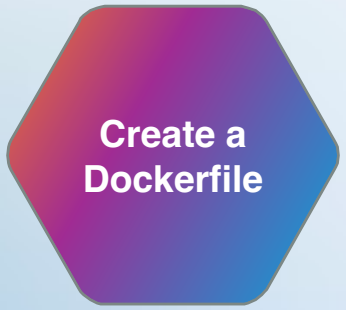
An image is composed of layers. Each Dockerfile instruction corresponds to a separate layer.

Each layer in an image contains a set of filesystem changes - additions, deletions, or modifications.

The usage of immutable layers helps containers to be lightweight and reuse the same base images.

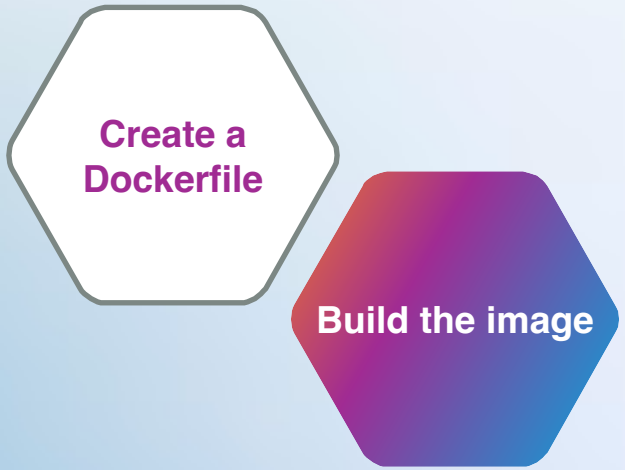


Dockerfile, Image, Container



- Dockerfile contains a source code for a Docker image creation

Dockerfile, Image, Container

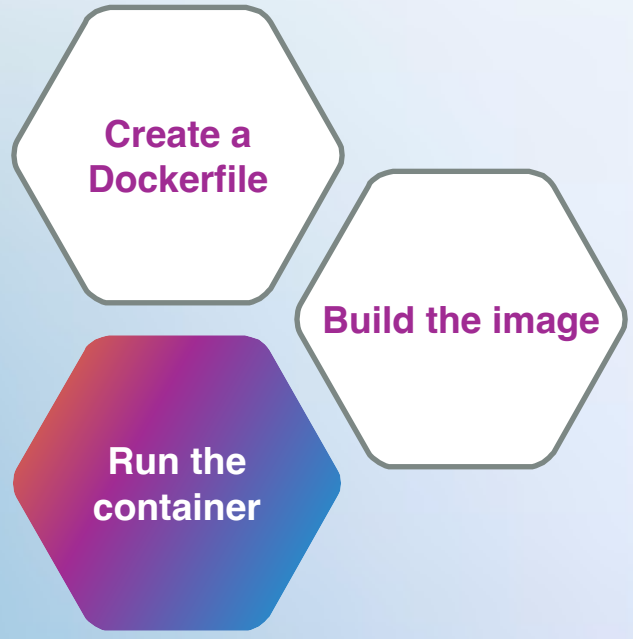


- Dockerfile contains a source code for a Docker image creation
- Docker image is a blueprint with instructions for a container creation. Like a recipe including code, libraries, environment needed to run your application

Command example:

```
docker build -t my-app:1.0 .
```

Dockerfile, Image, Container



- Dockerfile contains a source code for a Docker image creation
- Docker image is a blueprint with instructions for a container creation. Like a recipe including code, libraries, environment needed to run your application

Command example:

```
docker build -t my-app:1.0 .
```

- Start a Docker container based on the image

Command example:

```
docker run -p 8080:8080 -d --rm --name app-container my-app:1.0
```

Bind mounts

- Bind mount is a way to mount a file of directory on host into a container. This leads to a container being able to access the file/ directory from your host.

Bind mounts are often used when:

- you need to create files in container and persist them on host
- you need to share configuration files from the host machine to containers

Considerations:

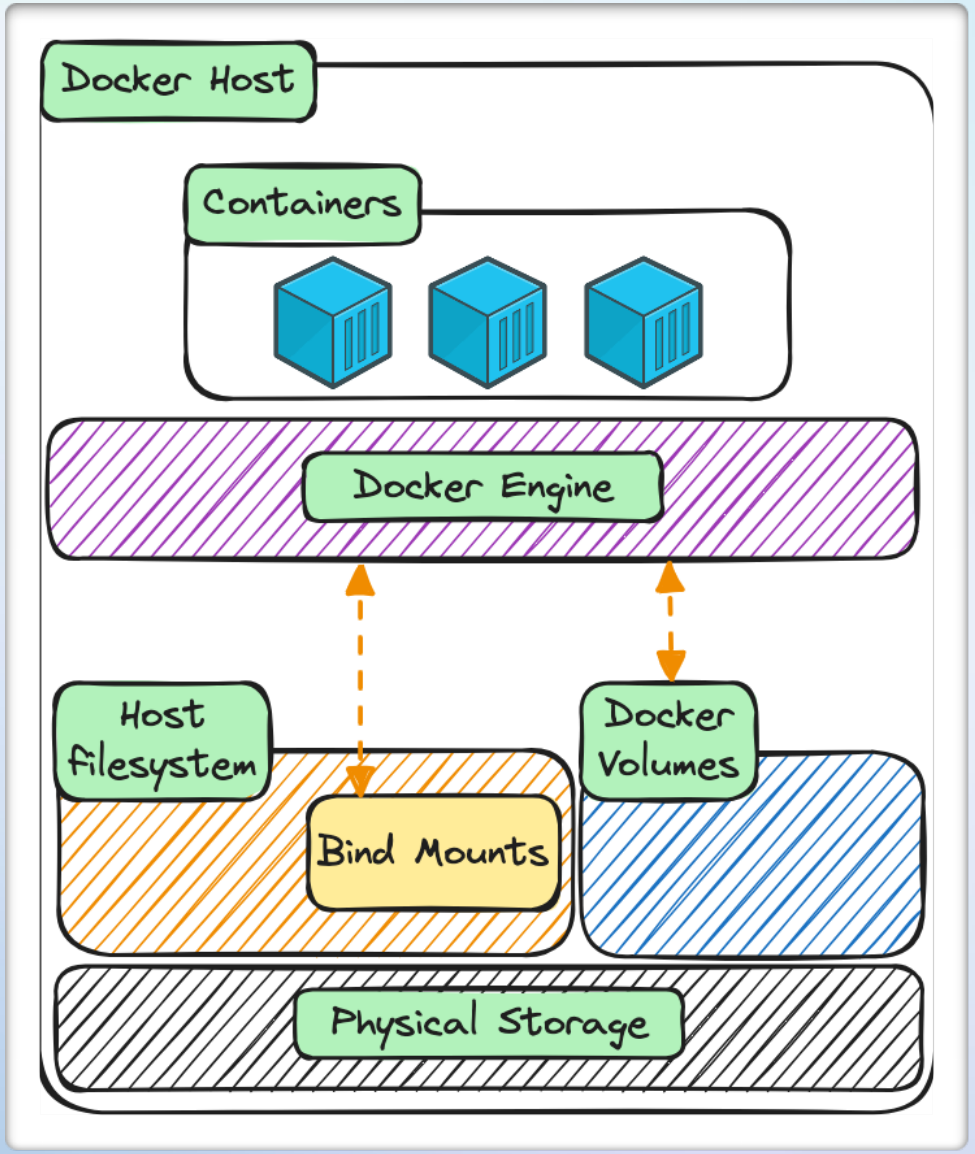
- bind mounts have write access to files on the host by default. Use `readonly` or `ro` option to prevent it when needed
- containers with bind mount are strongly tied to the host. The container may fail if run on a different host with different directory structure

Example:

```
docker run -dit --name my-apache-app -p 8080:80 -v “$PWD”:/usr/local/apache2/htdocs/:ro httpd:2.4
```

Alternatives:

- Volumes
- tmpfs mounts



CODEUS_

Useful commands cheatsheet

Command	Options	Description	Example
`pwd`		Print working directory	`pwd`
`ls [directory]`		List directory contents	`ls /`
	`-R`	Recursively list subdirectories	`ls -R ~/Downloads`
`cat [file]`		Concatenate and print files	`cat index.html`
`docker build [options] PATH`		Build an image from Dockerfile	`docker build -t app:1.0 .`
	`-t, --tag`	Name and optionally tag in the `name:tag` format	
`docker run [options] IMAGE [command] [arg…]`		Create and run a new container from an image	`docker run --name test -d nginx:alpine`
	`-d, --detach`	Run a container in background and print container ID	`-d`
	`-i, --interactive`	Keep STDIN open even if not attached	`--interactive --tty`
	`-t, --tty`	Allocate a pseudo-TTY	`-it`
	`--name`	Assign a name to the container	`--name test`
	`-p, –publish`	Publish a container's port(s) to the host in in the `host_port:container_port` format	`-p 80:8080`
	`--rm`	Automatically remove the container and its associated anonymous volumes when it exits	`--rm`
	`-v, --volume`	Bind mount a volume	`-v $(pwd):$(pwd)`

Useful commands cheatsheet

Command	Options	Description	Example
`docker exec [options] CONTAINER COMMAND [args…]`		Execute a command in a running container	`docker exec -it my-container bash`
	`-i, --interactive`	Keep STDIN open even if not attached	`--interactive --tty`
	`-t, --tty`	Allocate a pseudo-TTY	`-it`
`docker logs`		Fetch the logs of a container	`docker logs my-container`
	`-f ,--follow`	Follow log output	`-f`

Thank you

- Author: Yevhenii Savonenko
- My LinkedIn: www.linkedin.com/in/yevhenii-savonenko-624a32172
- Date: January 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn