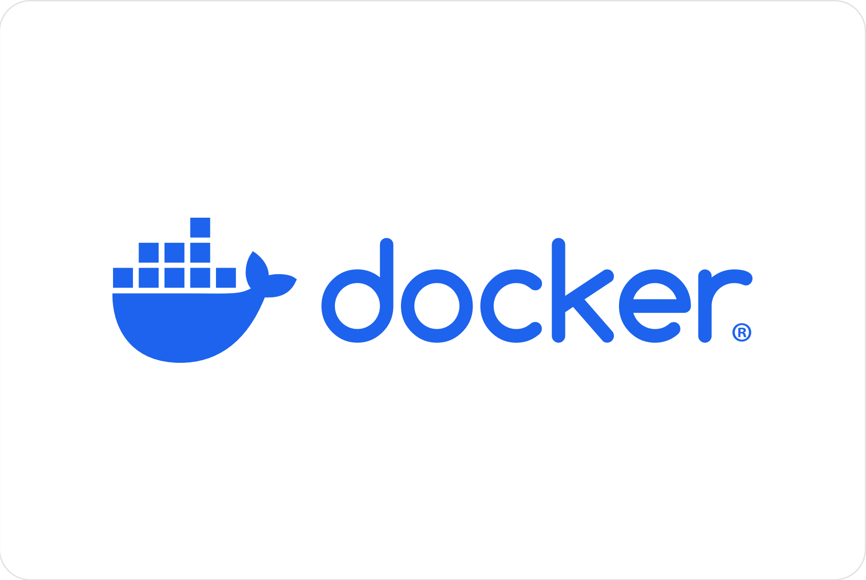
Intro to Docker

What is Docker?

- software that allows you to run software in an isolated environment
- enviroment is built using code and therefore reproducible
- software packaged together with the enviroment is called container
- containers can be distributed



Isn’t that just a VM?

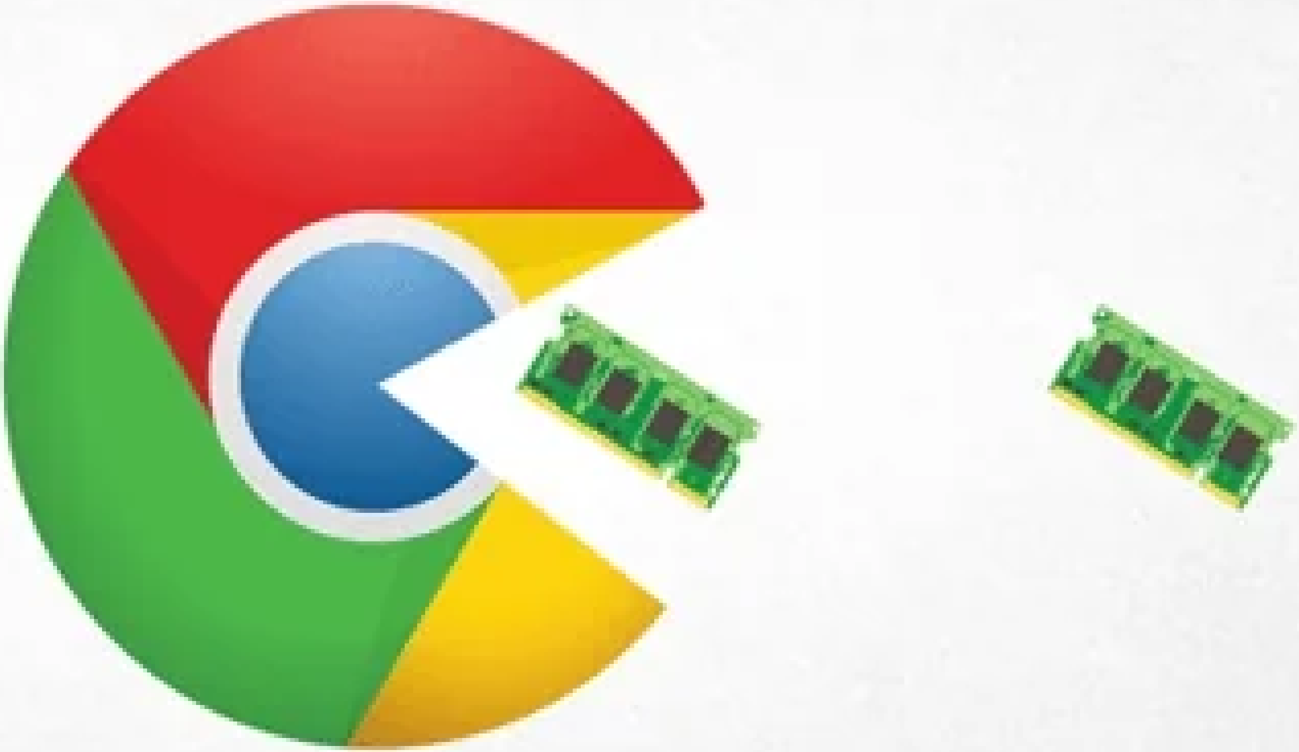
NO

Quick primer on how computers work

Some smart people thought, mhhh why not take sand

And make it into this

Some others thought this would be a good idea



Brief overview of whats happening in a computer

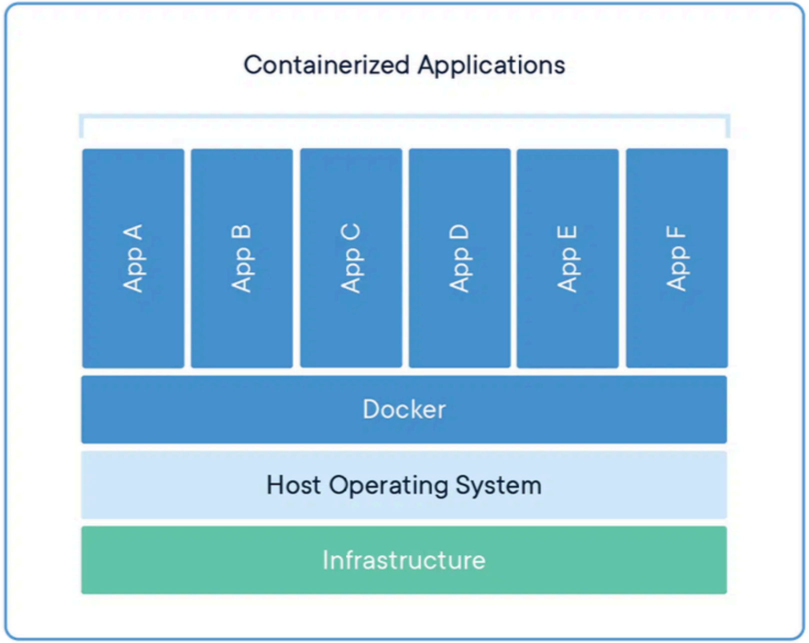
Containers vs VMs

Container

- only files which are needed for the application, sometimes an OS, but without the kernel
- zero overhead for compute, slight overhead for storage
- small size
- not as secure as a VM (don’t run malware inside a docker container)
- work by utilising linux namespaces and cgroups

VM

- contains entire OS, with its own kernel
- big overhead from the entire OS + all its processes
- big size
- very secure, malware breaking out is basically impossible



Containerized Applications

App A

App B

App C

App D

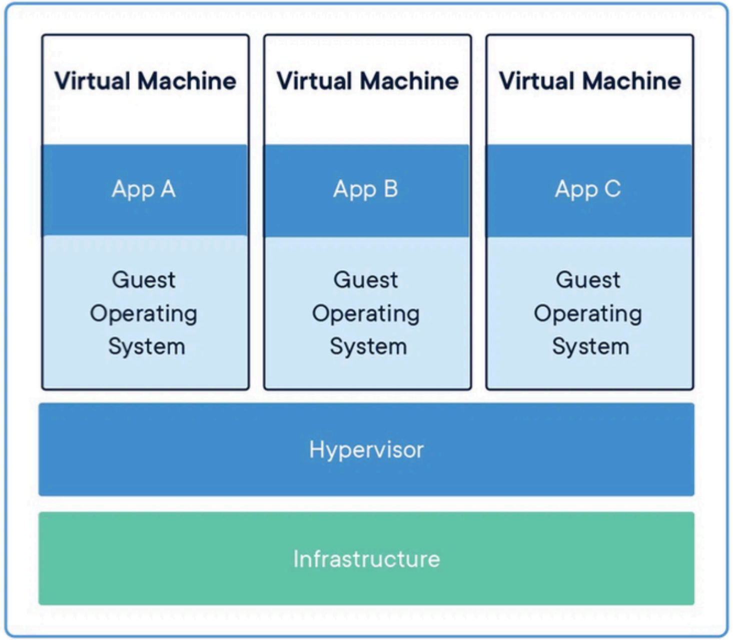
App E

App F

Docker

Host Operating System

Infrastructure



Virtual Machine

App A

Guest Operating System

Virtual Machine

App B

Guest Operating System

Virtual Machine

App C

Guest Operating System

Hypervisor

Infrastructure

Why would you want a container?

IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

Why would you want to use Docker?

- Docker is THE industry standard for deploying software

Standard use cases for Docker:

- Easy + reliable dev setup for a project Instead of having to follow a long readme with loads of steps for getting the project up and running, you simply run a single command, spinning up one (or more) containers
- Easy automated testing setup without loads of steps
- Being able to build projects without polluting your entire OS with globally installed packages, node_modules, etc
- Fully (self)documenting instructions for running a project
- Additional safety layer for deployed applications
- Quickly spinning up a distro without having to actually install it
- Being able to deploy a service to the cloud without vendor lock-in

Problems it solves

- Most programs are reliant on external configurations, software and files

=> software might work at one point in time but not at another

Example scenarios:

- You want to run some piece of software that relies on an old version of a runtime, such as python 2
    - Most distros don’t come with python 2 installed anymore and it’s often not even available in the repos anymore
    - even if you could install it, it might break other parts of your system

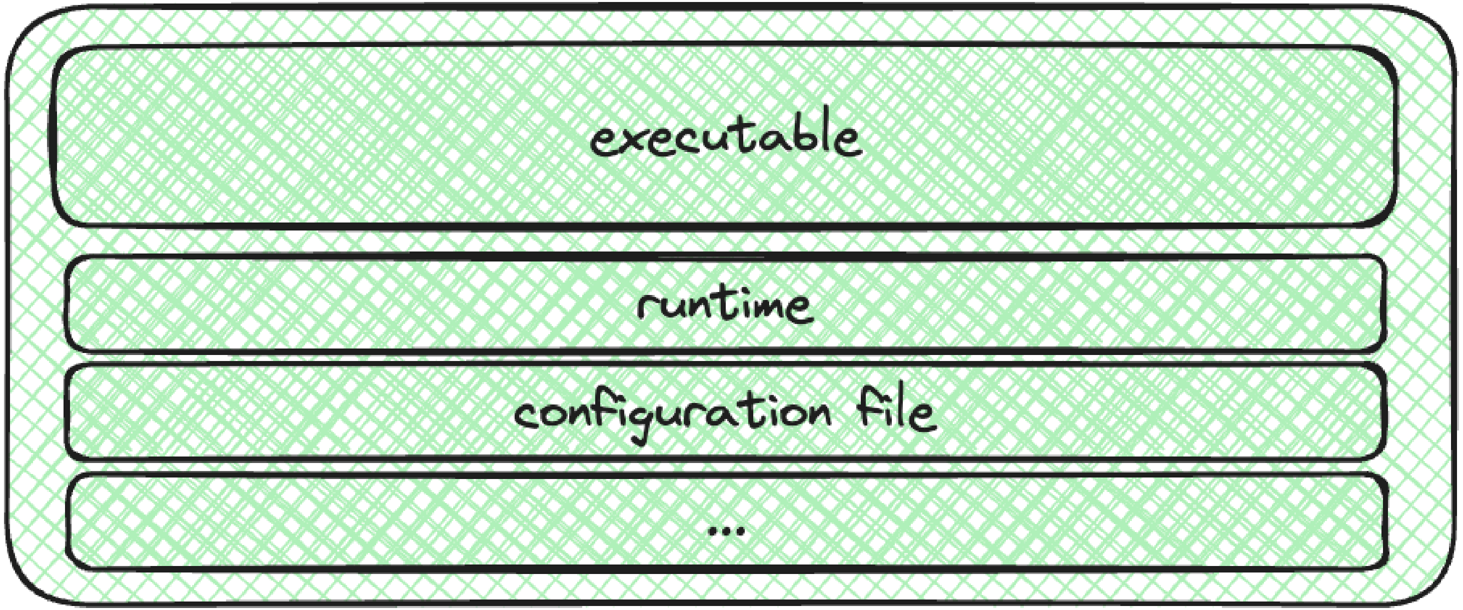
  => **Containers**

- You’re building a complex SaaS application with microservices, which are deployed on both a testing and production enviroment
    - How do you ensure that the software that you tested on the testing enviroment is also gonna run the same in the production environment?

Sounds great, how do I get started?

```
$ docker run hello-world
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Image - just a snapshot



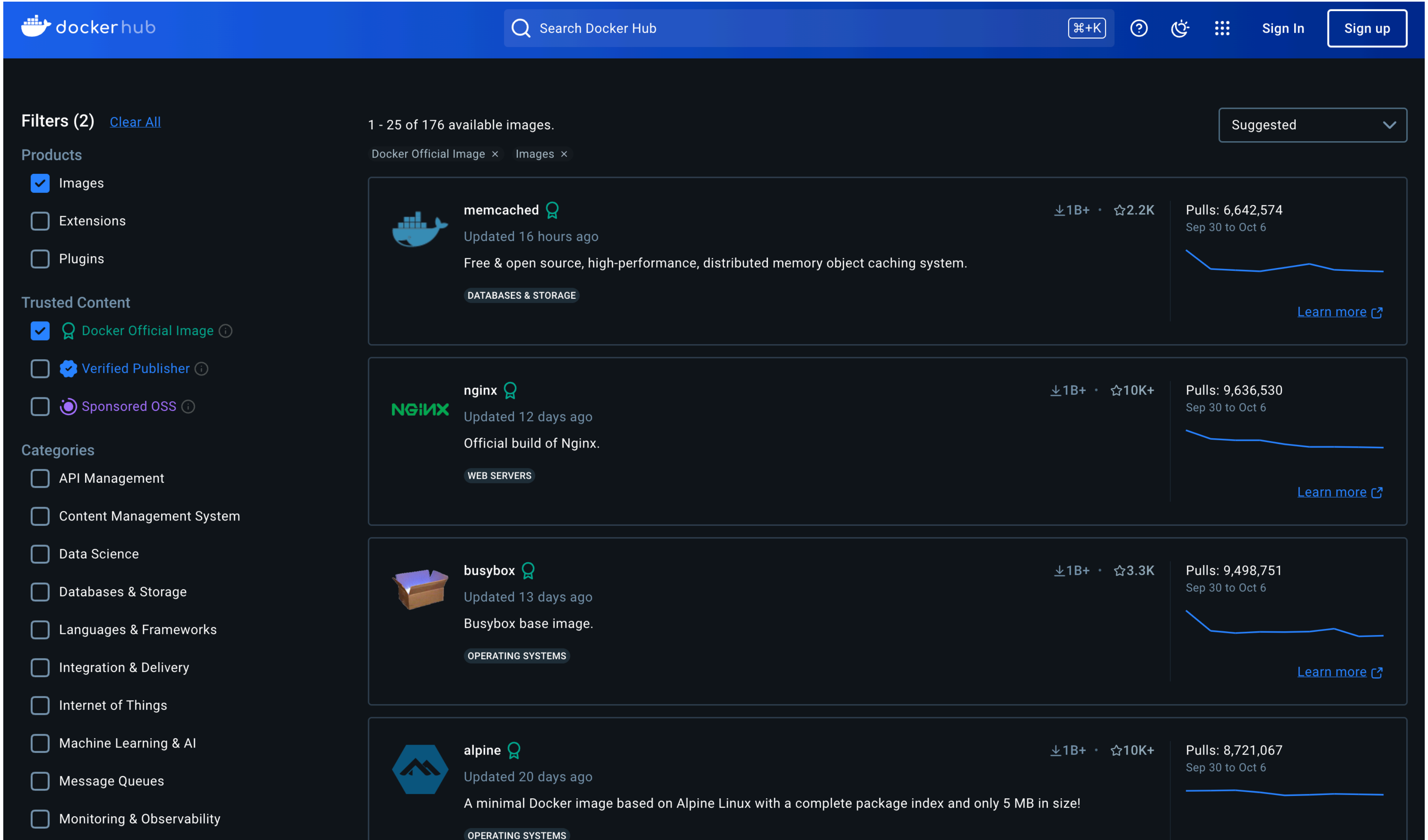
executable

runtime

configuration file

...

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```



docker hub

Search Docker Hub ⌘+K

Sign In

Sign up

Filters (2) Clear All

Products

☑ Images

□ Extensions

□ Plugins

Trusted Content

☑ Docker Official Image ⓘ

□ Verified Publisher ⓘ

□ Sponsored OSS ⓘ

Categories

□ API Management

□ Content Management System

□ Data Science

□ Databases & Storage

□ Languages & Frameworks

□ Integration & Delivery

□ Internet of Things

□ Machine Learning & AI

□ Message Queues

□ Monitoring & Observability

1 - 25 of 176 available images.

Suggested

Docker Official Image × Images ×

memcached

Updated 16 hours ago

Free & open source, high-performance, distributed memory object caching system.

DATABASES & STORAGE

1B+ • ☆2.2K

Pulls: 6,642,574
Sep 30 to Oct 6

Learn more

nginx

Updated 12 days ago

Official build of Nginx.

WEB SERVERS

1B+ • ☆10K+

Pulls: 9,636,530
Sep 30 to Oct 6

Learn more

busybox

Updated 13 days ago

Busybox base image.

OPERATING SYSTEMS

1B+ • ☆3.3K

Pulls: 9,498,751
Sep 30 to Oct 6

Learn more

alpine

Updated 20 days ago

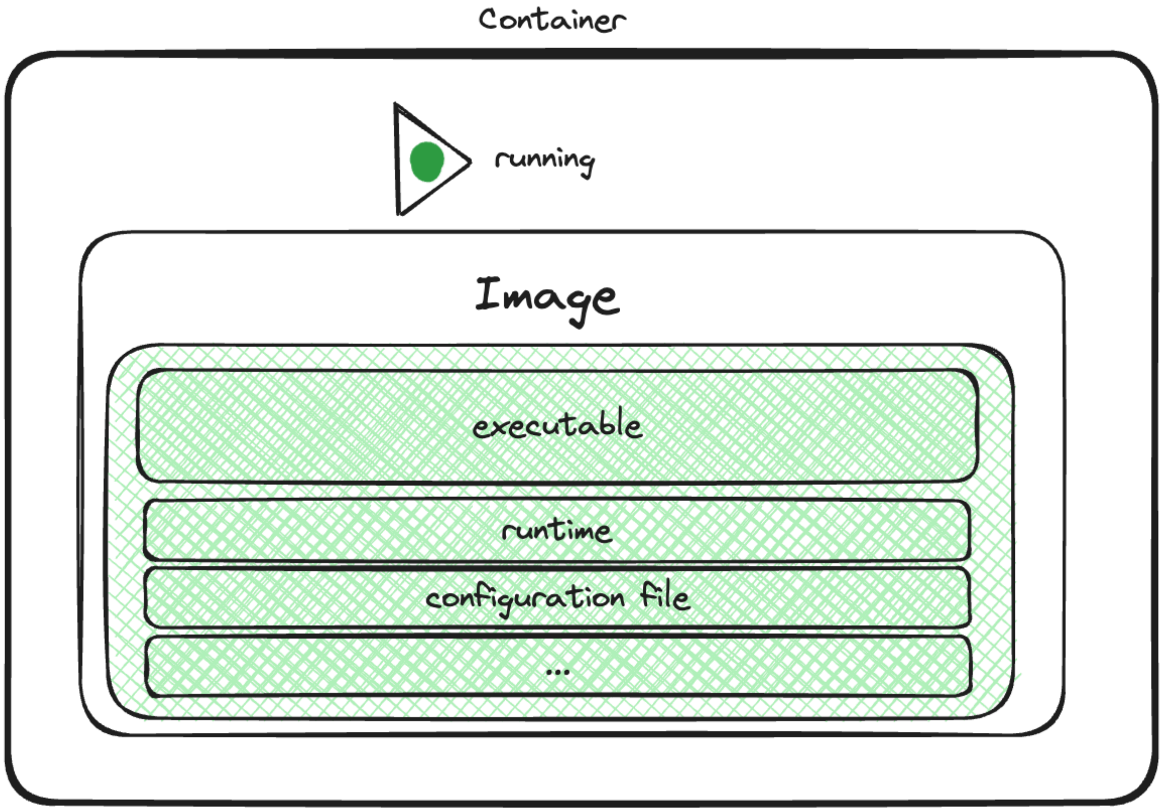
A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

OPERATING SYSTEMS

1B+ • ☆10K+

Pulls: 8,721,067
Sep 30 to Oct 6

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

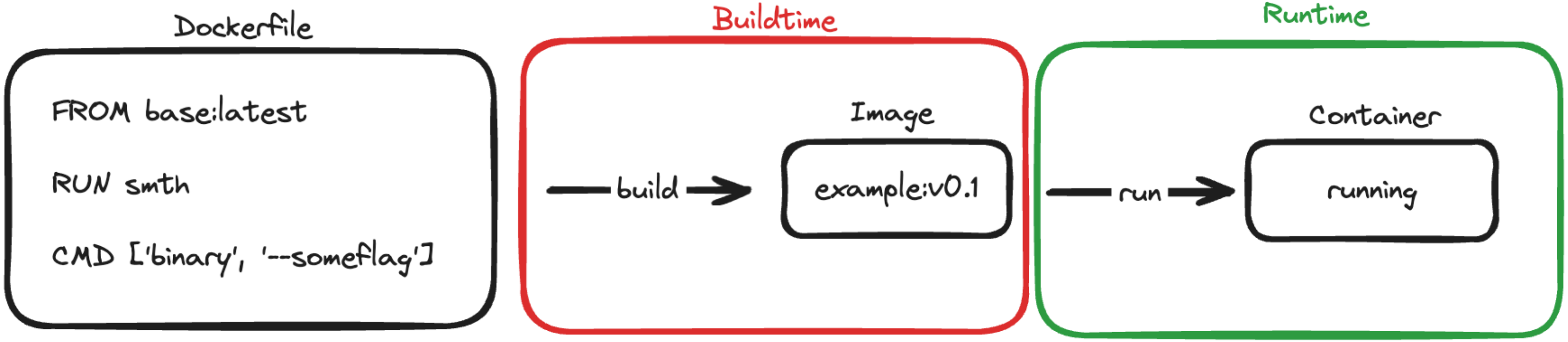

Container
running
Image
executable
runtime
configuration file
...

Basic Dockerfile

Code available at https://github.com/hugohabicht01/dockerintro

`Dockerfile`:

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```



Dockerfile

```
FROM base:latest

RUN smth

CMD ['binary', '--someflag']
```

Buildtime

build

Image

example:v0.1

Runtime

run

Container

running

Basic Dockerfile

Code available at https://github.com/hugohabicht01/dockerintro

`Dockerfile` :

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

`src/index.js` :

```
const Koa = require('koa');
const app = new Koa();

app.use(ctx => {
  console.log(`[*] incoming request`)
  ctx.body = 'Hello from inside the container';
});

app.listen(3000);
console.log('[+] Server listening on port 3000')
```

Demo

Ok, how did that work?

`Dockerfile`:

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

`src/index.js`:

```
const Koa = require('koa');
const app = new Koa();

app.use(ctx => {
  console.log(`[*] incoming request`)
  ctx.body = 'Hello from inside the container';
});

app.listen(3000);
console.log('[+] Server listening on port 3000')
```

```
$ docker build -t intro:v0.1 .
[+] Building 2.2s (9/9) FINISHED                              d
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 156B
 => [internal] load metadata for docker.io/library/node:18-
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/node:18-alpine@sha256:0237
 => [internal] load build context
 => => transferring context: 29.04kB
 => CACHED [2/4] WORKDIR /app
 => [3/4] COPY . .
 => [4/4] RUN yarn install --production
 => exporting to image
 => => exporting layers
 => => writing image sha256:5d652563bf0520e57d4dc24ad33db56
 => => naming to docker.io/library/intro:v0.1
$ docker run -p 3000:3000 --name introcontainer intro:v0.1
[+] Server listening on port 3000
```

Layers

```
$ docker build -t intro:v0.1 .
[+] Building 2.2s (9/9) FINISHED                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 156B                                      0.0s
 => [internal] load metadata for docker.io/library/node:18-alpine         0.9s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [1/4] FROM docker.io/library/node:18-alpine@sha256:02376a266c84acbf4  0.0s
 => [internal] load build context                                         0.0s
 => => transferring context: 29.04kB                                      0.0s
 => CACHED [2/4] WORKDIR /app                                             0.0s
 => [3/4] COPY . .                                                        0.0s
 => [4/4] RUN yarn install --production                                   1.1s
 => exporting to image                                                    0.0s
 => => exporting layers                                                   0.0s
 => => writing image sha256:5d652563bf0520e57d4dc24ad33db56fa6f1a550a6ca  0.0s
 => => naming to docker.io/library/intro:v0.1                             0.0s
```

Layers

continued

Docker works in layers, for each line in the Dockerfile a new layer is created

Important docker commands

`$ curl cheat.sh/docker`

```
# List all docker containers (running and stopped):
docker ps --all
# Start a container from an image, with a custom name:
docker run --name container_name image
# Start or stop an existing container:
docker start|stop container_name
# Pull an image from a docker registry:
docker pull image
# Display the list of already downloaded images:
docker images
# Open a shell inside a running container:
docker exec -it container_name sh
# Remove a stopped container:
docker rm container_name
# Fetch and follow the logs of a container:
docker logs -f container_name
# Quickly spin up a debian system
docker run -it debian:latest /bin/bash
```

Important Dockerfile instructions

```
FROM - base image to start from
RUN - execute any command
COPY - copies new files or directories to the filesystem
ADD - same as COPY, but supports remote (git, tar and plain) urls
ENTRYPOINT - configure command that runs when the container is started
CMD - configures the parameters passed to ENTRYPOINT
WORKDIR - sets working directory for following instructions
ENV - sets environment variables
ARG - defines variable that can be passed during build time
EXPOSE - inform docker that container listens on specified network port
LABEL - adds metadata to image
USER - defines default user and/or group
HEALTHCHECK - tells docker how to test if a container still works
SHELL - overrides shell form used in commands
```

Multi stage docker

```
FROM golang AS builder
WORKDIR /build
COPY . ./
RUN CGO_ENABLED=0 go build \
-ldflags '-extldflags "-static"' -o main main.go

FROM scratch
COPY --from=builder /build/main /main
ENTRYPOINT ["/main"]
```

```
package main

import "fmt"

func main() {
    fmt.Println("Hello world from inside the container")
}
```

```
$ docker build -t goexample:v0.1 .
[+] Building 3.1s (10/10) FINISHED                                 d
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 236B
 => [internal] load metadata for docker.io/library/golang:l
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [builder 1/4] FROM docker.io/library/golang:latest@sha2
 => [internal] load build context
 => => transferring context: 2.97kB
 => CACHED [builder 2/4] WORKDIR /build
 => [builder 3/4] COPY . ./
 => [builder 4/4] RUN CGO_ENABLED=0 go build -ldflags '-ext
 => CACHED [stage-1 1/1] COPY --from=builder /build/main /m
 => exporting to image
 => => exporting layers
 => => writing image sha256:21af68b0874d5ea6f51436a05c5c075
 => => naming to docker.io/library/goexample:v0.1
$ docker run goexample:v0.1
Hello, World!
```

Output size comparison

Raw binary:

```
$ ls -la main
.rwxr-xr-x 2.0M cedric 14 Oct 18:38 main
```

Docker image:

```
$ docker images goexample:v0.1
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
goexample           v0.1      21af68b0874d   2 minutes ago   2.15MB
```

Tasks for you

1. Build a simple hello world application in the language of your choice, build a docker image for that and run it
2. Try to deploy a simple website with just an index.html inside a docker container using caddy as the webserver the `Caddyfile` and the `index.html` can be found in the dockerintro repo
3. Create a new version (v0.2) for the last task, with some changes to the index.html

`Hacker tasks`:

1. Create a multi-stage Dockerfile that compiles + runs any application of your choice in a language of your choice
2. Minimize the size of the resulting docker image as much as possible
3. Create a simple microservice setup:

- an API, that responds with the current time, in one container
- another container that fetches that API and then displays the result, either on a webpage or in the console
- feel free to add more services and a DB

How can you try out Docker?

Many options available

Either

- online at https://labs.play-with-docker.com
- by installing Docker or any other OCI compatible runtime locally, such as `podman`

If you’re on a GNU/Linux system, you can

- run `docker` or `podman` natively

if you’re on MacOS or Windows, you’ll need some software that spins up a Linux VM, such as

- Orbstack (Mac)
- Rancher Desktop (Mac and Windows)
- Docker Desktop (Mac and Windows) (not really recommended…)

Cool links to learn more about containers:

- Building a container runtime from scratch
- Interesting usecases for docker
- Deep dive into the building blocks of docker
- Intro to Linux namespaces
- Docker Compose
- Kubernetes Tutorials
- Great book about k8s
- Great book about docker

Presentation made with sli.dev

Documentation · GitHub · Showcases

Powered by Slidev