Why Docker?

Image generated with DALLE 3

A long time ago a developer far, far away developed an application

It ran perfectly on their machine, and all was well

Image generated with DALLE 3

Until it was time for deployment

Image generated with DALLE 3

So, a virtual machine was in order

The thing is, virtual machines have many down sides

Size

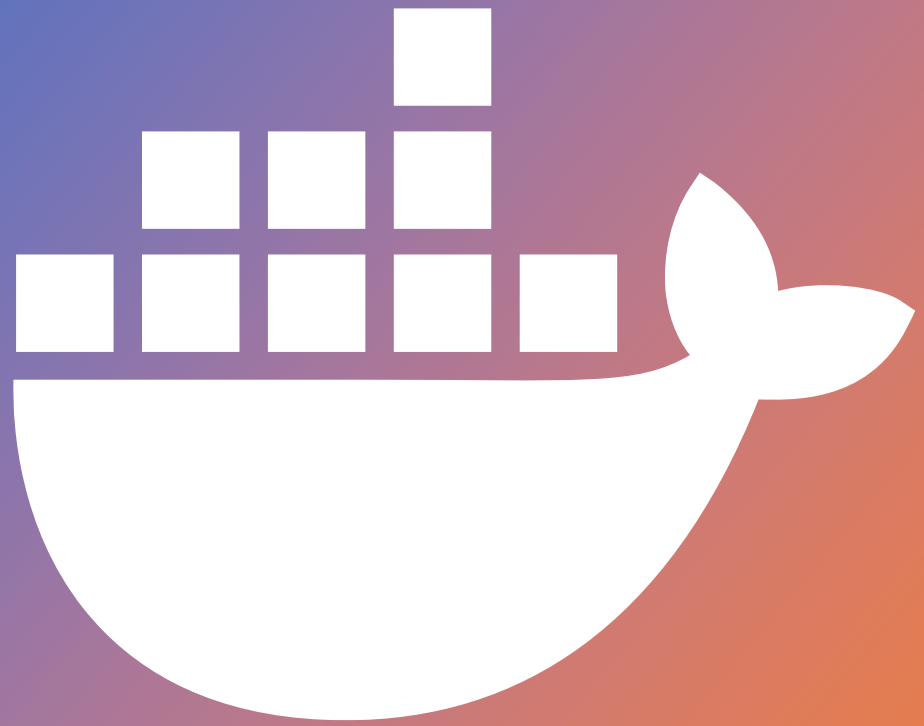
Often, VM images are in the 10s of Gigabytes

Compute Resources

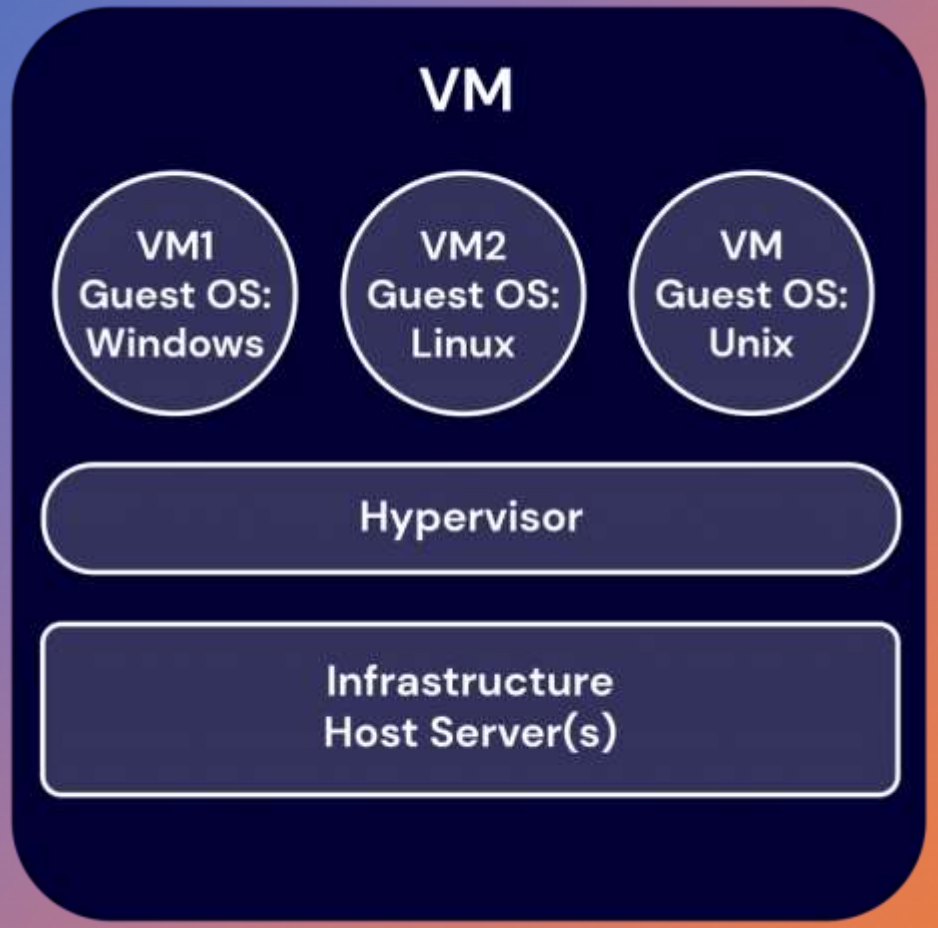
Every VM comes with its own OS, requiring significantly higher memory and CPU time to host the application

Image generated with DALLE 3

So, what can we do?



Enter, a friendly whale

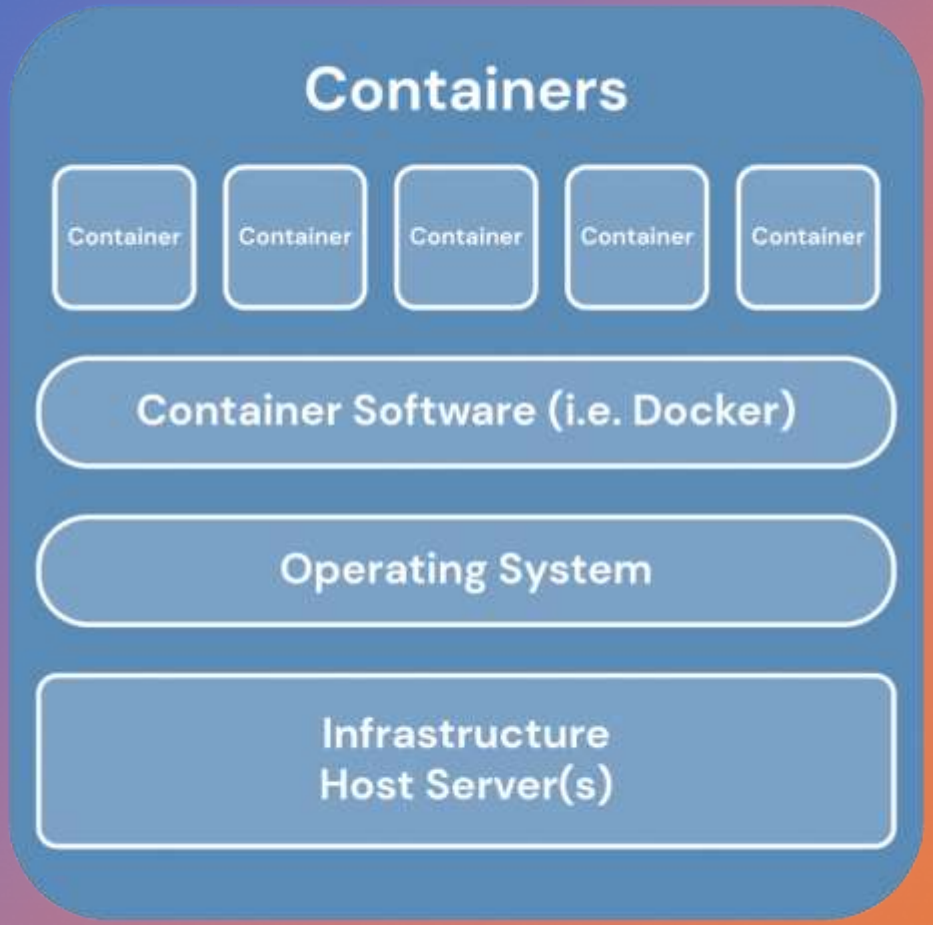


Img src: https://www.backblaze.com/blog/vm-vs-containers/

Docker is built different

VMs require:

- A host server
- Hypervisor/Host OS
- Guest OS
- Dependencies
- Application



Img src: https://www.backblaze.com/blog/vm-vs-containers/

Docker is built different

Docker Requires

- Host server
- Host OS
- Container software
- Container with application and dependencies

Resources

Docker containers share the host OS's kernel, meaning they use much less resources than a VM

Resources

Docker containers do not require dedicated memory allocation like VMs

Resources

Docker containers lack an entire OS, so they are much smaller in size

Resources

Windows VM: 30+ GB

Resources

Windows container: 
~300MB

*nanoserver

Resources

Debian container:
74.8MB

*bookworm-slim

Resources

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
portainer/portainer-ce	latest	a3f85c245ec3	27 hours ago	293MB
lscr.io/linuxserver/calibre	latest	78cdfd7753e5	3 days ago	2.79GB
lscr.io/linuxserver/tautulli	latest	a798bcc66398	4 days ago	188MB
nginx	alpine	11d76b979f02	6 days ago	48.3MB
mongo	latest	3902e4c1fa5c	2 weeks ago	758MB
alpine	latest	05455a08881e	2 months ago	7.38MB
jaymoulin/jdownloader	latest	4c2bc32eff85	7 months ago	182MB
dnsforge/xteve	latest	27c7553946cf	14 months ago	644MB
akhilrex/podgrab	latest	6e0094ece2a4	19 months ago	32.4MB

Building a container

Image generated with DALLE 3

A Basic Dockerfile

```
FROM node:18-alpine

WORKDIR /usr/src/app

COPY . .

RUN npm install

EXPOSE 3000

CMD [ "node", "app.js" ]
```

The `FROM` instruction

The `FROM` instruction initializes a new build stage with a base image for subsequent instructions

The `WORKDIR` instruction

The `WORKDIR` instruction sets the working directory for subsequent instructions (like `cd`)

The `COPY` instruction

The `COPY` instruction copies files into the build stage

The `RUN` instruction

The `RUN` instruction executes commands, *creating a new layer in the image*

The `EXPOSE` instruction

The `EXPOSE` instruction specifies what port(s) the container listens on

CMD/ ENTRYPOINT

These instructions specify how your application should be started in the container

More Instructions

More instructions are available and are well documented in the Dockerfile reference

docker build

Most of the time, the only command you will need to build containers is

```
docker build -t
<image_name>:<tag> .
```

Run the container

We could run the container like this:

```
docker run -d -p 3000:3000 \
       --restart=unless-stopped \
       --name my_container \
       my_image
```

Run the container

Or we could configure our container(s) as code with Docker Compose

Docker Compose

Docker Compose allows you to:

- Define container configurations as code
- Define application stacks to orchestrate related containers
- Use secrets specified outside of the compose file

+

CI/CD

CI/CD

Pretty much all relevant cloud platforms support the deployment of Docker containers as scalable infrastructure.

Here is an example of this concept in the wild.

Example time

The example repo can be found here