Introduction to Docker

Edmund Fokschaner
Software Engineer, Riot Games
edfokschaner@gmail.com

Agenda

Origins

Architecture

Ecosystem

Usage

Questions (but feel free to interrupt anytime)

Where did Docker come from?

Docker began development around 2010 inside dotCloud, a “Platform as a Service” company.

Hosting applications rather than hosting virtual machines.

Many existing language-specific application formats.

An easy way to package arbitrary software and data to be run on dotCloud servers without the administrative or computational overhead of Virtual Machines.

How did dotCloud address this with Docker?

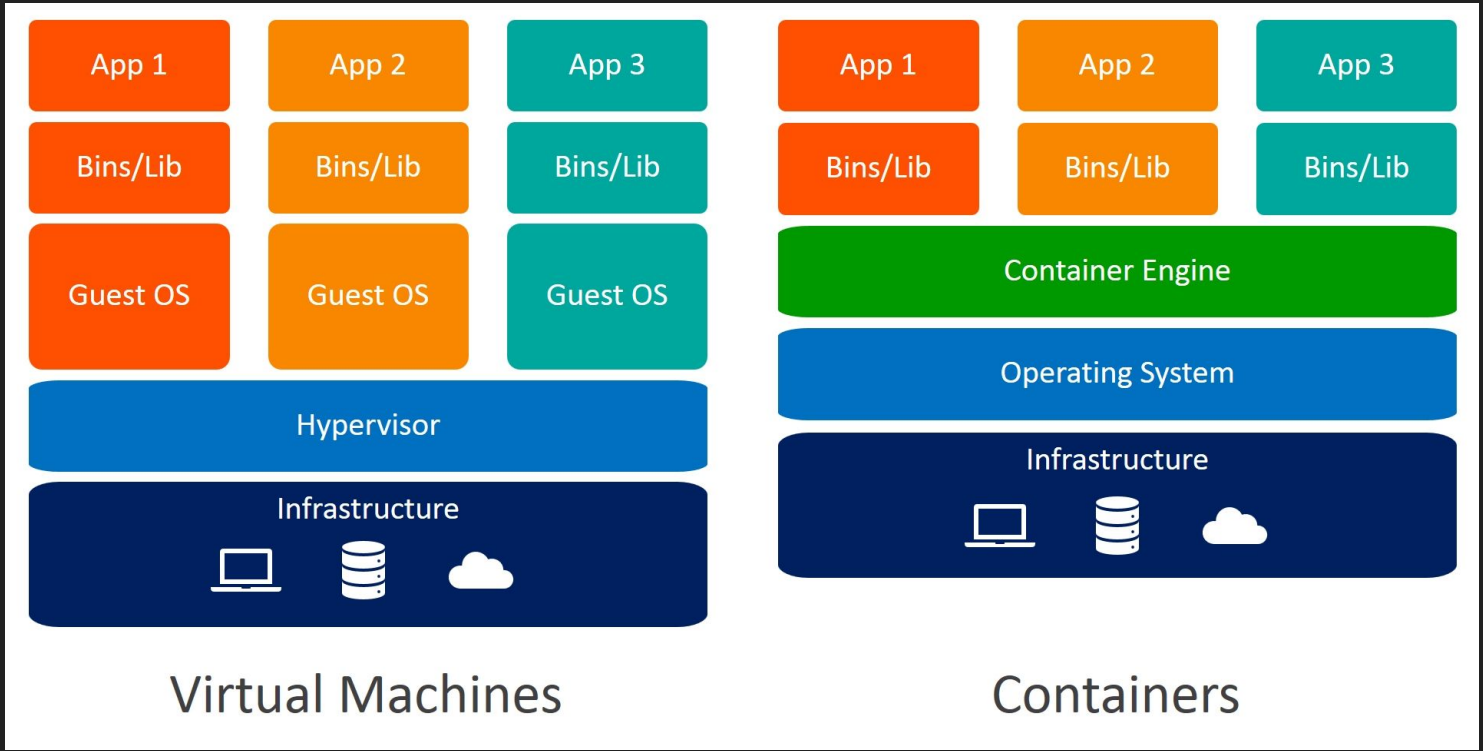
Built on “Linux Containers” (LXC) (est. 2008) which used Linux “namespaces” to:

- Provide logical isolation of:
  - Process IDs
  - Filesystem
  - Network ports
  - User IDs
- Control utilization of:
  - CPU time
  - Virtual memory usage
  - Network iops
  - Disk iops

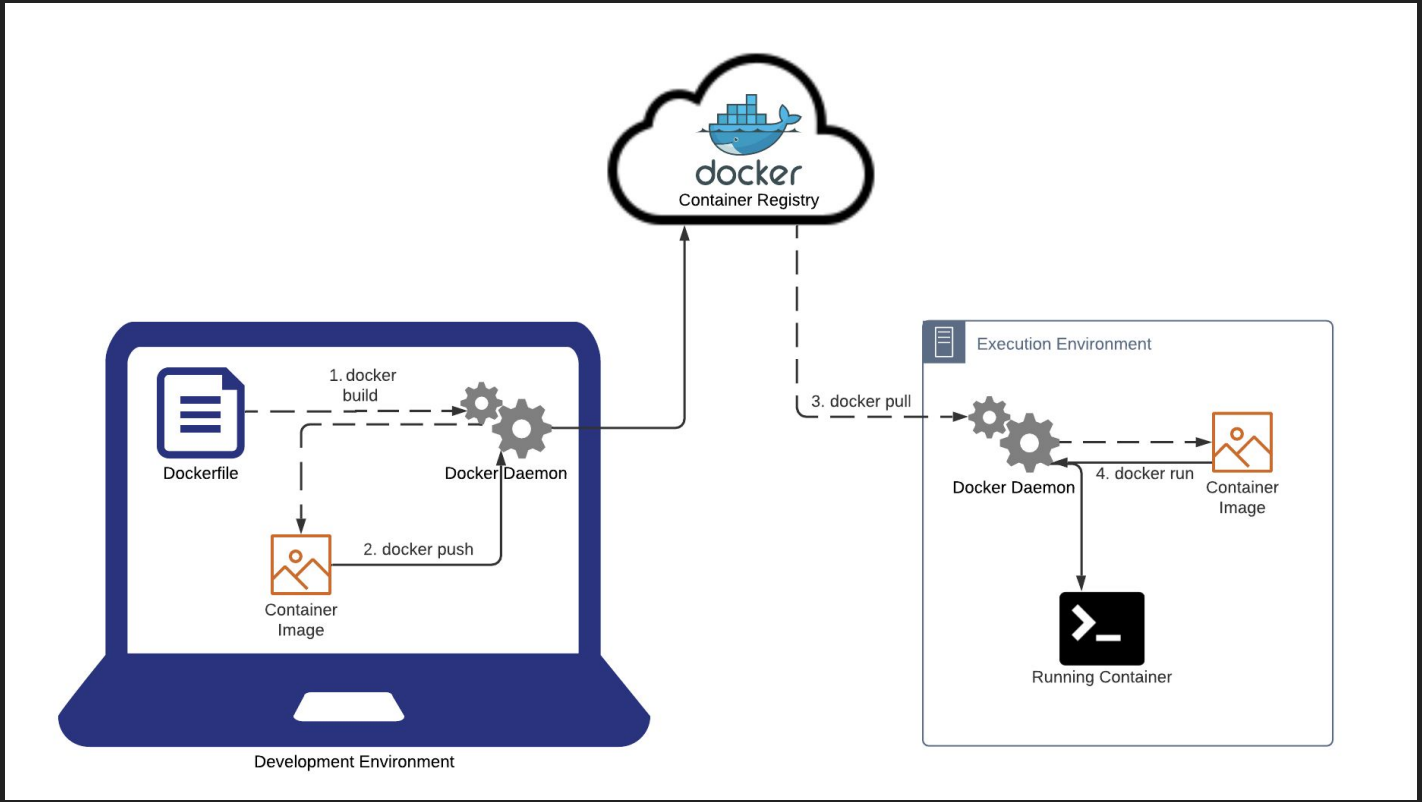
Docker tooling solved the challenge of:

- Image creation: Defining what software and data should be packaged together as an “image”.
- Image distribution: Efficiently moving the image from where it was created to where it needs to be run.
- Image execution: Invoking LXC appropriately to launch a running instance of the container image.

Linux Containers vs Virtual Machines



The Docker Tooling



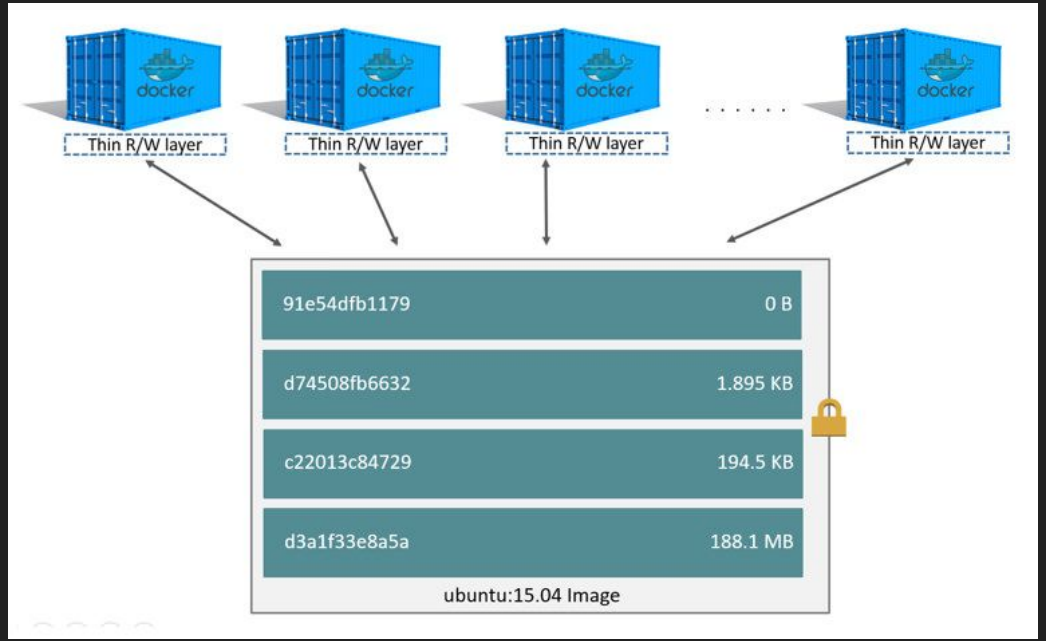
The Layered Container Format

Layers are deltas to the filesystem + metadata that informs execution and which layer it builds on top of.

The bottom layer is usually a fresh linux distribution eg. Ubuntu

Each layer has a checksum-based ID

When a container is run, it gets another read-write layer and the storage driver presents a unioned view of the layer hierarchy.



The Layered Container Format

This layer architecture allows for coarse deduplication / caching at many stages in the Docker lifecycle.

build: Only re-build layers which have changes.

push: Only upload layers which are new.

pull: Only download layers which you haven’t downloaded before.

run: All unmodified files are shared across all container instances.

10 years on

Docker launched in 2013 and it caught on.

Because it was successful and encouraged an ecosystem of intercompatibility, three big things started to happen.

- Other cloud providers started offering docker as a service.
- Companies adopted it as a better way to delineate between infrastructure engineering vs. product engineering.
- Standardisation and alternative implementations

LXC was replaced.

Standardization of container image format and runtime, via the vendor-neutral “Open Container Initiative”.

Extraction of official implementations of libcontainer, runc, and containerd.

Alternative implementations

- Container Registry - Docker Hub (https://hub.docker.com/), AWS Elastic Container Registry, Google Cloud Container Registry, jFrog Artifactory
- Container Engine - Docker Engine, Podman, LXD
- Container Build - Kaniko, Buildah, Buildkit (an alternative to docker build made by Docker)
- Cluster Management - Docker Swarm, Kubernetes, Nomad

Using Docker

First you'll need a Docker toolchain, I recommend Docker Desktop - https://www.docker.com/products/docker-desktop

Confirm it's working with the official "Hello World" container:

```
> docker run hello-world

Hello from Docker!

This message shows that your installation appears to be
working correctly.
```

analyze.sh

We make file called `analyze.sh` with the following contents

```
#!/bin/bash

samtools --help
```

Dockerfile

We make a file, called “Dockerfile” with the following contents.

```
FROM ubuntu:20.04

# Update the apt listings because this distro install is effectively "brand new"
RUN apt-get update

# Install external dependencies for samtools
RUN apt-get install -y autoconf automake make gcc perl zlib1g-dev libbz2-dev liblzma-dev libcurl4-gnutls-dev libssl-dev libncurses5-dev

# Install curl, used to fetch samtools source
RUN apt-get install -y curl

# Download the source tarball for samtools (with --location to follow the redirects)
RUN curl "https://github.com/samtools/samtools/releases/download/1.14/samtools-1.14.tar.bz2" -o /opt/samtools-1.14.tar.bz2 --location

# Extract the source tarball for samtools
RUN tar --extract --file /opt/samtools-1.14.tar.bz2 --directory /opt

# Compile the source for samtools
RUN cd /opt/samtools-1.14 && ./configure && make install

# Copy our analyze.sh in to the container
COPY analyze.sh /opt/analyze.sh

# Specify our analyze.sh as the primary command to run
CMD ["/bin/bash", "/opt/analyze.sh"]
```

docker build

```
> docker build -f Dockerfile -t efokschaner/samtools:1.0.0 .

 => CACHED [1/8] FROM docker.io/library/ubuntu:20.04@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322
0.0s
 => [2/8] RUN apt-get update
8.1s
 => [3/8] RUN apt-get install -y autoconf automake make gcc perl zlib1g-dev libbz2-dev liblzma-dev libcurl4-gnutls-dev libssl-dev
libncurses5-dev                                                                                                    28.7s
 => [4/8] RUN apt-get install -y curl
4.5s
 => [5/8] RUN curl "https://github.com/samtools/samtools/releases/download/1.14/samtools-1.14.tar.bz2" -o /opt/samtools-1.14.tar.bz2
--location                                                                                                          2.2s
 => [6/8] RUN tar --extract --file /opt/samtools-1.14.tar.bz2 --directory /opt
3.0s
 => [7/8] RUN cd /opt/samtools-1.14 && ./configure && make install
108.8s
 => [8/8] COPY analyze.sh /opt/analyze.sh
 => exporting to image
 => => exporting layers
4.0s
 => => writing image sha256:9e15a0894936ee40402c4638caecad7db0d8441de7773130592c18b31a636f65
0.0s
 => => naming to docker.io/efokschaner/samtools:1.0.0
```

Image Names and Tags

`-t efokschaner/samtools:1.0.0` created `docker.io/efokschaner/samtools:1.0.0`

No domain in name => docker.io

No tag => “latest”

Prefer explicit tags if you can.

Distribution

```
> docker tag efokschaner/samtools:1.0.0
our-shared-registry.com/efokschaner/samtools:1.0.0

> docker push our-shared-registry.com/efokschaner/samtools:1.0.0
```

On the receiving side:

```
> docker run our-shared-registry.com/efokschaner/samtools:1.0.0

Program: samtools (Tools for alignments in the SAM format)

Version: 1.14 (using htslib 1.14)
```

“pull” is implied.

For an interactive shell:

```
> docker run -it efokschaner/samtools:1.0.0 /bin/bash
```

Remarks

Author and recipient now running the exact same copy of samtools, with the exact same surrounding files and folders.

As nice as the samtools INSTALL instructions are, the recipient did not need to follow them.

The only thing the author and the recipient need to have in common is a version of the Linux kernel compatible with all the software inside the container. Not necessarily identical kernel versions.

Dockerfile forces us to explain how to create the environment in which our code runs.

The container image is the immutable program, not the Dockerfile.

Changing a layer rebuilds all subsequent layers, when possible put the steps which change most frequently later in the Dockerfile.

Best Practices

Minimize Build Context using .Dockerignore

```
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 44B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04
```

Ignore everything with `*` and allow-list items with `!`

```
*
!analyze.sh
```

Best Practices

Use multi-stage builds to separate build-dependencies from runtime-dependencies.

Clean the apt package manager cache after installing packages.

Take 2

```
FROM ubuntu:20.04 AS builder

RUN apt-get update && apt-get install -y \
autoconf automake curl make gcc perl zlib1g-dev libbz2-dev liblzma-dev libcurl4-gnutls-devlibssl-dev libncurses5-dev \
&& rm -rf /var/lib/apt/lists/*

RUN curl "https://github.com/samtools/samtools/releases/download/1.14/samtools-1.14.tar.b≠ô"/opt/samtools-1.14.tar.bz2--location
RUN tar --extract --file /opt/samtools-1.14.tar.bz2--directory /opt
RUN cd /opt/samtools-1.14&& ./configure --prefix=/opt/samtools-install&& make install

FROM ubuntu:20.04

# install the non-development versions of runtime dependencies
# Confirmed using https://packages.debian.org/search?keywords=search and trial and error
RUN apt-get update && apt-get install -y \
perl zlib1g libbz2-1.0 liblzma5 libcurl4-gnutls-devlibssl1.1 libncurses5 \
&& rm -rf /var/lib/apt/lists/*

COPY --from=builder /opt/samtools-install/opt/samtools-install

ENV PATH /opt/samtools-install/bin:$PATH

COPY analyze.sh /opt/analyze.sh

CMD ["/bin/bash", "/opt/analyze.sh"]
```

Size Comparison

```
> docker inspect -f "{{ .Size }}" efokschaner/samtools:1.0.0

398583228

> docker inspect -f "{{ .Size }}"
efokschaner/samtools-optimized:1.0.0

145691766
```

400MB reduced to 150MB!

Input/Output

Generally docker is used for software which chats over the network and also chats to a specialised database service of some sort. For scientific data crunching, what's the

Getting data in:

- docker build (`COPY`) (smaller data, <1GB)
- mounts
- network storage (S3 / SMB / NFS)

Getting data out:

- stdout
- `docker cp`
- mounts
- network storage (S3 / SMB / NFS)

Remember to version data for maximum reproducibility. (eg. checksum or metadata with version number)

Other Notes

Debugging containers

- `docker exec ...`
- sshd

GPUs `docker run --gpus ...`

Orphaned child processes and pid 1 `docker --init`

Network inception, don’t get lost.

When to use docker?

Applications vs tools vs toolkits

Great for “applications”

Great for a toolkit of tools (“All the tools you’ll need working on X”)

Not so great for individual tools that are intended to be composed with others on the command line. docker-in-docker, or docker-from-docker, creates new challenges in deployment.

Can I safely run untrusted code?

Security is relative and depends on the threat-model. Consult an expert.

Early days, there were many sandbox escapes.

These days it’s pretty secure.

Depends on the correctness of:

- Your docker configuration / additional hardening.
- Your linux kernel
- Hypervisor (if virtualizing too)
- CPU

In contrast to VM’s where security depends on the correctness of just the hypervisor and CPU.

Questions?

References

History of Containers - https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

LXC - https://linuxcontainers.org/

Docker Storage Drivers / Image Format - https://docs.docker.com/storage/storagedriver/

Open Container Initiative - https://opencontainers.org/

Docker Command Line reference - https://docs.docker.com/engine/reference/commandline/cli/

Dockerfile reference - https://docs.docker.com/engine/reference/builder/

“An introduction to Docker for reproducible research, with

examples from the R environment” - https://arxiv.org/abs/1410.0846