The Problem

Different application stacks

Different hardware deployment environments

How to run all applications across different environments?

How to easily migrate from one environment to another?

	Dev VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web fronted	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

The Solution

Unit of software delivery

Lightweight, portable, consistent

Deploy and run everywhere

Deploy and run anything

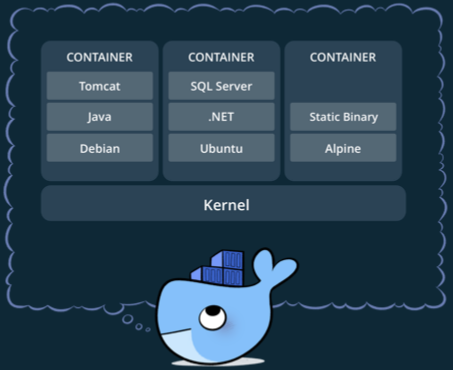
	Dev VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor’s laptop	Customer Servers
Static website	🐳	🐳	🐳	🐳	🐳	🐳	🐳
Web fronted	🐳	🐳	🐳	🐳	🐳	🐳	🐳
Background workers	🐳	🐳	🐳	🐳	🐳	🐳	🐳
User DB	🐳	🐳	🐳	🐳	🐳	🐳	🐳
Analytics DB	🐳	🐳	🐳	🐳	🐳	🐳	🐳
Queue	🐳	🐳	🐳	🐳	🐳	🐳	🐳

What is a container?

- A lightweight, stand-alone, executable package of software that includes all dependencies: code, runtime, system tools, system libraries, settings.
- Containers isolate software from its surroundings
  - development and staging environments
  - help reduce conflicts between teams running different software on the same infrastructure.
- Long history: chroot, FreeBSD Jails, Solaris Containers, OpenVZ, LXC
- Docker simplified creation/management/operation of containers

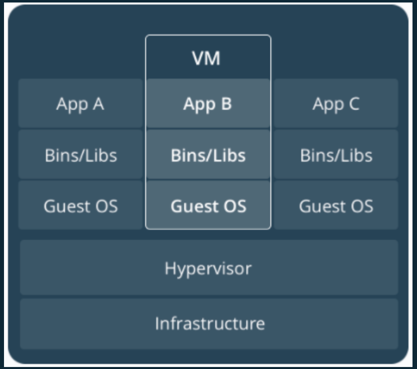
What is a container?

- Containers share a machine's OS kernel.
- They start instantly[1] and use less compute and RAM.
- Images are constructed from filesystem layers and share common files. This minimizes disk usage and image downloads are much faster.



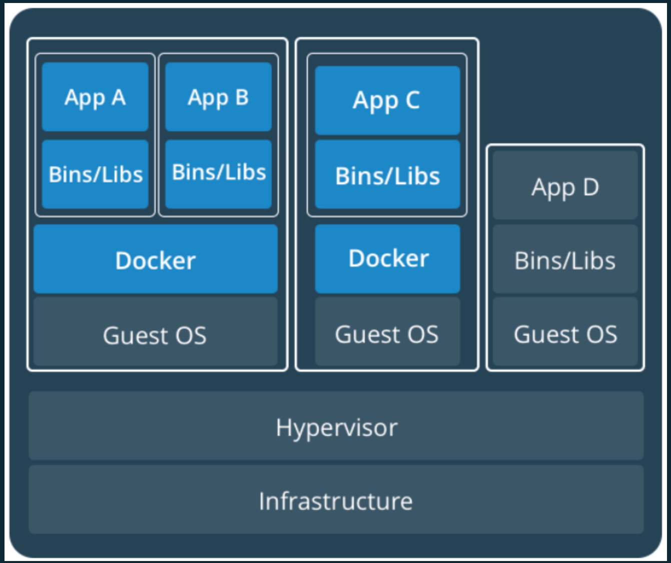
[1]Windows containers can take minutes to start and have other differences from Linux containers.

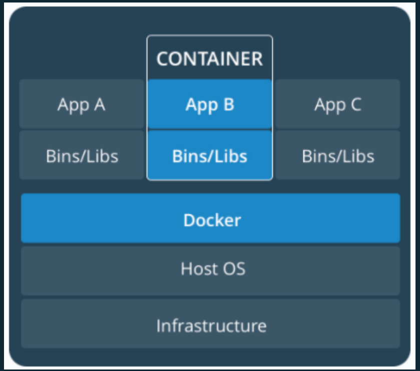
Virtual Machines vs Containers



VMs

Containers on VMs





Containers

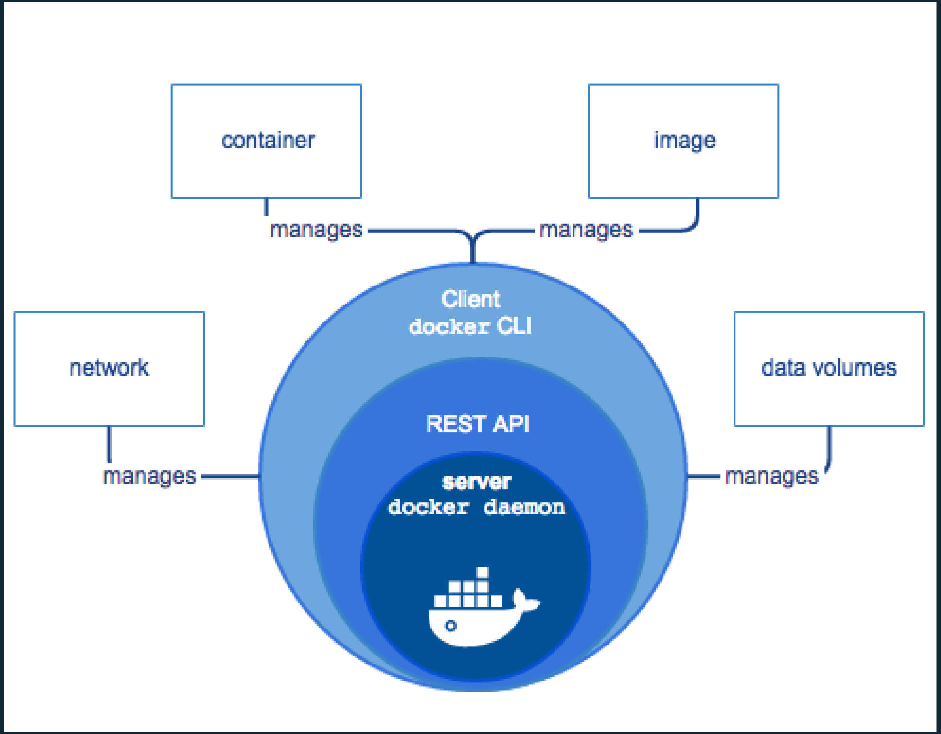
Benefits

- Portable runtime application environment
- Package application and dependencies in a single artifact
- Run different application versions (different dependencies) simultaneously
- Faster development & deployment cycles
- Better resource utilization

Use Cases

- Consistent environment between Development & Production
- Service-Oriented Architectures / Microservices
- Short lived workflows
- Isolated environments for testing

Docker Engine



Container registries

- A Docker registry stores Docker images
- Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  - https://hub.docker.com/
- Amazon EC2 Container Registry (Amazon ECR)
  - Fully managed Docker container registry
  - Makes it easy for developers to store, manage, and deploy container images

Container registries

- **docker pull** or **docker run**: required images are pulled from your configured registry.

- **docker push**: your image is pushed to your configured registry.

Docker Image

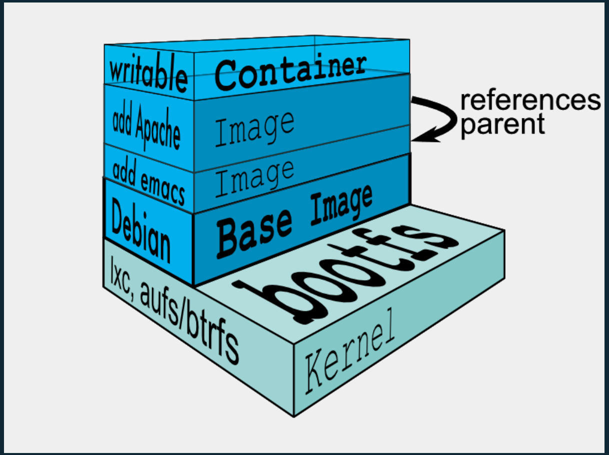
- A read-only template with instructions for creating a Docker container.
- Often, an image is *based on* another image, with some additional customization.
  - E.g, you may build an image which is based on the Ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
- Create your own images.
- Use images created by others and published in a registry.

Docker Image

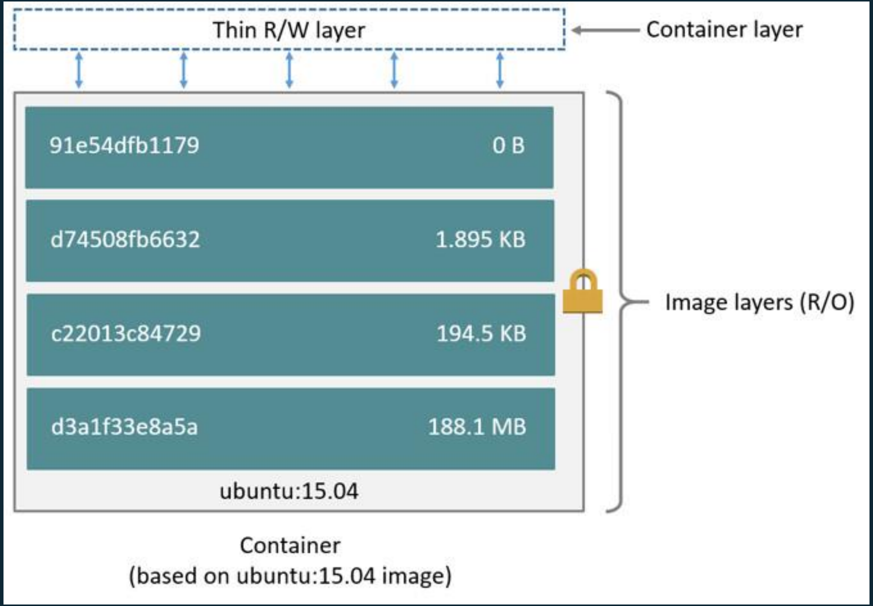
- To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it.
    - Each instruction in a Dockerfile creates a layer in the image.
    - When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt.
    - This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.
- Creating Effective Images, video, slides

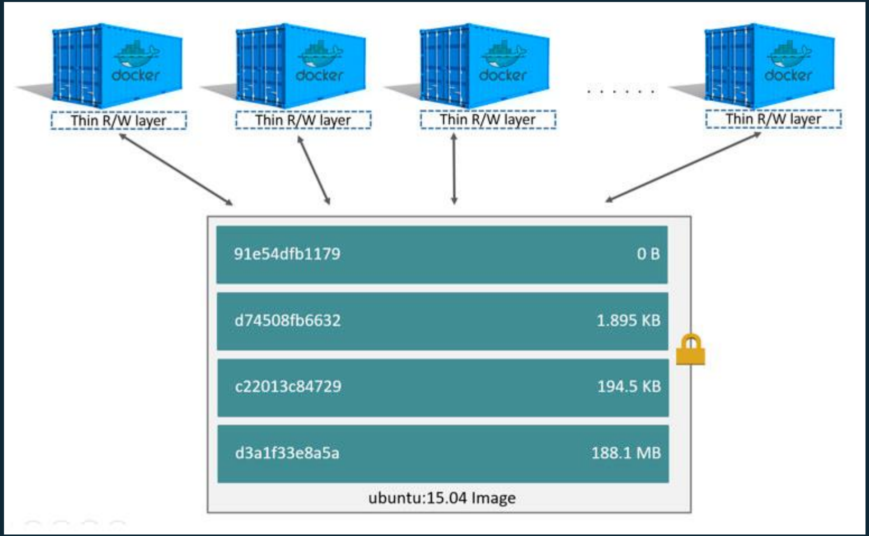
Docker Image Layers

- Union file system creates layers
  - aufs, overlay2, zfs, etc
- Image composed on read-only (RO) layers
- Containers get a thin read-write layer
- Container layer removed when container is deleted
  - Store persistent data on a separate data volume
- Copy-on-Write strategy used when modifying files from RO lower layers



Docker Image Layers





Docker Containers

- A runnable instance of an image.
- Create, run, stop, move, or delete a container using the Docker API or CLI.
- Connect a container to one or more networks, attach storage to it, create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine.

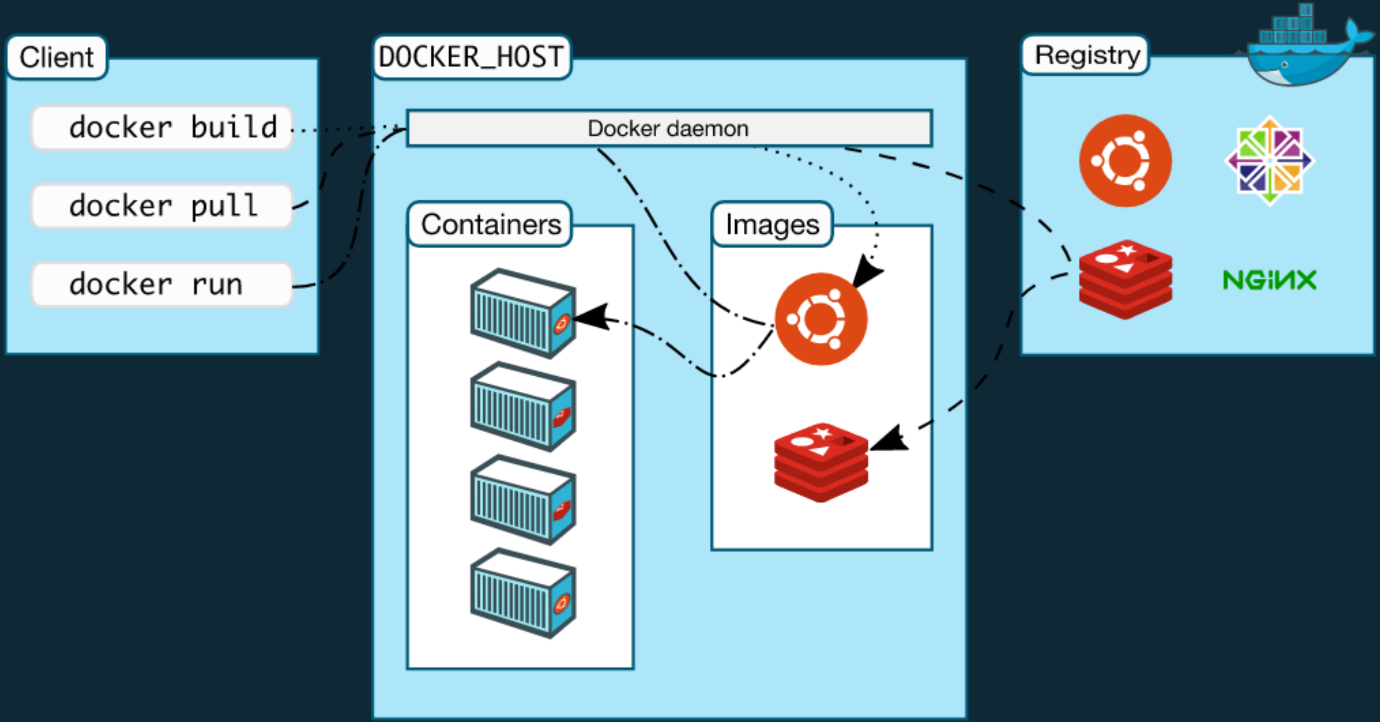
Docker Containers

- You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

- A container is defined by its image as well as any configuration options you provide to it when you create or run it.

- When a container stops, any changes to its state that are not stored in persistent storage disappears.

Docker Architecture



Docker Workflow

- Find an Image on registry (Docker Hub, ECR, etc.)
- Pull an Image from registry
- Run an Image
- Stop a Container
- Modify & rebuild Image
- Rerun Image
- Push Image
- Remove a Container
- Remove an Image

Docker CLI

• docker build	# Build an image from a Dockerfile
• docker info	# Display system-wide information
• docker images	# List all images on a Docker host
• docker run	# Run an image
• docker ps	# List all running and stopped instances
• docker stop	# Stop a running instances
• docker rm	# Remove an instance
• docker rmi	# Remove an image
• docker pull	# Download an image from registry
• docker push	# Upload an image to the registry

Images are Built from a Dockerfile

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y redis-server
EXPOSE 6379
ENTRYPOINT ["/usr/bin/redis-server"]
```

Dockerfile Reference

FROM <image>:<tag>
Sets the base image. Must be first instruction in Dockerfile.

RUN <command>
Executes any commands in a new layer on top of the current image and commit the results.

CMD [“exec”, “param1”, “param2”]
Sets the command to be executed when running the image.

ENTRYPOINT [“exec”, “param1”, “param2”]
Configures a container that will run as an executable.

Dockerfile Reference

EXPOSE <port>
Informs Docker that the container listens on the specified network ports at runtime.

ENV <key> <value>
Sets the environment variable <key> to the value <value>.

ADD <src> <dest>
Copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the container at the path <dest>. *

COPY <src> <dest>
Copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>. *

*ADD or COPY - https://docs.docker.com/engine/articles/dockerfile_best-practices/#add-or-copy

Dockerfile Reference

VOLUME <path>
Creates a mount point with the specified name.

USER <user>
Sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it.

Dockerfile examples

```
FROM ubuntu
CMD echo "Hello world"
```

```
FROM openjdk
COPY target/hello.jar /usr/src/hello.jar
CMD java -cp /usr/src/hello.jar org.example.App
```

Dockerfile examples

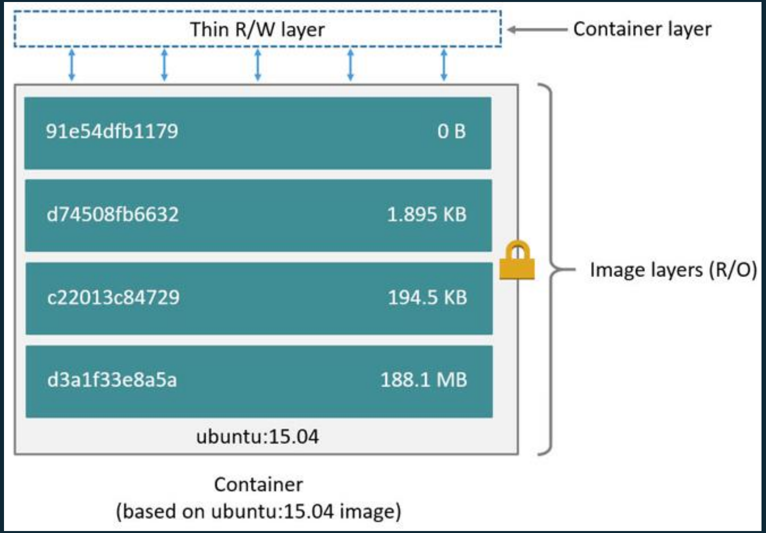
```
FROM centos:7

RUN yum -y --setopt=tsflags=nodocs update && \ yum
-y --setopt=tsflags=nodocs install httpd && \ yum
clean all

EXPOSE 80 # Simple startup script to avoid some
issues observed with container restart

ADD run-httpd.sh /run-httpd.sh
RUN chmod -v +x /run-httpd.sh

CMD ["/run-httpd.sh"]
```

Images vs. Containers

- Containers: running instance of an image.
- Containers have a top level writable layer.
- AWS analogy: EC2 AMIs vs EC2 Instances

$ docker build --rm -t my-app .

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```


Thin R/W layer
Container layer
91e54dfb1179 0 B
d74508fb6632 1.895 KB
c22013c84729 194.5 KB
d3a1f33e8a5a 188.1 MB
Image layers (R/O)
ubuntu:15.04
Container
(based on ubuntu:15.04 image)