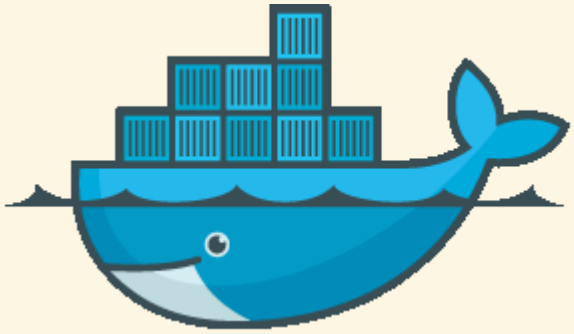


Introduction to Docker Containers

Micro-Services Academy, November 2018

Yonatan Bitton/@bityob

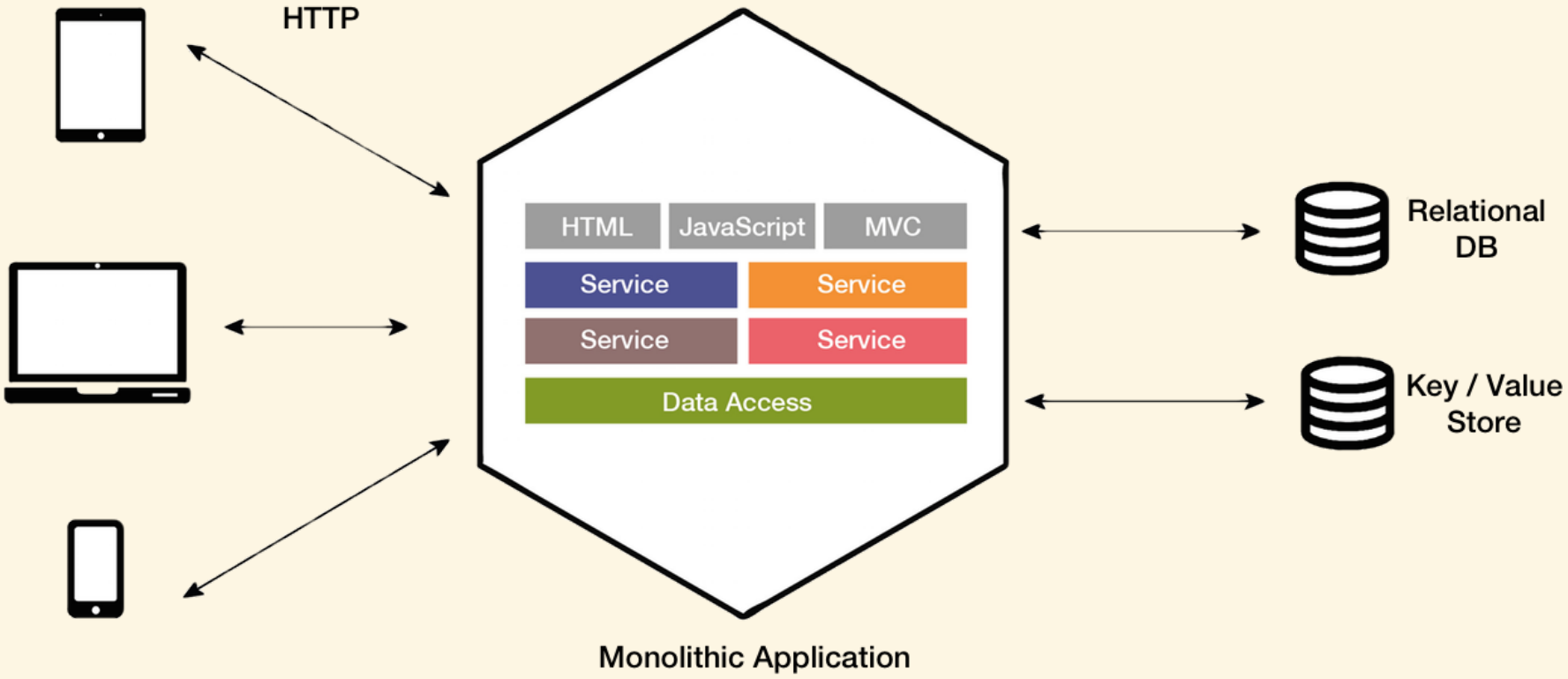
Agenda

- Micro Services
- Containers
- Docker

What is a Monolitic Service

*In software engineering, a monolithic application describes a **single-tiered software** application... code are combined into a single program from a single platform.*

What is a Monolitic Service



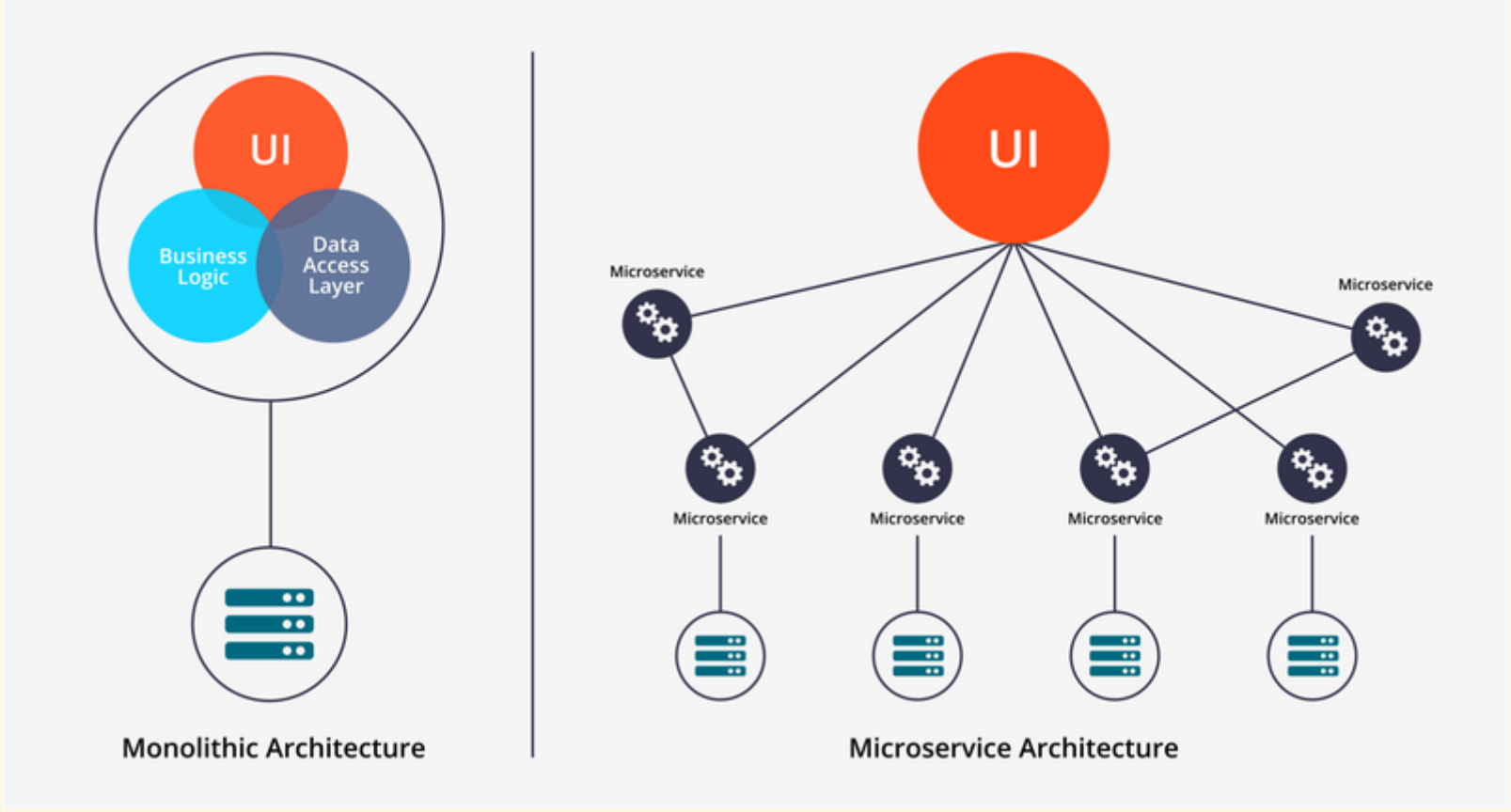
Strengths of the Monolithic Architecture

- Less cross-cutting concerns
- Easier debugging and testing
- Simple to deploy

Weaknesses of the Monolithic Architecture

- Understanding
- Making changes
- Scalability
- New technology barriers

What is a Micro Service

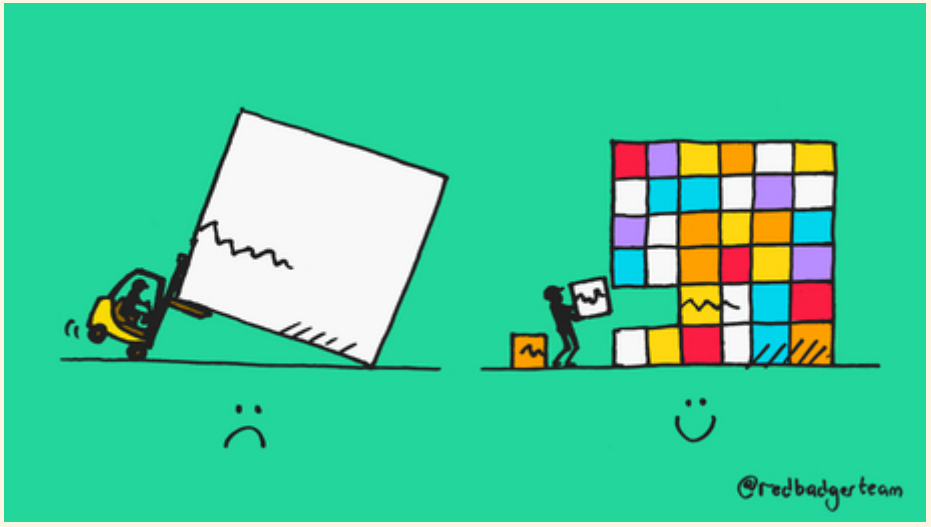


Source: Medium/startlovingyourself

Why Micro-services?

- Single Responsibility Principle
- Application is easier to understand, develop and test
- Allows high scalability and reusability
- Parallelizes development
- Enable continuous delivery and deployment
- Better fault isolation
- Code can be written in different languages

Micro Services using Containers



Source: RedbadgerTeam

World before Containers

Multiplicity of Goods







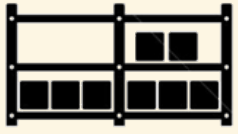
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing









Can I transport quickly and smoothly (e.g. from boat to train to truck)

What are Containers

Container Adoption

By 2020, more than 50% of global organizations will be running containerized applications in production

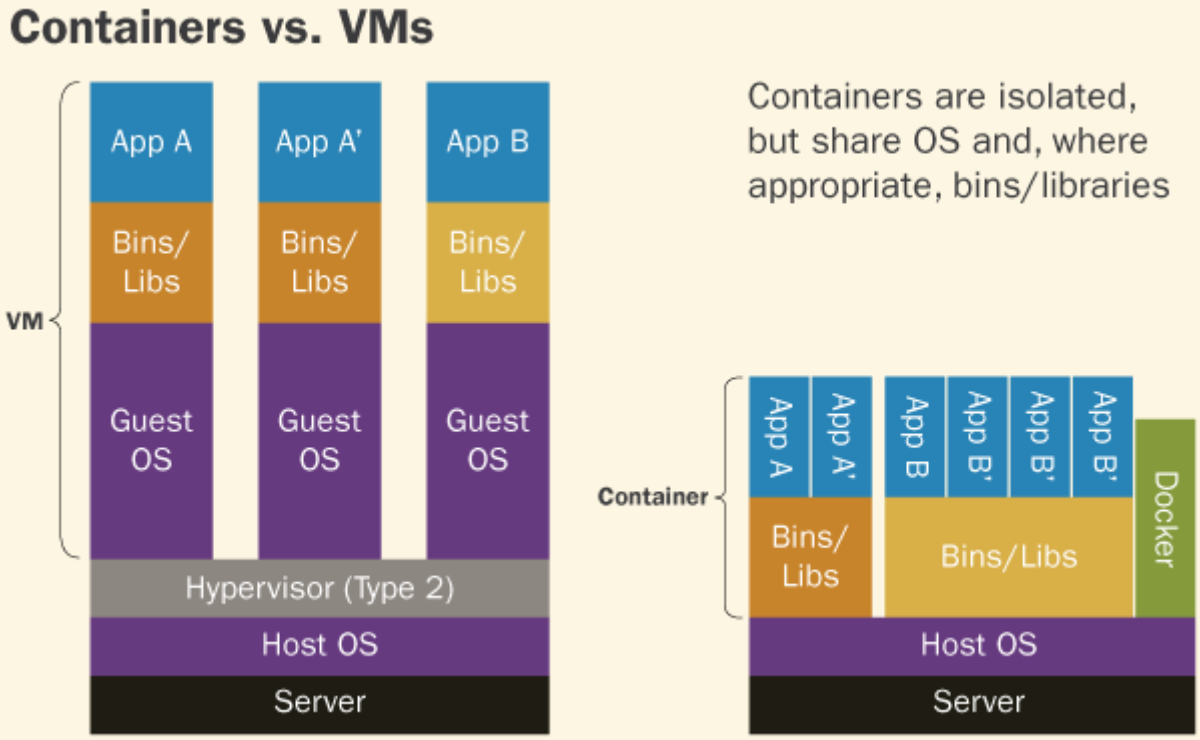
Source: Gartner

What is Docker

Docker is an open platform for developing, shipping, and running applications.

Docker allows you to package an application with all of its dependencies into a standardized unit for software development.

Docker vs VMs



Docker Benefits

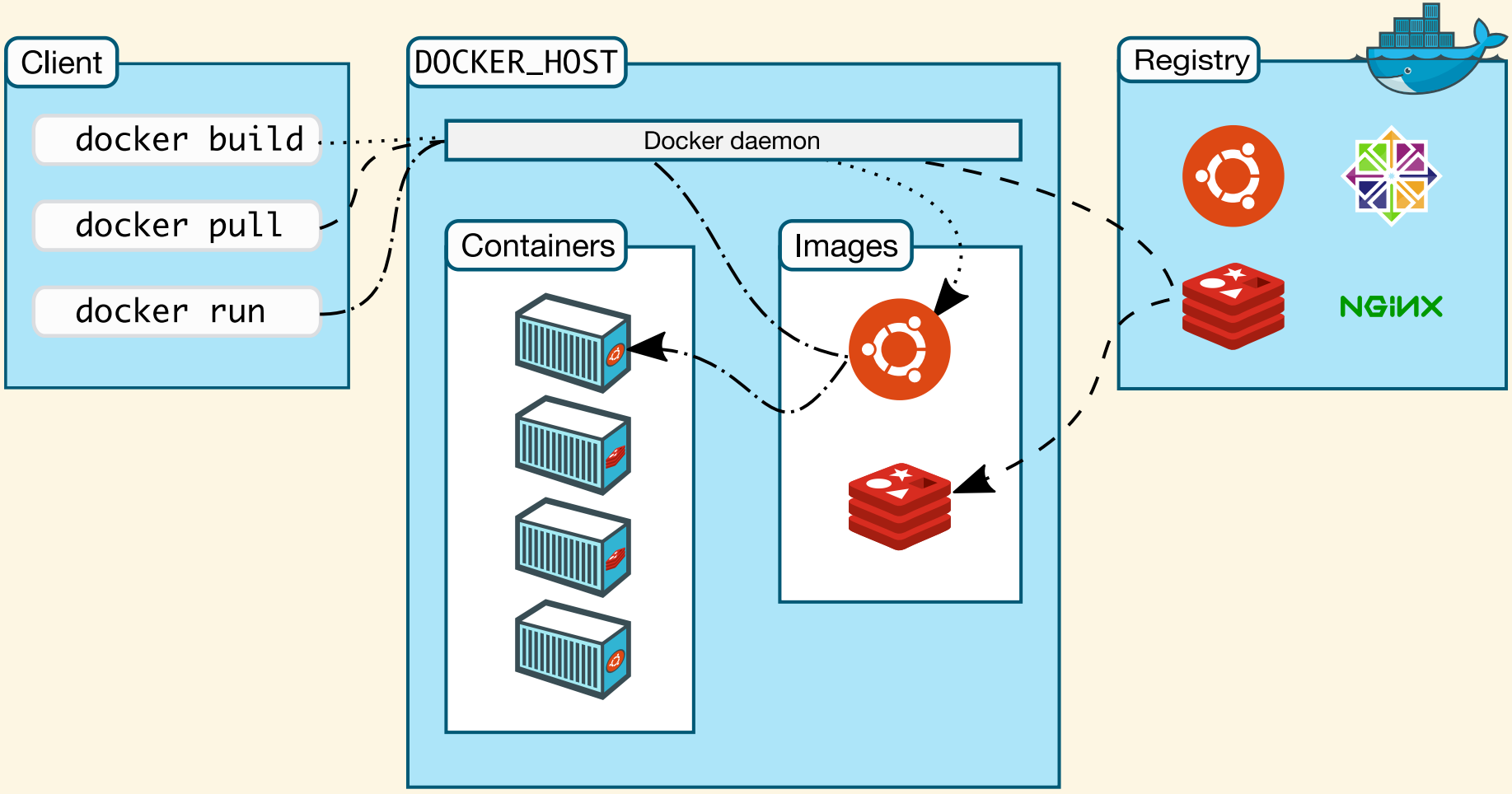
Common Docker usages

- Sandbox environment (develop, test, debug, educate)
- Continuous Integration & Deployment
- Scaling apps
- Development collaboration
- Local development

Technology behind Docker

- Linux x86-64
- Go language
- Namespaces (pid, net, ipc, mnt, uts)
- Control Groups (cgroups)
- Union file systems (UnionFS)
- Client - Server (deamon) architecture
- Container format (libcontainer)

The Docker architecture



See more at Understanding docker

Docker components/objects

- (Docker) client
- daemon/engine
- registry
- image
- container
- Dockerfile

Docker client

It is the primary user interface to Docker. It accepts commands from the user and communicates back and forth with a Docker daemon.

Docker daemon/engine

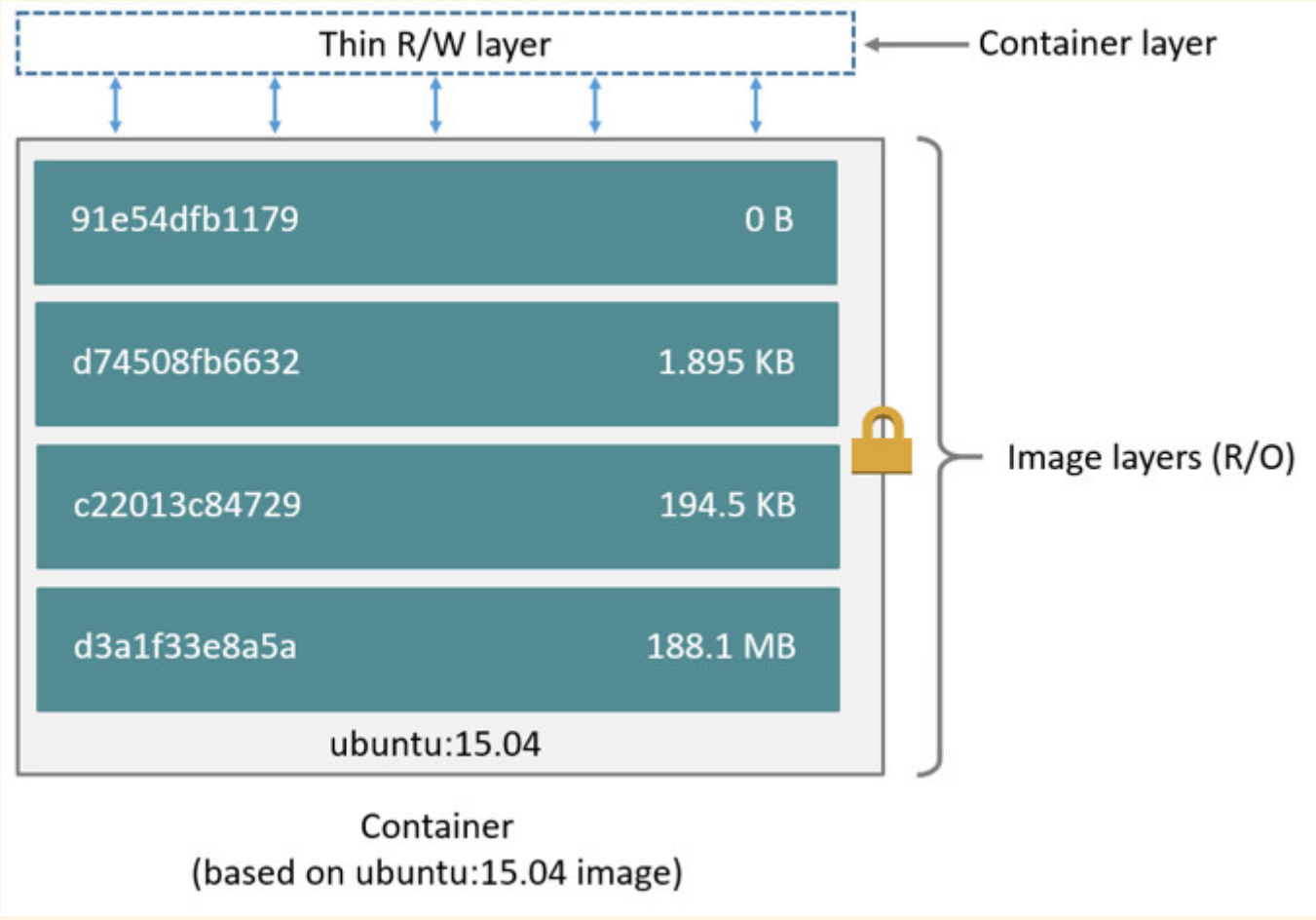
It runs on a host machine. The user does not directly interact with the daemon, but instead through the Docker client with the RESTful api or sockets.

Docker distribution/registry

A (hosted) service containing repositories of images which responds to the Registry API.

The docker image

The docker container



The Dockerfile

A Dockerfile is a text document that contains all the commands a user could call on the command line to create an image.

Docker Run With Port Mapping

```
docker run -d -p 8080:80 nginx
```

Docker Run With File System Mount

```
docker run -p 8888:80 -v <local_path>:/usr/share/nginx/html nginx
```

Docker Common Commands

```
// List all images that are locally stored with the Docker engine
docker images

// Delete an image from the local image store
docker rmi alpine:3.4

// Stop a running container through SIGTERM
docker stop web

//Stop a running container through SIGKILL
docker kill web

// List the running containers
docker ps

// Delete container (using container name or id)
docker rm web

// Print the last 100 lines of a container's logs
docker logs --tail 100 web
```

Convert An App into Docker Image

```python
import responder
from datetime import datetime
import socket

api = responder.API()

@api.route("/")
async def root(req, resp):
    resp.text = f"Hello Micro Service Academy!\n" + \
                f"Time Now: {datetime.now()}\n" + \
                f"Hostname: {socket.gethostname()}"

@api.route("/name/{name}")
async def hello_name(req, resp, *, name):
    resp.text = f"Hello {name}!"

@api.route("/index")
async def index(req, resp):
    resp.text = open('index.html').read()

if __name__ == '__main__':
    api.run(port=5000)
```

Overview of Dockerfile for our App

```
1 # Use an official Python runtime as a parent image
2 FROM python:3.6-slim
3
4 # Install update packages list
5 RUN apt-get update
6
7 # Install linux packages
8 RUN apt-get install -y gcc
9
10 # Install any needed packages
11 RUN pip install responder
12
13 # Set the working directory to /app
14 WORKDIR /app
15
16 # Copy the current directory contents into the container at /app
17 COPY . /app
18
19 # Define environment variable
20 ENV PORT '5000'
21
22 # Make port 80 available to the world outside this container
23 EXPOSE 5000
24
25 # Run app.py when the container launches
26 CMD python3 app.py
```

Build the image

```
docker build . -t docker.io/bityob/docker-python-app
```

Run the application using Docker

```
docker run --name webapp -p 5555:5000 docker.io/bityob/docker-python-app
```

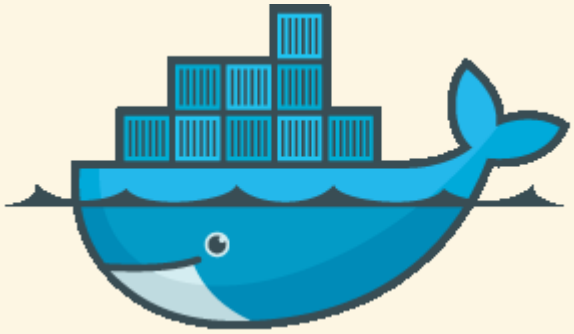
SSH into the container

```
docker exec -it webapp bash
```

Instead of Resources

- Awesome Docker (list of Docker resources)
- Docker cheat sheet (GitHub, Docker Pdf)
- Docker aliases/shortcuts
- Docker Tips examples/tips
- Introduction to Docker (US PyCon 2016, Slides)

Questions?



docker

In this presentation I used Reveal.js and slides/ideas from theodorosploumis.