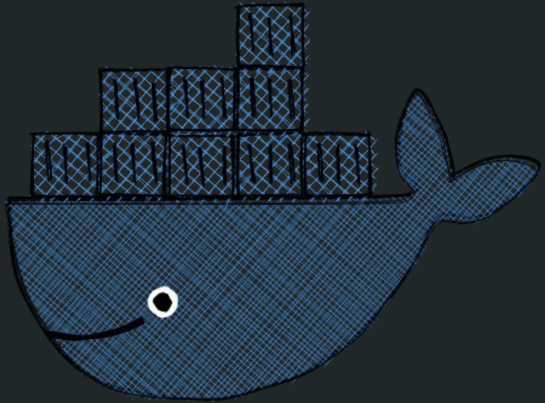


Docker Workshop

Jayapriya Pai

About me

Who am I?

- Senior Software Engineer at Red Hat
- Previously at Cerner as System Architect and Test Engineer at Infosys
- 7.5 years of Industry Experience in cross functional backgrounds QE, SRE and SE

Social Profiles

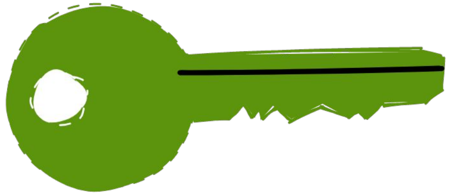
- https://github.com/slashpai
- https://www.linkedin.com/in/jayapriyapai/
- https://slashpai.medium.com/

Agenda

- History of application deployments
- Why Docker?
- Docker Overview
- Docker Architecture
- Handson with docker commands
- Building applications with docker
- Using docker-compose for running multi-container applications

Key Takeaway

- Beginner to intermediate level understanding of docker to explore further



Pre-requisite

- Basic knowledge on linux and virtual machines
- Basic programming knowledge
- Comfortable working with linux terminal

Softwares needed for training

- Docker installed on machine
- Git installed (this should be part of most linux distributions install if it doesn't)
- Git-bash incase of windows users

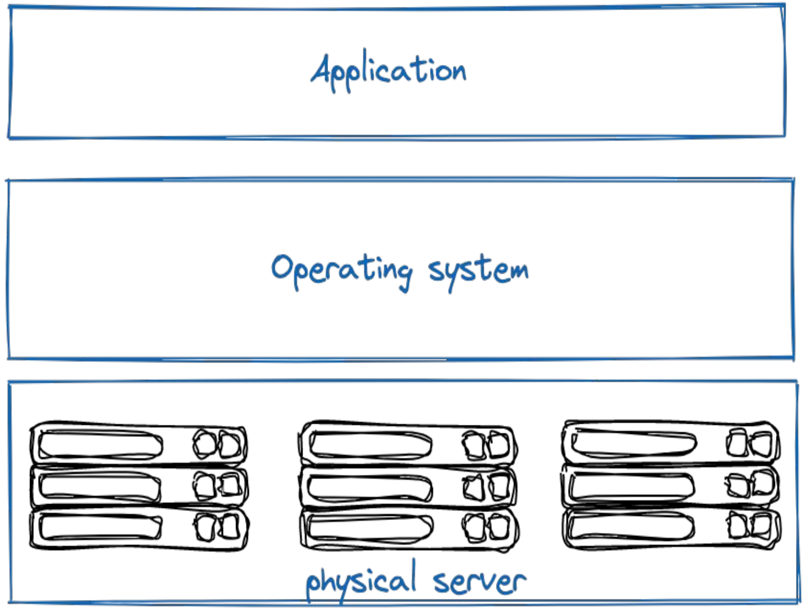
History of application deployments

Back in the history

One physical server per application

Limitations

- Huge costs
- Low resource utilization
- Difficult to port to another machine
- Slow deployment
- Difficult to scale



Virtual Machines

One server can host multiple applications

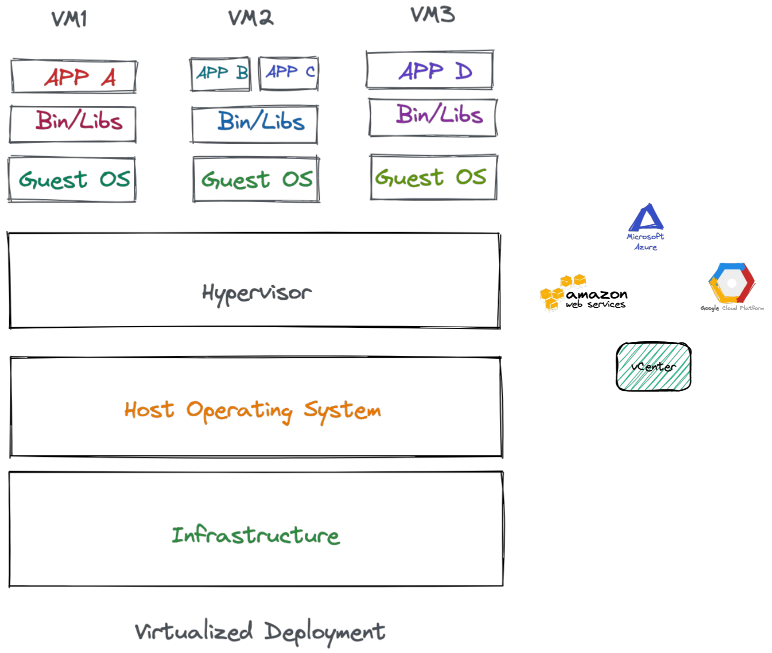
- One or two applications runs in a VM

Advantages:

- Better resource utilization
- Easier to scale
- Utilize cloud providers
  - Elasticity
  - Pay per usage

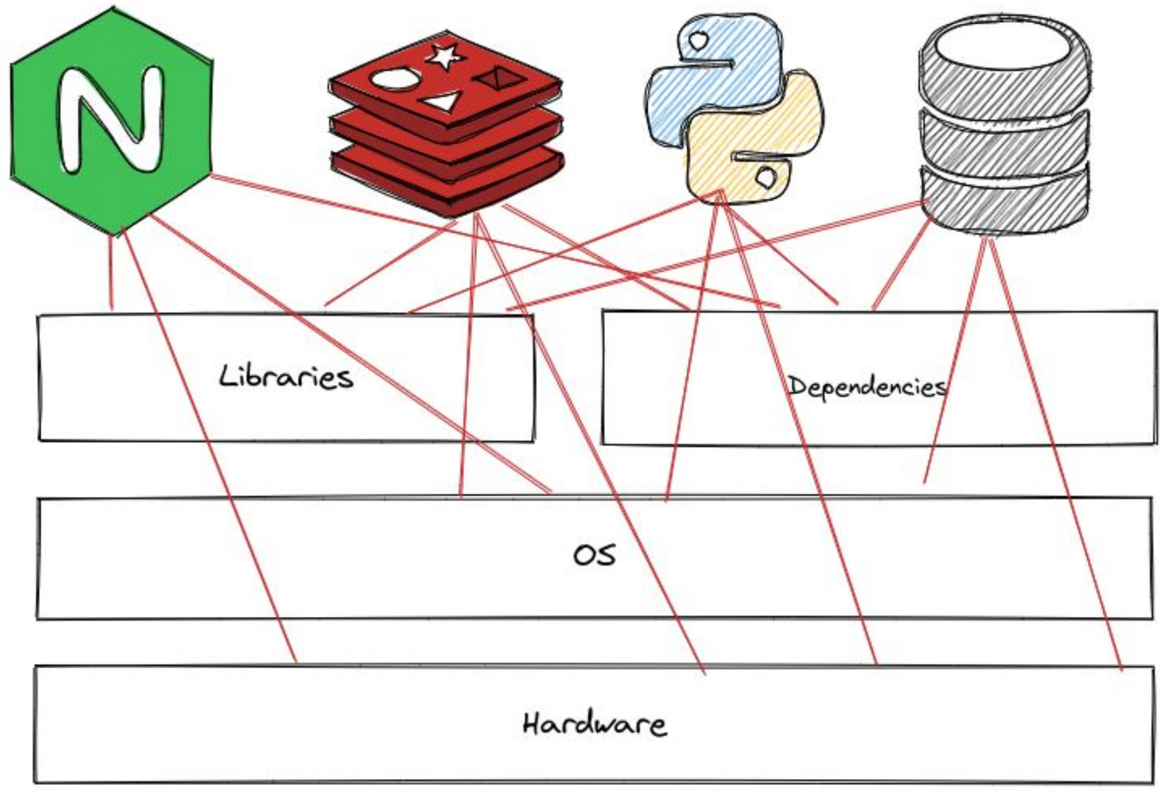
Limitations:

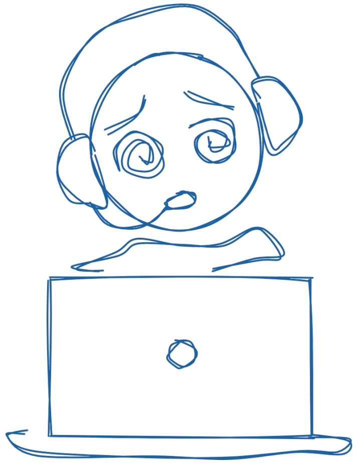
- Each VM requires CPU, RAM, Storage, OS
- More VMs -> More resource usage
- Wasted space for Guest OS
- No guarantee on portability of application



Why Docker/Containers?

Finding dependencies which fits all applications





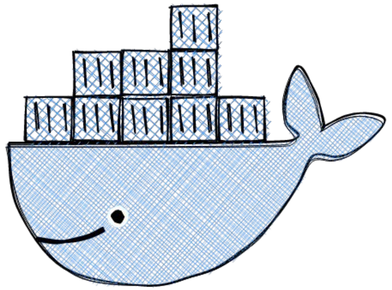
Problems

- Dependency issues
- Long time to setup system
- Drift in dev, staging and production environment
- Difficulty in upgrading software finding right compatibility between components

Building, developing and shipping application difficult

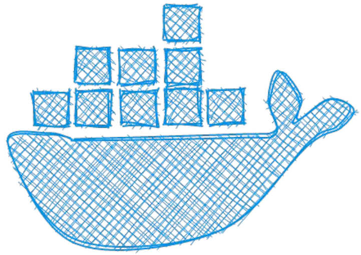
How docker helps

- Helps you create a single deployable unit for your application
- Has everything the application needs to work
  - Includes the code (or binary), the runtime, the system tools and libraries
- Helps to maintain identical environments in dev, staging, production



What can docker do?

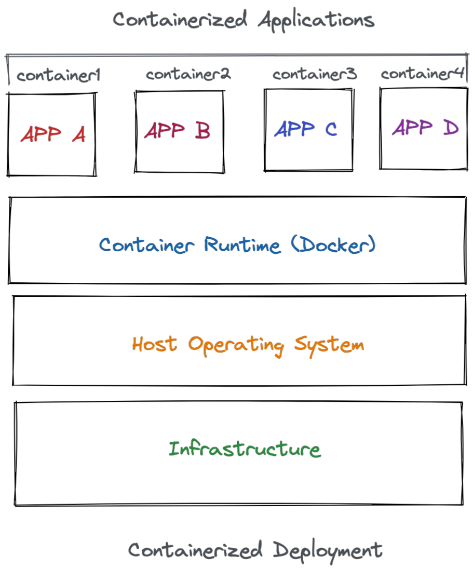
- Fast, consistent delivery of your applications
  - Allows developers to work in standardized environments using local containers
  - Great for continuous integration and continuous delivery (CI/CD) workflows
- Responsive deployment and scaling
  - Highly portable workloads (local, VM, physical machines, cloud provider or in hybrid fashion)
  - lightweight nature also make it easy to dynamically manage workloads in near real time
- Running more workloads on the same hardware
  - Cost effective
  - Use more of your compute capacity to achieve your business goals



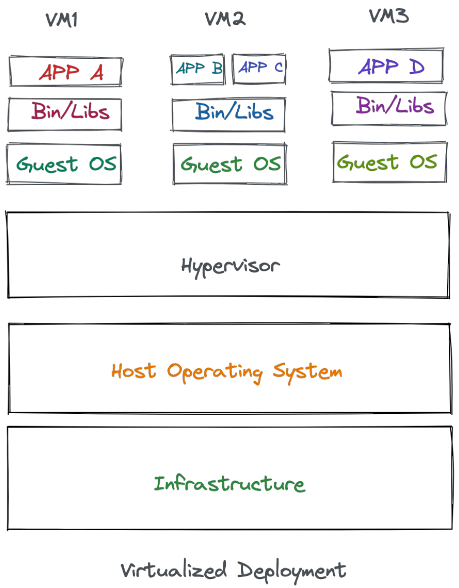
How containers work?

- **Namespaces**
  a. Isolating resources per process or group of processes
- **Control groups (cgroups)**
  a. Accounts for the resource usage (CPU, memory, disk I/O, network, etc.) of a set of processes
  b. cgroup ensures that containers only use the resources they need and set up limits if needed
- **Union file systems**
  a. You can think of a Union File System as a stackable file system
  b. Layered systems offer two main benefits:
     i. **Duplication-free:** layers help avoid duplicating a complete set of files every time you use an image to create and run a new container, making instantiation of docker containers very fast and cheap.
     ii. **Layer segregation:** Making a change is much faster — when you change an image, Docker only propagates the updates to the layer that was changed.

Containers vs Virtual Machines



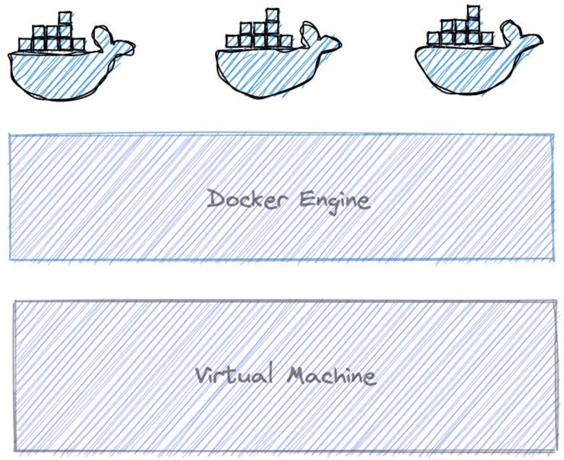
Containers are app level constructs



VM's are infrastructure level construct to turn one physical machine to multiple servers

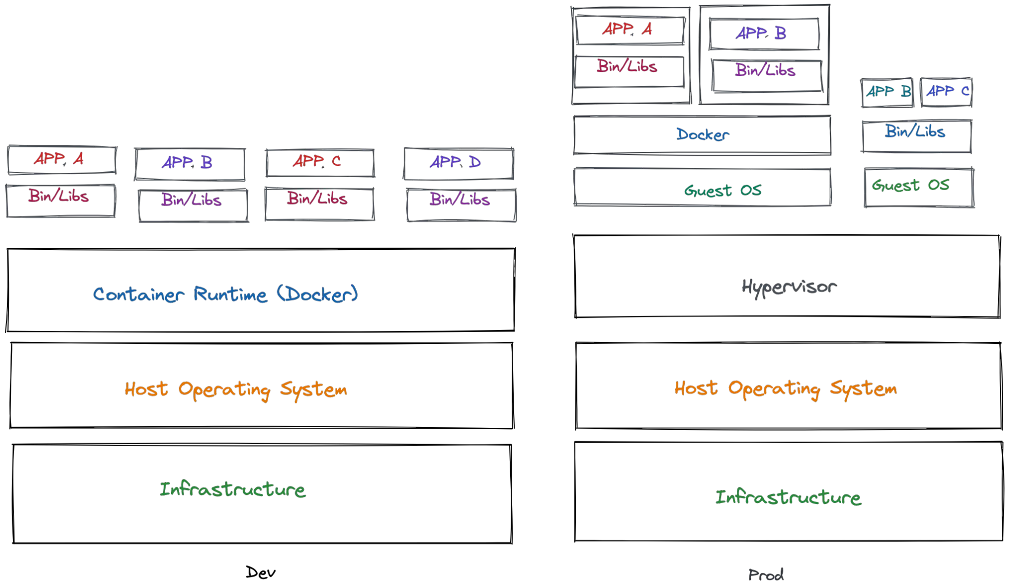
The Future of Docker

Docker and VMs will co-exist



Combining container and virtual machines

- Combining both container and vm provides flexibility in deploy and manage applications

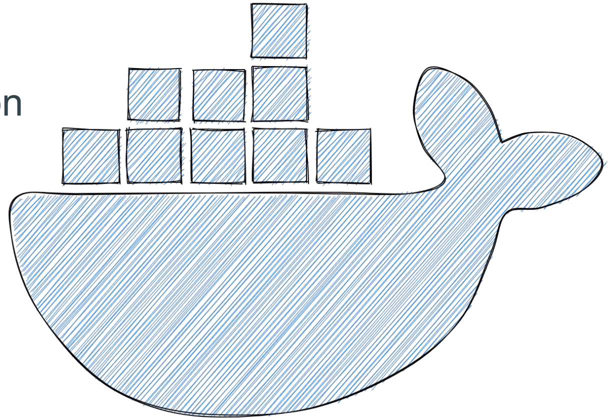


Docker Overview

Docker Overview

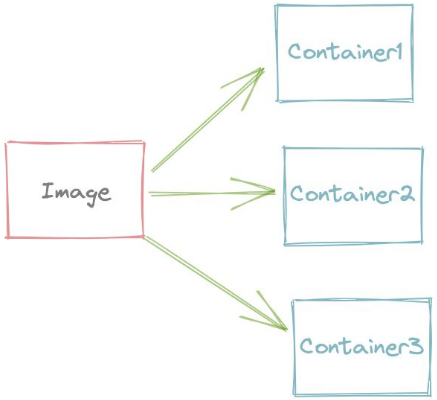
An open platform for building, shipping and running applications

- Separate your applications from your infrastructure
- Reduce the delay between writing code and running it in production



Docker Basics

- Image
  - Images define both what you want your packaged application and its dependencies to look like *and* what processes to run when it's launched
- Container
  - Built off Docker images
  - Since images are read-only, Docker adds a read-write file system over the read-only file system of the image to create a container
- Engine
  - Client-server application for building and containerizing your applications
- Registry
  - Lets you store and distribute Docker images



Foundation: Docker Engine

- **Docker Daemon**: A persistent background process that manages Docker images, containers, networks, and storage volumes. The Docker daemon constantly listens for Docker API requests and processes them.
- **Docker Engine REST API**: An API used by applications to interact with the Docker daemon; it can be accessed by an HTTP client.
- **Docker CLI**: A command line interface client for interacting with the Docker daemon. It greatly simplifies how you manage container instances and is one of the key reasons why developers love using Docker.

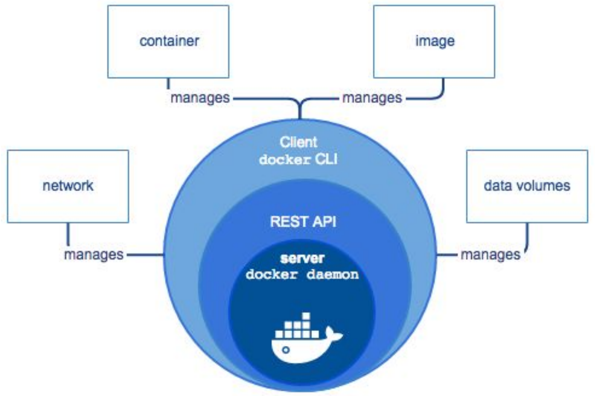


Image Courtesy: https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/

Docker Architecture

Docker Architecture

The Docker architecture uses a client-server model and comprises of the Docker Client, Docker Host, Network and Storage components, and the Docker Registry/Hub

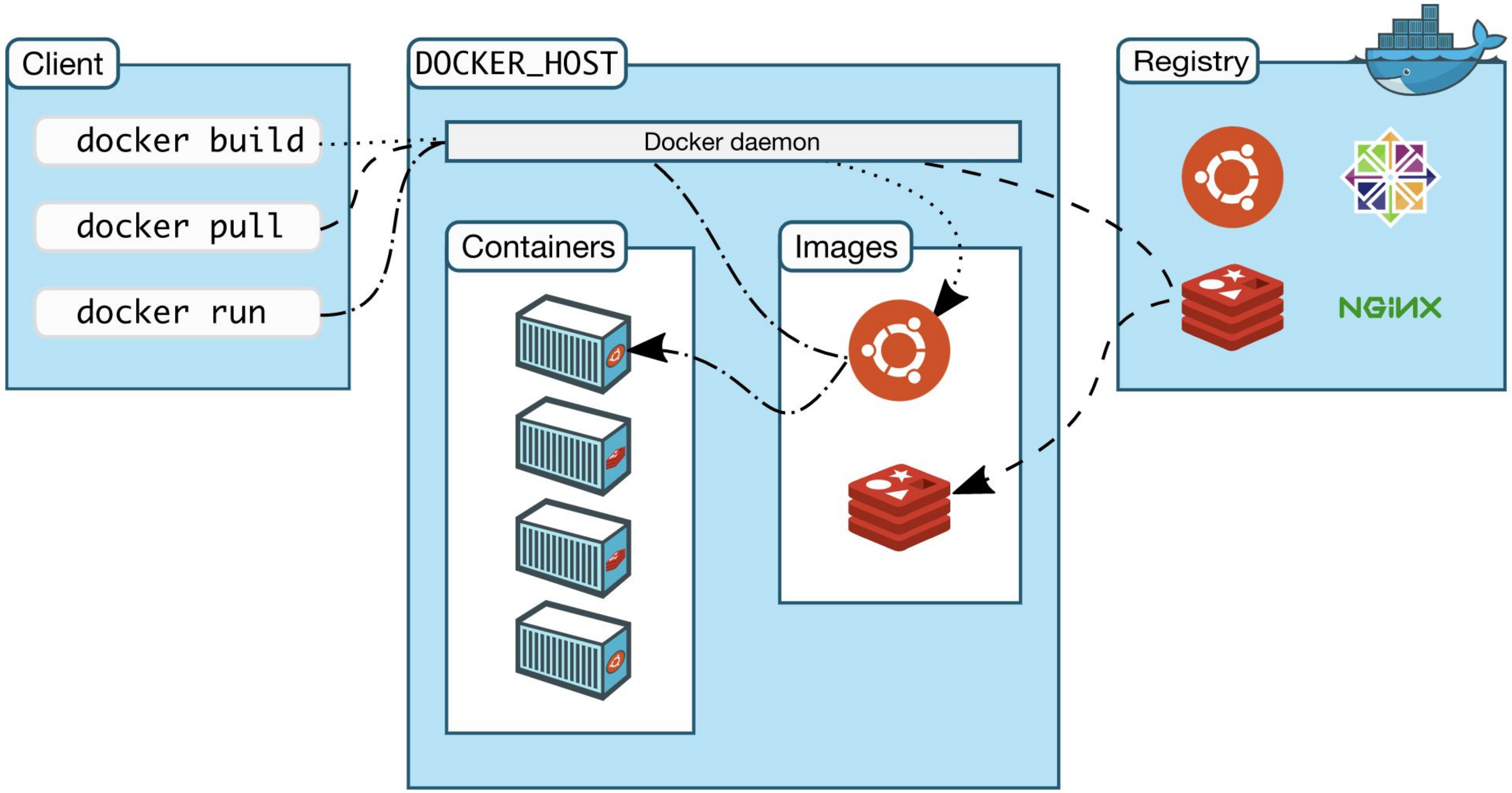


Image Courtesy: https://docs.docker.com/get-started/overview/

Dockerfile

- A Dockerfile is where you write the instructions to build a Docker image
- Docker can build images automatically by reading the instructions from a Dockerfile
- A text document that contains all the commands a user could call on the command line to assemble an image
- Each instruction in the Dockerfile adds a new “layer” to the image
- With layers representing a portion of the images file system that either adds to or replaces the layer below it
- Layers are key to Docker’s lightweight yet powerful structure

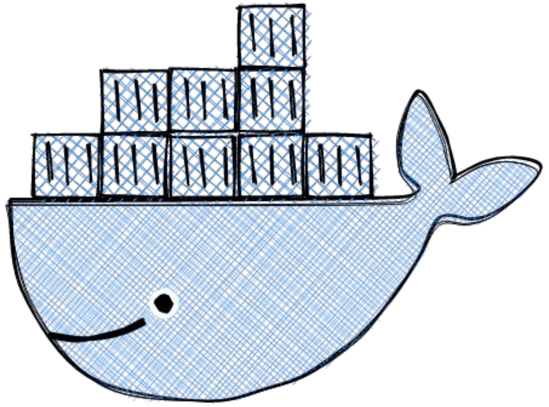
Dockerfile

```
FROM node:15.3.0-alpine3.10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
COPY package*.json ./

RUN npm install

# Bundle app source
COPY . .

EXPOSE 3000
CMD [ "node", "index.js" ]
```

Docker Handson

Docker in Action

https://slashpai.github.io/presentations/docker_in_action/#/



Docker Alternatives

- rkt (pronounced as Rocket)
- LXD
- Podman
- CRI-O
- Containerd

What next?

Container Orchestration

Options?

- Kubernetes
- Apache Mesos
- Docker Swarm

References

- https://docs.docker.com/get-started/overview/
- https://www.slideshare.net/Docker/introduction-to-docker-2017?qid=946fadc3-bb76-4ec1-ba40-564023b21bac&v=&b=&from_search=36
- https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/
- https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/
- https://dzone.com/articles/docker-layers-explained

More Resources for Learning

- https://www.manning.com/books/docker-in-action-second-edition#toc
- https://www.katacoda.com/courses/docker

Questions?

Thank you