INTRODUCTION TO DOCKER

Stephen Schaub

Application Dependencies

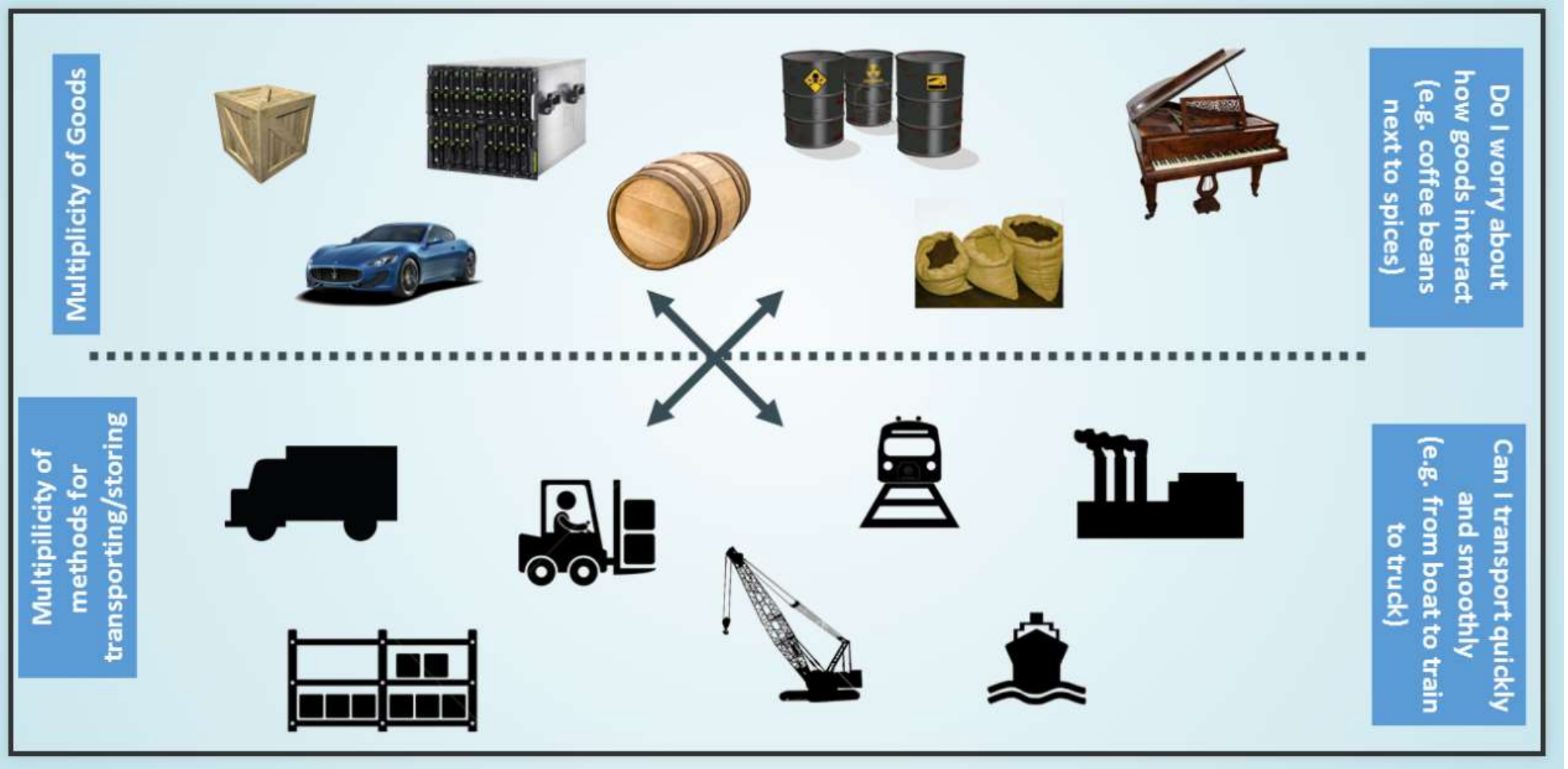
- Modern web applications rely on several components and related configuration
  - Operating system
  - Language Runtime
  - Third Party Libraries
  - Application Server
  - Web Server
  - Database Server
  - Filesystem Permissions
- Applications are run in different environments
  - Developer workstation
  - QA Lab
  - Deployment server

Dependency Matrix

- Keeping different application environments synchronized is a challenge

	Developer	QA	Deployment
Operating system	?	?	?
Language Runtime	?	?	?
Third Party Libraries	?	?	?
Application Server	?	?	?
Web Server	?	?	?
Database Server	?	?	?
Filesystem Permissions	?	?	?

Cargo Transport Pre-1960



From https://pointful.github.io/docker-intro

Solution: Intermodal Shipping Container

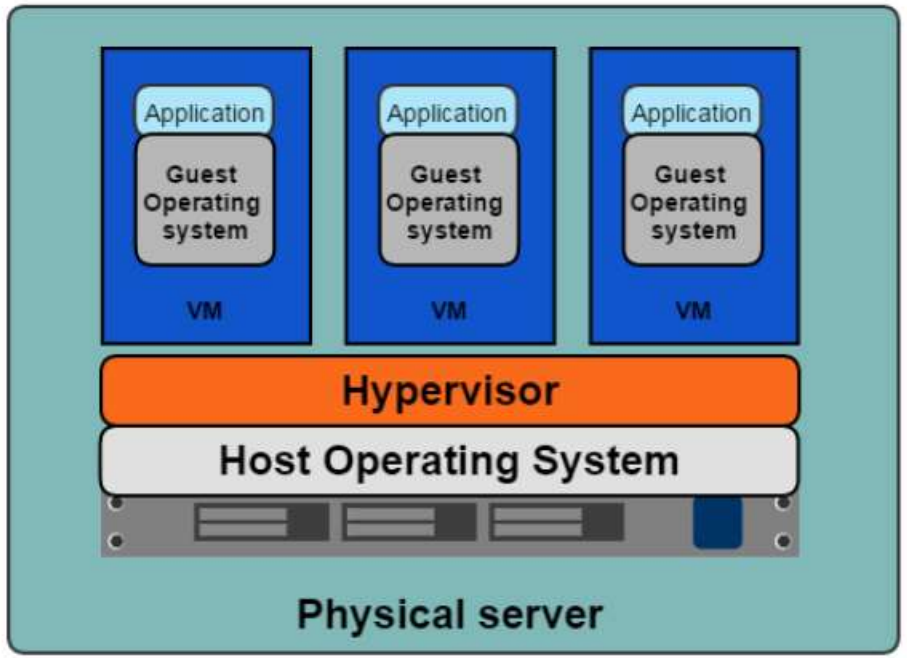
From https://pointful.github.io/docker-intro

Meet Docker

- A Container System for Applications
- A container consists of an application, together with its dependencies
  - … including web server, database server, application server …
- Can be deployed and run on almost any hardware platform with a compatible OS
- Why Developers Care:
  - Build once … run “anywhere”
  - A clean, safe, portable runtime environment for your app

What about Virtual Machines?

- One physical server can run multiple virtual machines
- Each virtual machine can run multiple applications
- Benefits:
  - Good resource utilization
  - Leverage cloud computing and scalability
  - Pay as you go

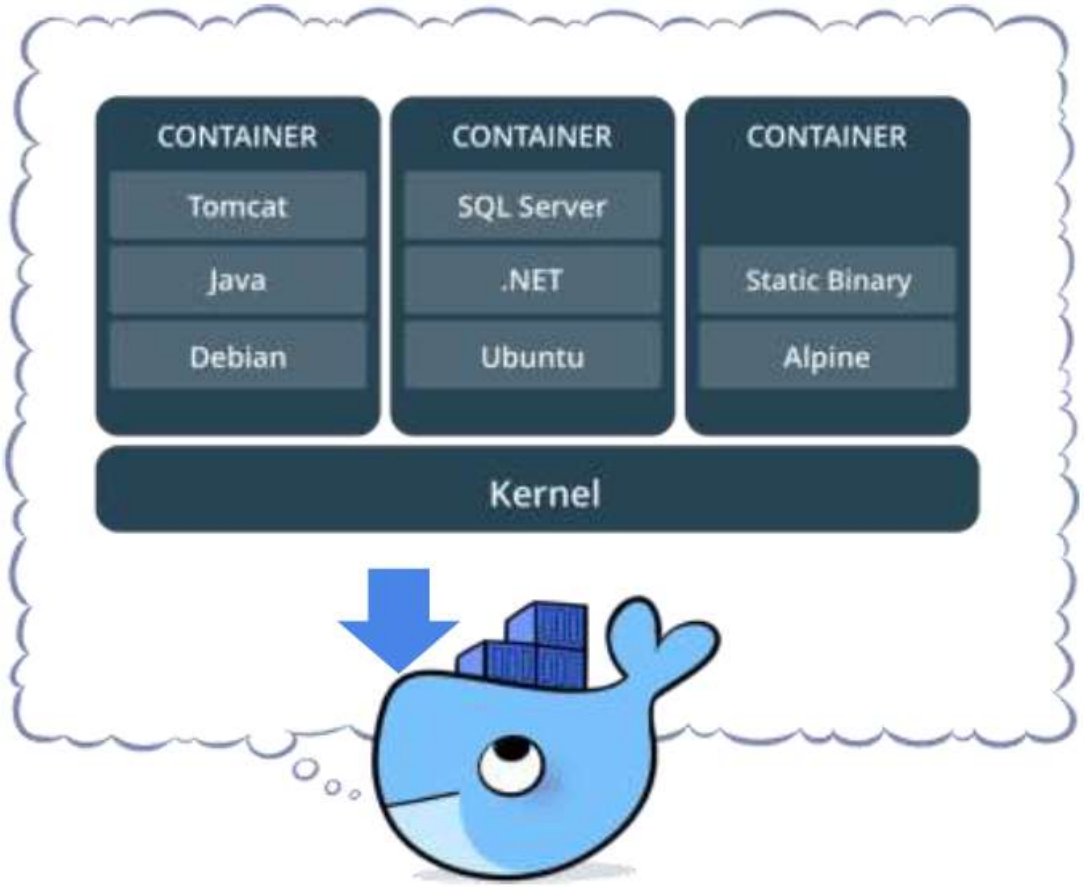


Limitations of VM's

- Each VM requires an entire guest OS
- Guest OS means wasted resources
- Doesn't address the application dependency issue

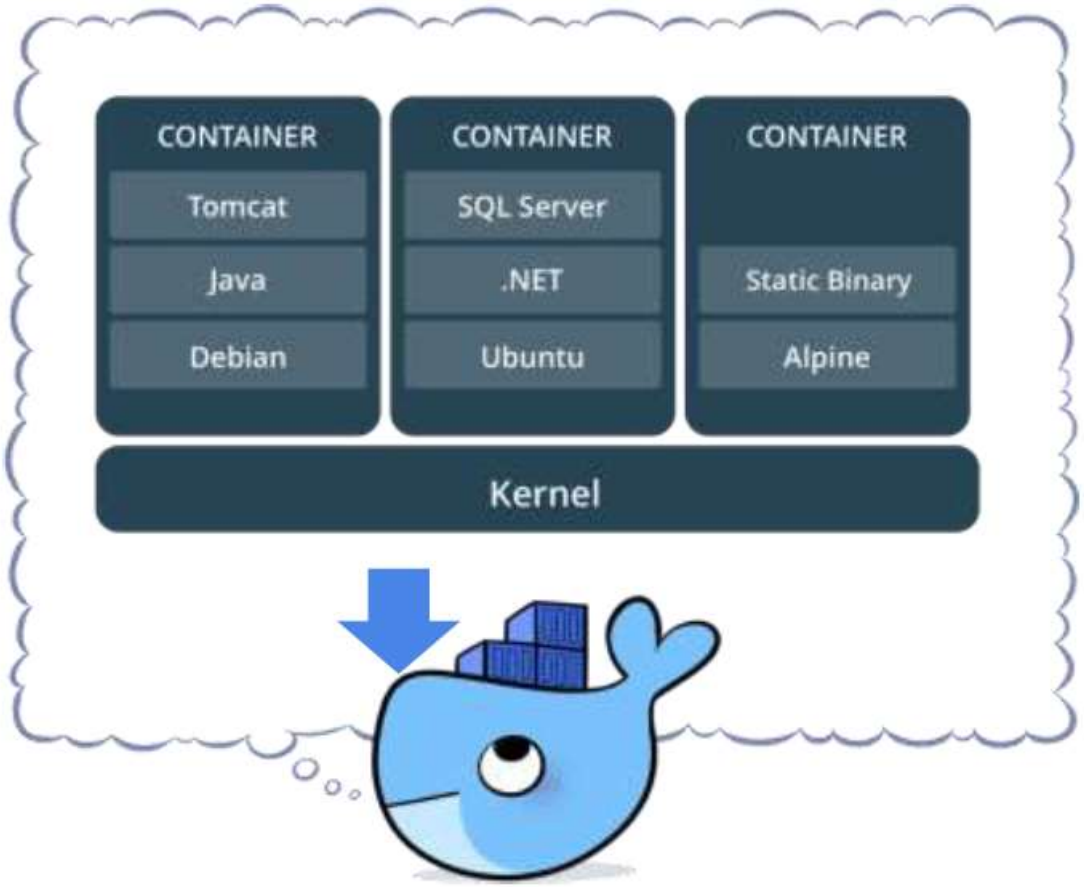
Containers

- A container is a package containing an application with its dependencies
- A Virtual Machine can run multiple containers in the same OS ...
  - ... just like it can run multiple applications in the same OS ...



Containers

- The containers are logically isolated from each other ...
  - ... but make better use of shared resources (CPU, RAM)
  - ... and enable convenient application deployment and upgrade



Containers vs. Virtual Machines (Similarities)

- Both technologies:
  - Run on a host OS (Windows / Linux)
  - Provide a virtualized computing environment for applications
    - Virtual filesystem, RAM, CPU
- A container is like a lightweight virtual machine:
  - Can be started and stopped
  - Admins can login to container via command line shell and interactively manipulate environment
  - Applications running in container can access network

Containers vs. Virtual Machines (Differences)

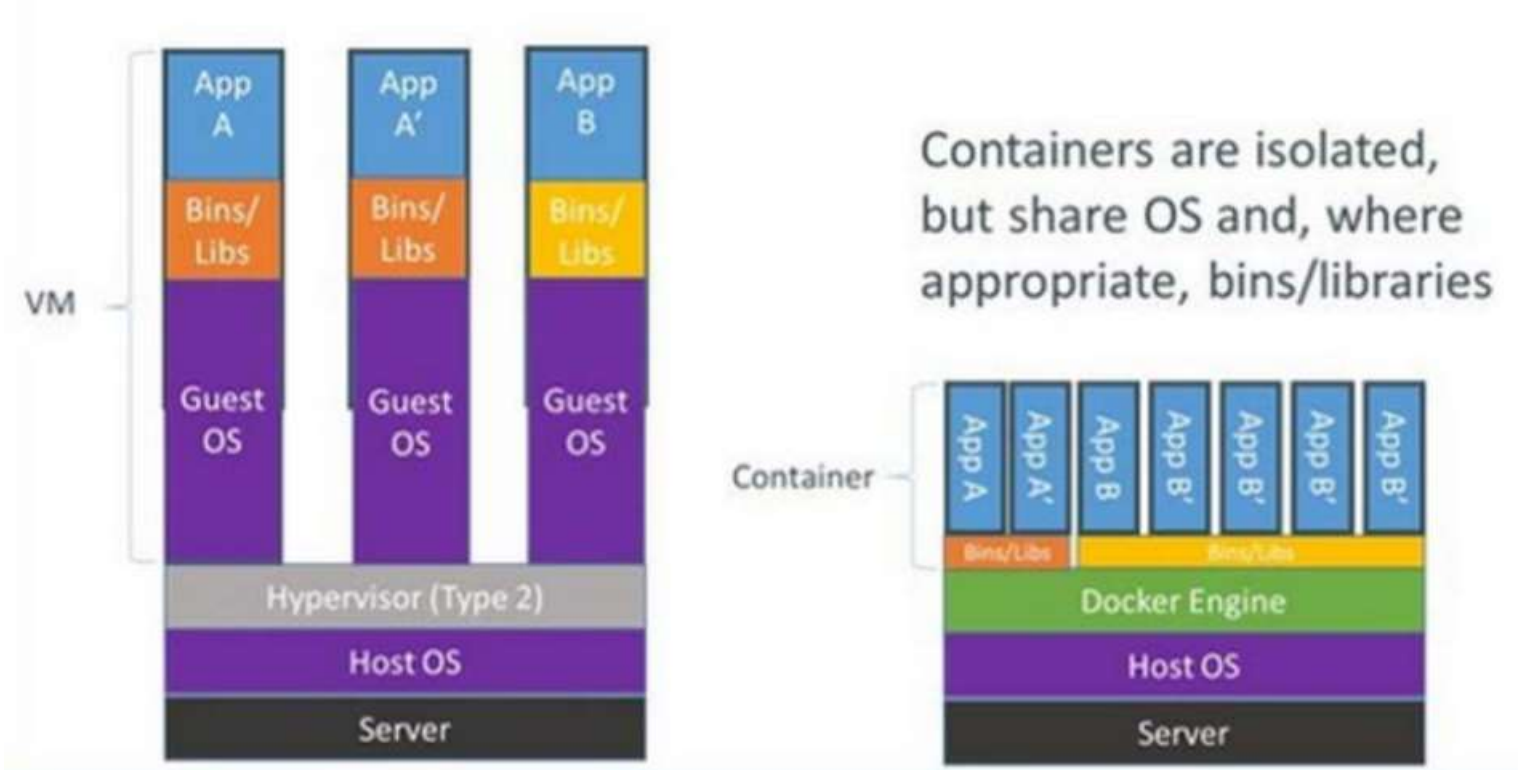
Virtual Machine

- Heavyweight
  - Boots in tens of seconds
  - Overhead of full guest OS
- No application isolation
  - Applications can interfere with each other
  - Hard to keep application dependencies separate
- No convenient application deployment / upgrade

Container

- Lightweight
  - Starts in seconds (or milliseconds!)
  - Very little overhead for each container
- Application isolation
  - Each application runs in its own isolated environment, together with its own dependencies
- Convenient application deployment and upgrade

Containers vs. VM's



From https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_497378.pdf

Containers vs. Virtual Machines

- Unlike traditional virtual machines, containers are:
  - Designed for scripted deployment of applications
  - Can be easily reset to a baseline state
  - Can easily access host filesystem

Docker Concepts

Key Docker Terms

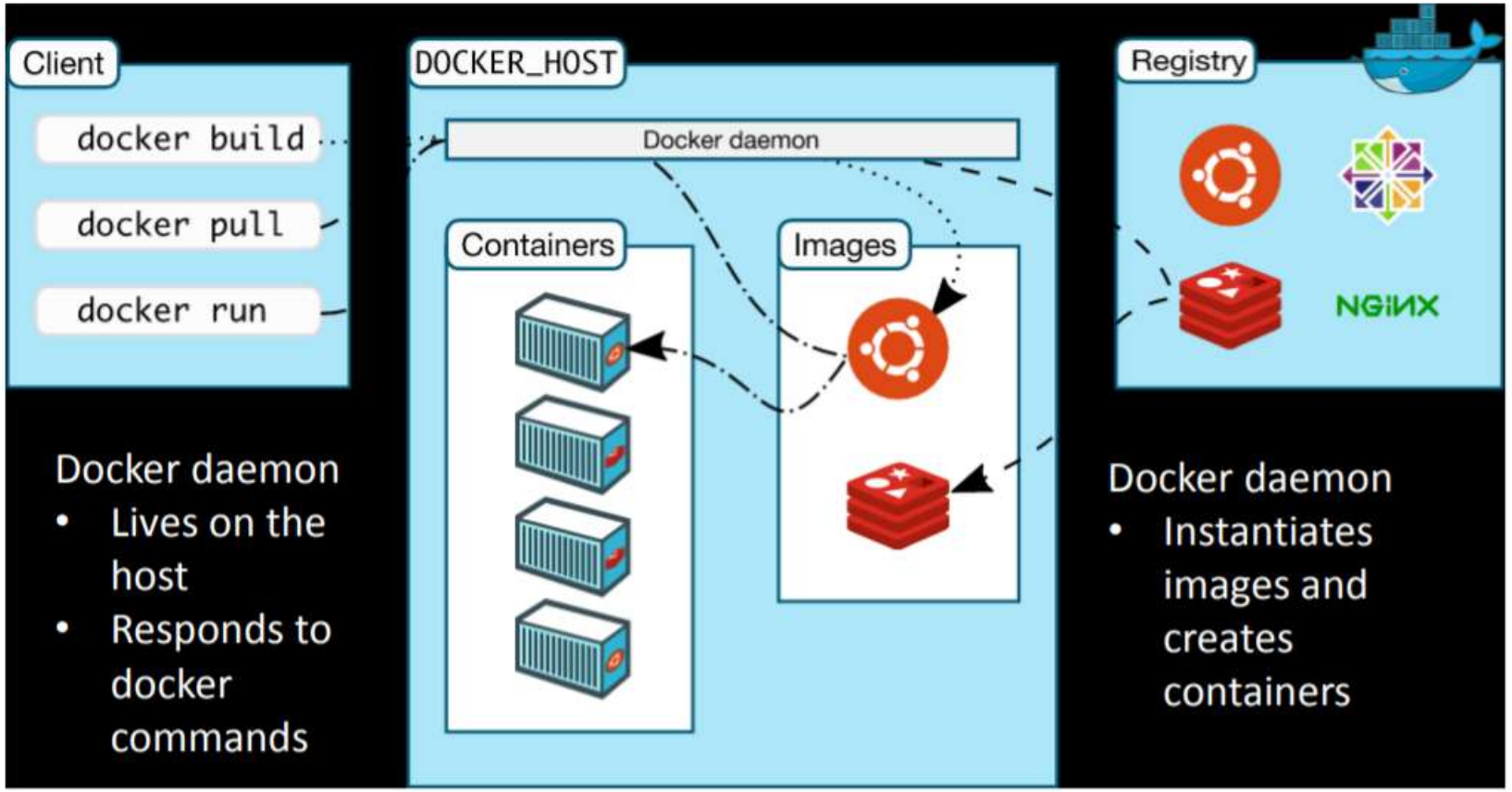
- Image
  - Basis of a Docker container. The initial container content.
- Container
  - The image when running.
- Engine
  - Software that manages containers.
  - Handles networking and container filesystems.
- Registry
  - Stores, distributes, and manages Images

Images and Containers

- Image
  - A software package containing configuration and applications
  - Must be installed on a local Docker host in order to run
  - Defined using a Dockerfile
  - Images can be versioned using tags
- Container
  - An Image running in a Docker host (an “instantiated Image”)
  - Multiple containers can be launched from the same Image
  - Each has its own private filesystem

Docker Architecture

- Image instantiated to form container



From https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_497378.pdf

Docker Registry

- Contains a library of predefined Images
- Examples:
  - hub.docker.com
  - Amazon ECR
- Docker Registry provides these capabilities:
  - You can create your own Image using an existing Image as a starting point
  - You can publish your Images in the Registry

Using Docker

Using Docker

- Run a command in a container
  - docker run -t ubuntu:18.04 **cat /etc/passwd**

Image to launch *Command to run*

What happens:

- Docker engine downloads ubuntu image tagged 18.04
- Docker engine starts container and executes command in the container
- Docker engine stops container

- Image is cached locally for subsequent launches

Using Docker

- Run an interactive shell in a container
  - docker run -i -t ubuntu:18.04 bash

What happens:

- Docker engine starts a new container using ubuntu:18.04 image
- Docker engine starts bash shell in container
  - -i option allows container bash shell to read from host shell stdin
- Container runs until bash process exits (when you press Ctrl-D or execute exit or logout)
- Any changes you make to the container's filesystem are discarded when container exits

Developing with Docker

- Want to run the app you're developing in Docker
- Problem: Docker can't access the files on the host filesystem
- Solution: Mount files from the host into container's private filesystem using -v option
- Example 1:

  - docker run -v /etc:/hostetc -t ubuntu:18.04 cat /hostetc/passwd
- Example 2: Run **hello.js** located in **/home/ubuntu** using image **node**
  - docker run -v /home/ubuntu:/myapp node:latest \
    node /myapp/hello.js

Running Server Apps in Docker

- Server apps listen for incoming connections on a port
- Map ports in container to ports on host using -p option
- Example: Run **webserver.js** located in **/home/ubuntu** using image **node,** mapping port 8888 in container to 80 on host
  - docker run -p 80:8888 \
    -v /home/ubuntu:/myapp node:latest \
    node /myapp/webserver.js

Using Docker Compose

- Docker Compose provides a convenient way to
  - Specify and configure an image to run
  - Start and stop multiple containers at the same time
- Basic usage:
  - Create a docker-compose.yml file
  - Start container with **docker-compose up -d**
  - Stop container with **docker-compose down**

docker-compose.yml

- Defines one or more “services” (containers)
- Each service specifies:
  - image to run
  - image configuration (ex. command to run in the container)
- Launch container from image:
  - **docker-compose up**
  - Container exits when command finishes

```
version: "2"
services:
  my_node_container:
    image: "node:9"
    command: "node -v"
```

Running Server Applications

- Use **ports** configuration to map ports on the host to ports in the container
- Use **volumes** option to specify a list of directories to mount
- Run container using
  - docker-compose up -d
- Stop container using
  - docker-compose down

```
version: "2"
services:
  my_node_container:
    image: "node:latest"
    working_dir: /app
    volumes:
      - ./:/app
    ports:
      - 80:8888
    command: "node webserver.js"
```

Creating Images

Layers

- Docker images are built using layers
- Images are built as follows:
  - Start with base image
  - Configure desired software for image using Dockerfile
- New image can be base for other images

Creating Images

- Images are defined in files named **Dockerfile**
- Dockerfile specifies
  - Base image
    - Example: FROM node:13.8.0
  - Files to copy into new image
    - COPY . /app
  - Default work directory for containers launched from image
    - WORKDIR /app
  - Commands to run to configure new image
    - RUN apt update && apt install mysql-client
  - Default command to run for containers launched from image
    - CMD node index.js

Building Images

- After creating Dockerfile, use **docker build** build to create an image
  - docker build -t my_container /path/to/directory/with/Dockerfile

Demo Time

Installation

- Install Docker Engine on Ubuntu 18.04
  - sudo apt install docker.io
- (Optional) Add your user account to docker group in /etc/group
  - Allows you to execute docker commands without “sudo”

- docker-compose down --rmi local
- -v removes the volume (containing the database)