

INTRODUCTION TO DOCKER

Stephen Schaub

Application Dependencies

2

- 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

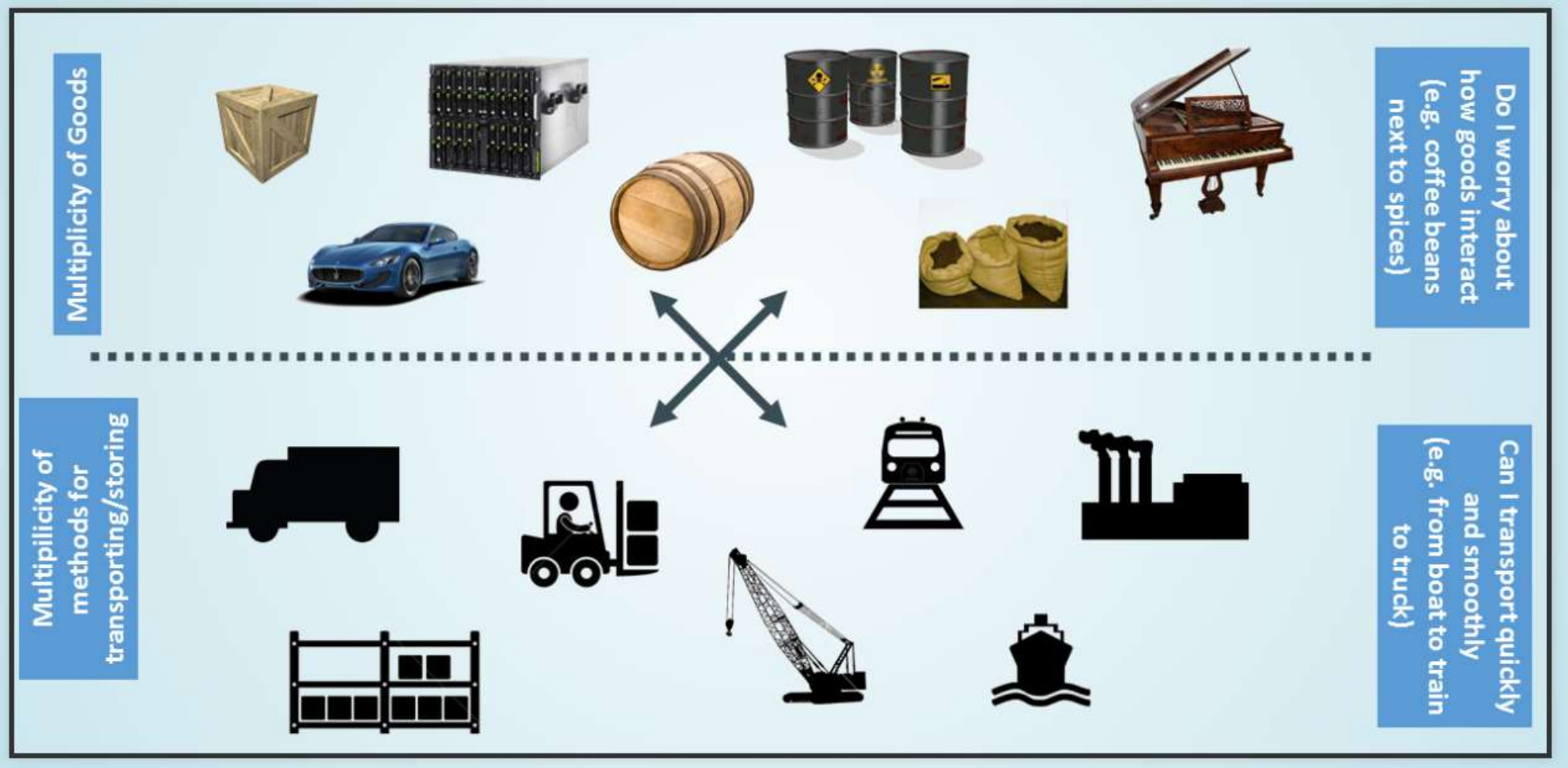
Dependency Matrix

3

- 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

5



Meet Docker

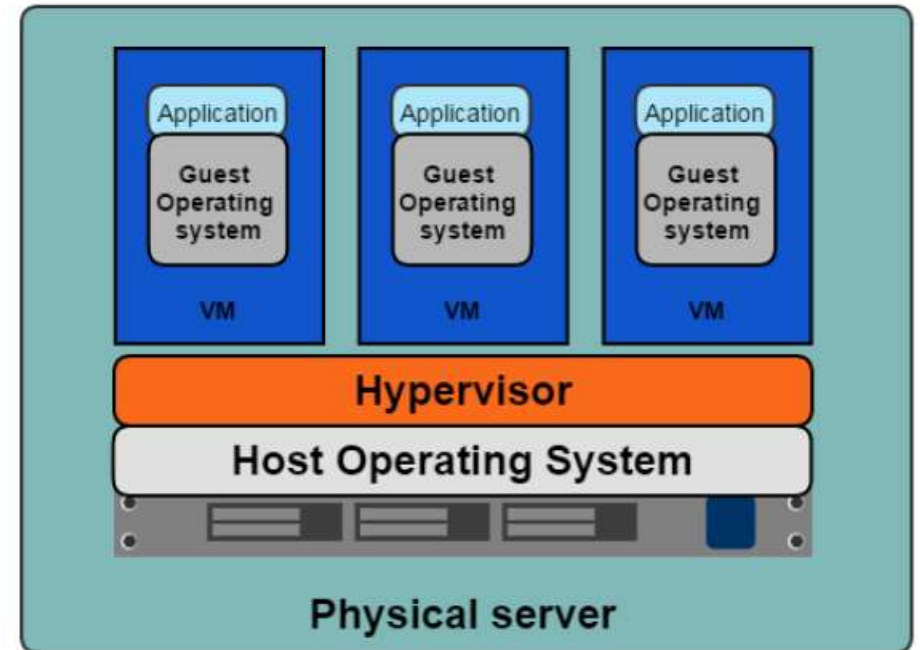
6

- A Container System for Applications
- A container consists of an application, together with its dependencies
 - ▣ ... including OS, 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?

7

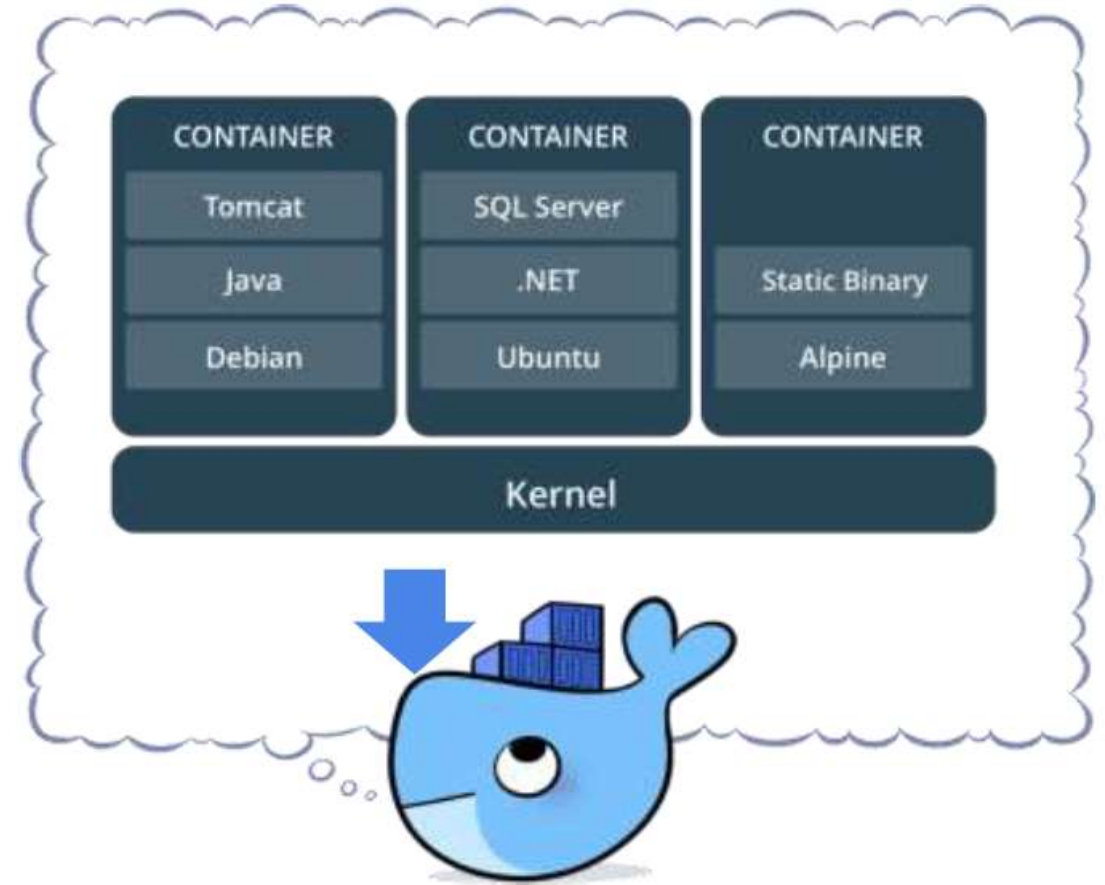
- We can package an application and its dependencies in a VM
- Don't VM's solve the dependency problem?
 - ▣ Well, yes, but...



Containers

8

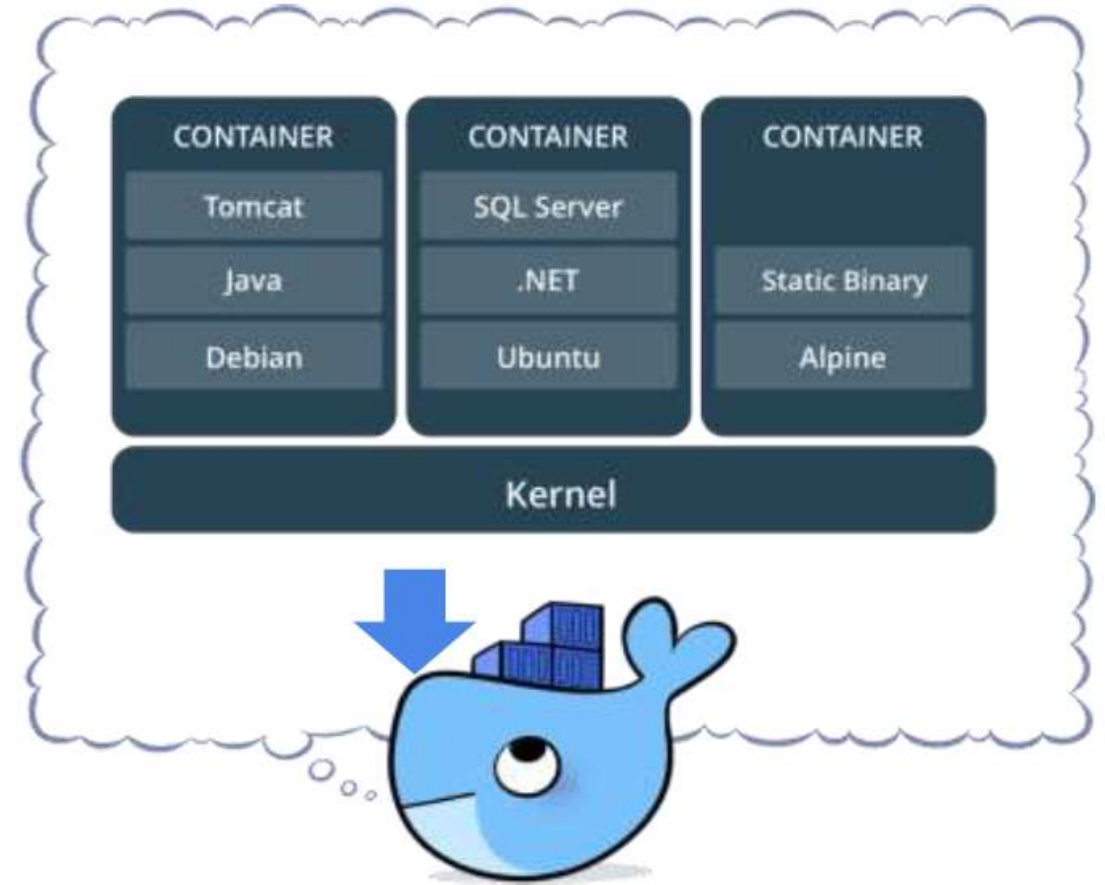
- A container is a package containing an application with its dependencies
- A single host OS can run multiple containers...
 - ▣ ... just like it can run multiple applications ...



Containers

9

- 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)

10

- Both technologies:
 - ▣ Run on a host OS (Windows / Linux)
 - ▣ Provide a virtualized computing environment for applications
 - Virtual OS, filesystem, RAM, CPU
 - ▣ Can be started and stopped
 - ▣ Admins can login and interactively manipulate environment
 - ▣ Applications can access network

Containers vs. Virtual Machines (Differences)

11

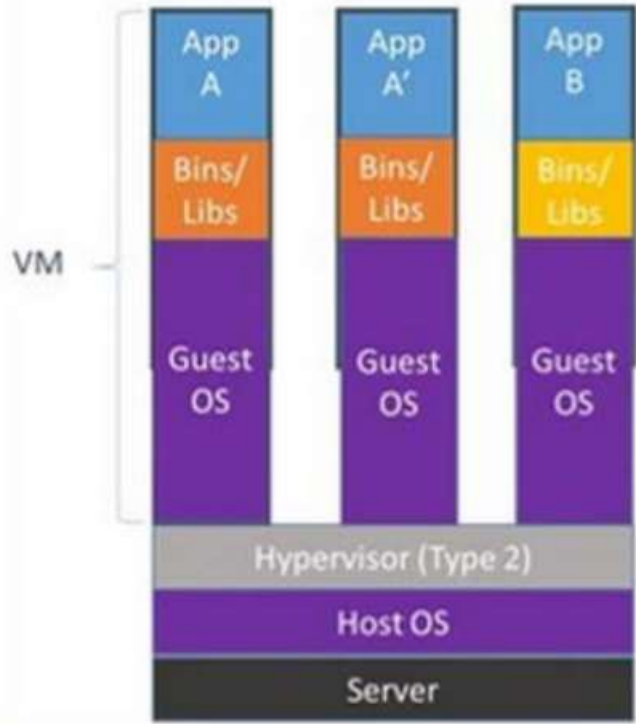
Virtual Machine

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

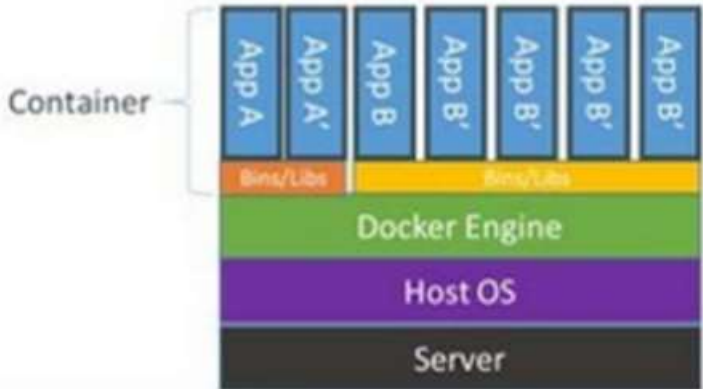
Container

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

Containers vs. VM's



Containers are isolated, but share OS and, where appropriate, bins/libraries



Containers vs. Virtual Machines

13

- 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

14

Docker Concepts

Key Docker Terms

15

- Image
 - ▣ Basis of a Docker container. A virtual filesystem with 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

16

□ 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

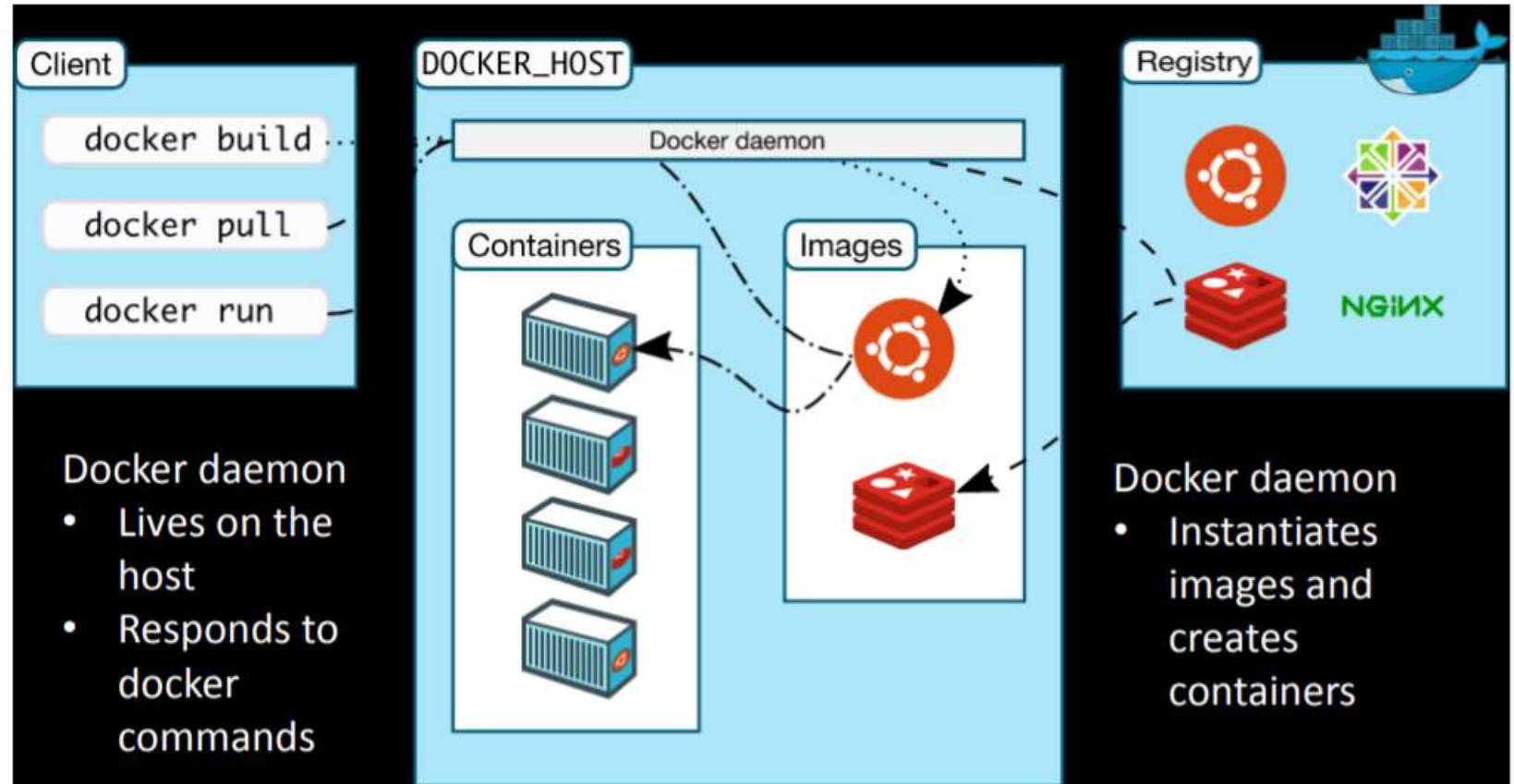
□ Container

- ▣ 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

17

- Image instantiated to form container



Docker Registry

18

- Contains a library of predefined Images
- Examples:
 - ▣ hub.docker.com
 - ▣ Amazon ECR
 - ▣ Private registries
- 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 to deploy to servers or other developers

19

Using Docker

Using Docker

20

- Installing Docker
 - ▣ Linux: apt get
 - ▣ Windows: Docker Desktop
- Interacting with Docker
 - ▣ Command Line
 - ▣ Docker Desktop GUI
 - ▣ Visual Studio Code

Running Docker Containers

21

□ Run a command in a container

□ docker run `ubuntu:22.04` `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

Interactive Containers

22

- Run an interactive shell in a container
 - ▣ `docker run -it ubuntu:22.04 bash`

What happens:

- ▣ Docker engine starts a new container using `ubuntu:22.04` image
- ▣ Docker engine starts bash shell in container
 - `-it` option ("interactive terminal") allows container bash shell to interact with user terminal
- ▣ 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

23

- 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:22.04 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

24

- 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

25

- 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** (run in foreground)
 - **docker compose up -d** (run in background)

docker-compose.yml

26

- 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

27

- 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`
- View logs:
 - ▣ `docker compose logs -f`
- 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"
```

28

Creating Images

Layers

29

- 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

30

- 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

31

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

32

Demo Time

Installation

33

- 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)