Lecture 2: Containers

AC295

Advanced Practical Data Science

Pavlos Protopapas

Outline

Class organization

Group formation

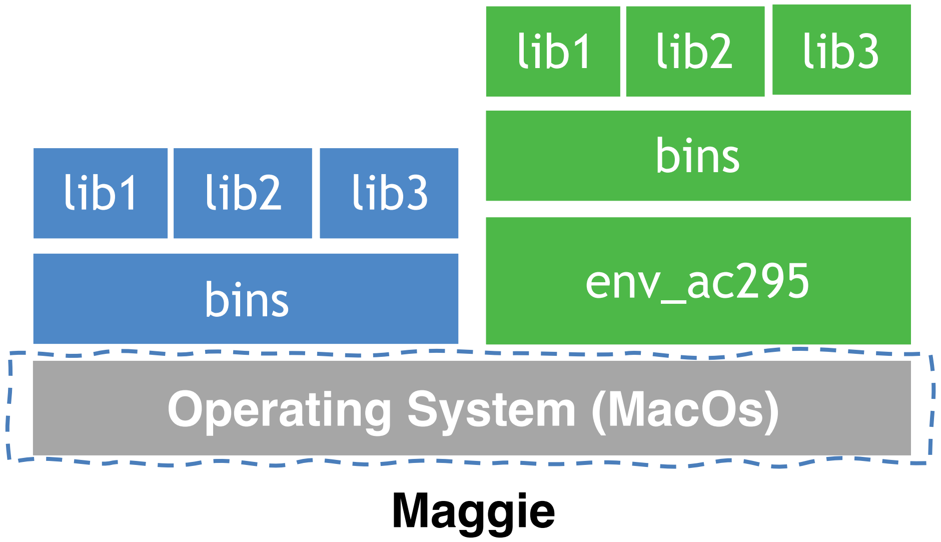
Presentation schedule

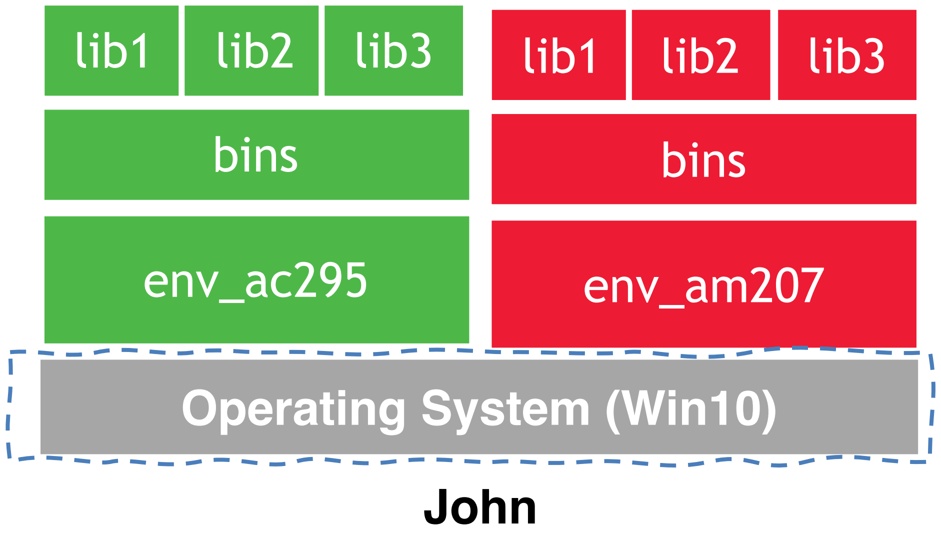
Review class flow

Outline

Why should we use virtual environment?

- What could go wrong? Unfortunately, Maggie and John reproduce different results and they think the issue relates to their operating systems. Indeed while Maggie has a MacOS, John uses a Win10.





Virtual environments

Pros

- Reproducible research
- Explicit dependencies
- Improved engineering collaboration
- Broader skill set

Cons

- Difficulty setting up your environment
- Not isolation
- Does not work across different OS

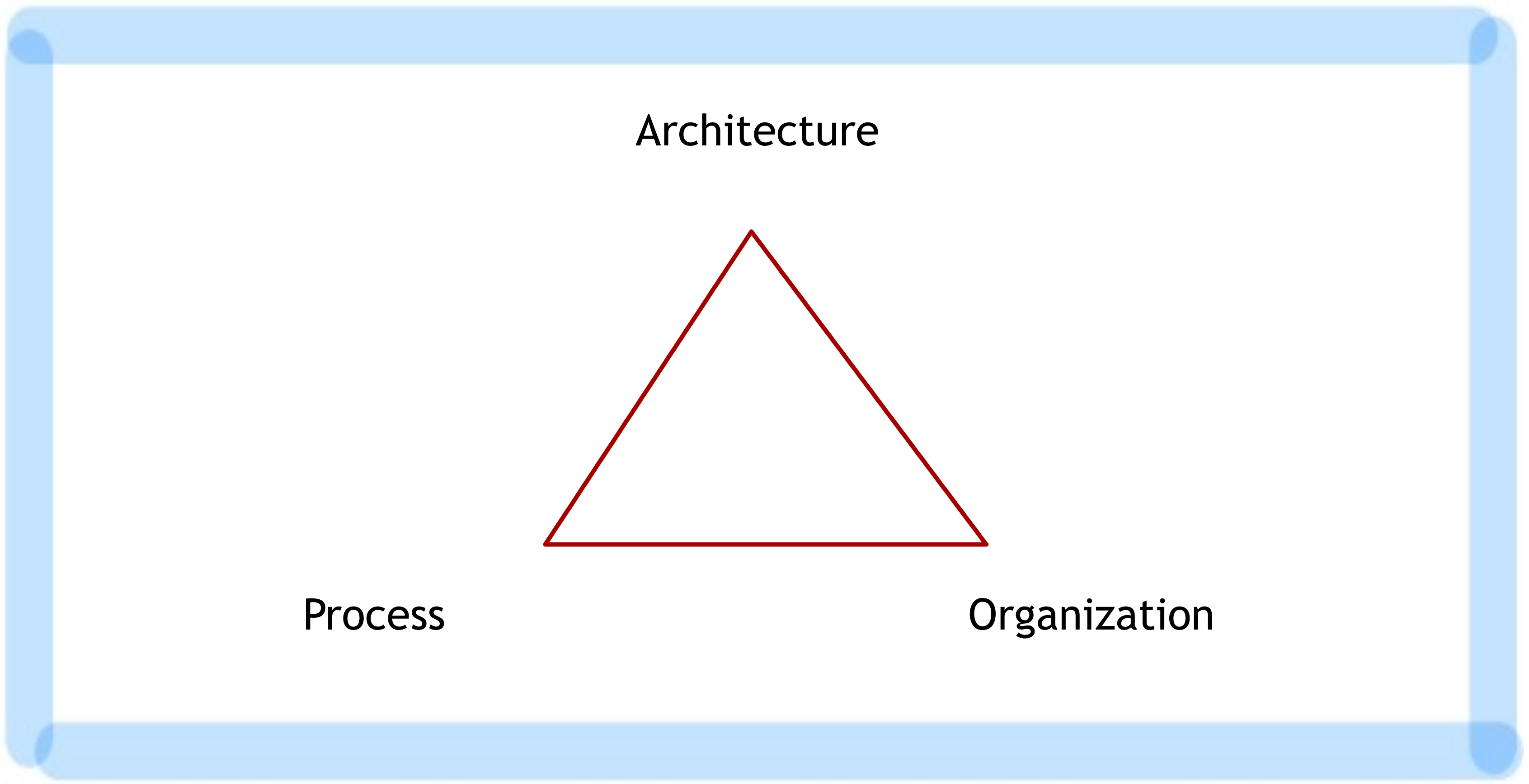
Virtual Machines Limitations

- Uses hardware in your local machine (cannot run more than two on an average laptop)
- There is overhead associated with virtual machines
    1. guest is not as fast as the host system
    2. Takes long time to start up
    3. may not have the same graphics capabilities

Outline

Successful Software Application

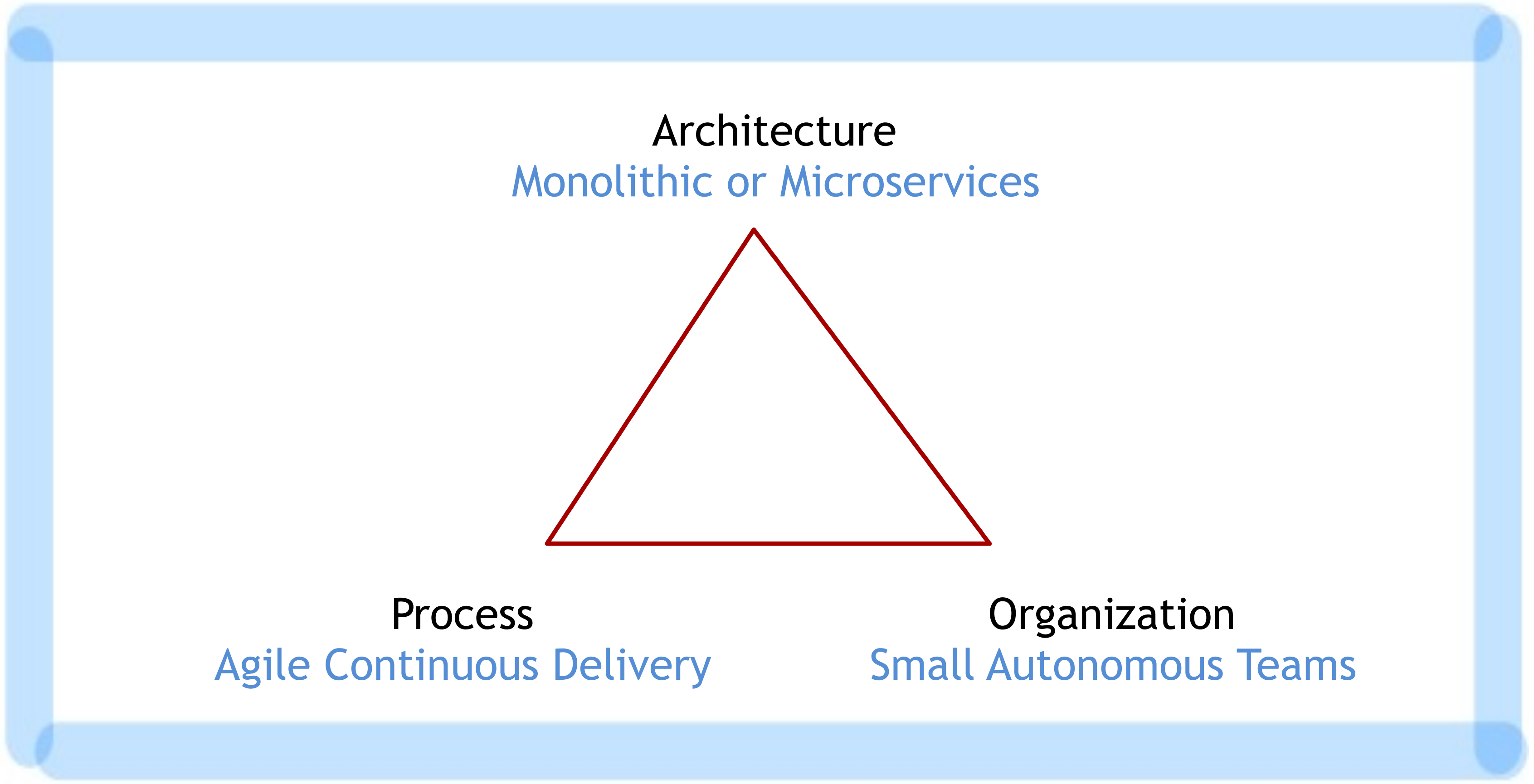
- Imagine you are building a large complex application (e.g. Online Store)



Slide is tare taken from Ajeet Singh Raine

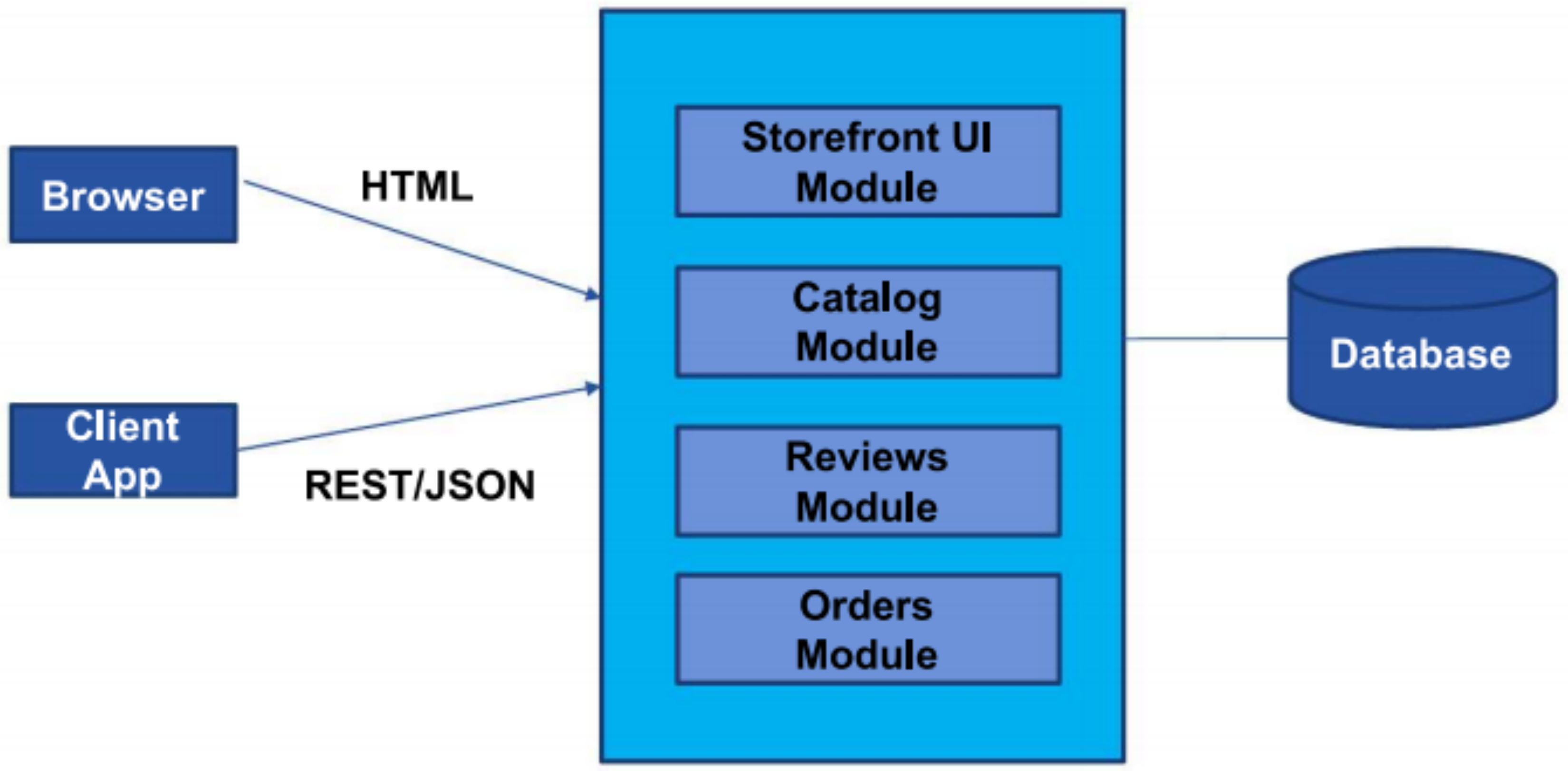
Successful Software Application

- Imagine you are building a large complex application (e.g. Online Store)



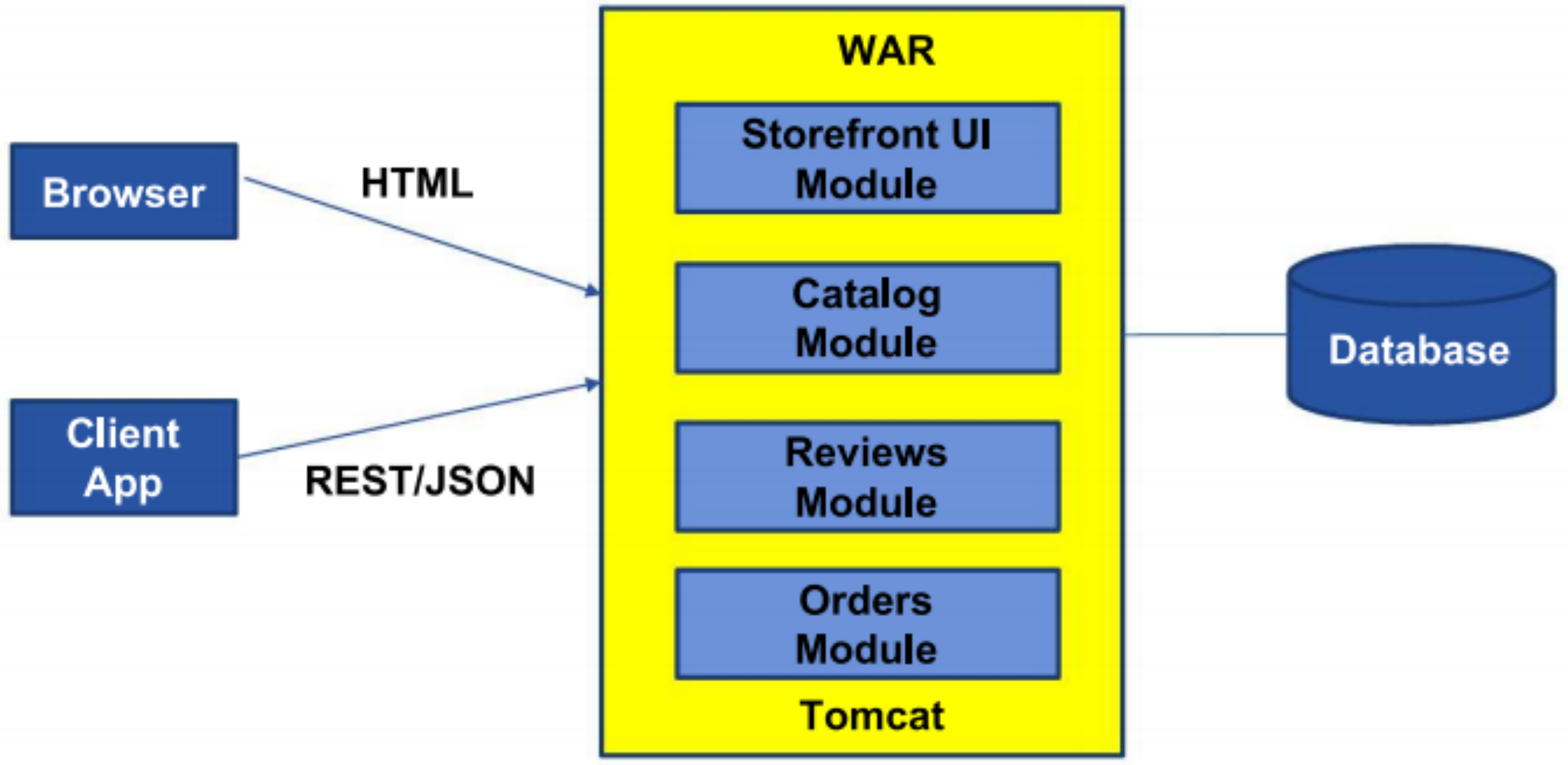
Slide is tare taken from Ajeet Singh Raine

Monolithic Architecture



Slide is tare taken from Ajeet Singh Raine

Monolithic Architecture



Slide is tare taken from Ajeet Singh Raine

I finally remember what Zoom meetings remind me of.

Benefits of Monolith

Simple to **Develop**, **Test**, **Deploy** and **Scale**:

1. Simple to develop because all the tools and IDEs support the applications by default
2. Easy to deploy because all components are packed into one bundle
3. Easy to scale the whole application

Disadvantages of Monolith

1. Very difficult to maintain
2. One component failure will cause the whole system to fail
3. Very difficult to create the patches for monolithic architecture
4. Adapting to new technologies is challenging
5. Take a long time to startup because all the components needs to get started

Applications have changed drammatically

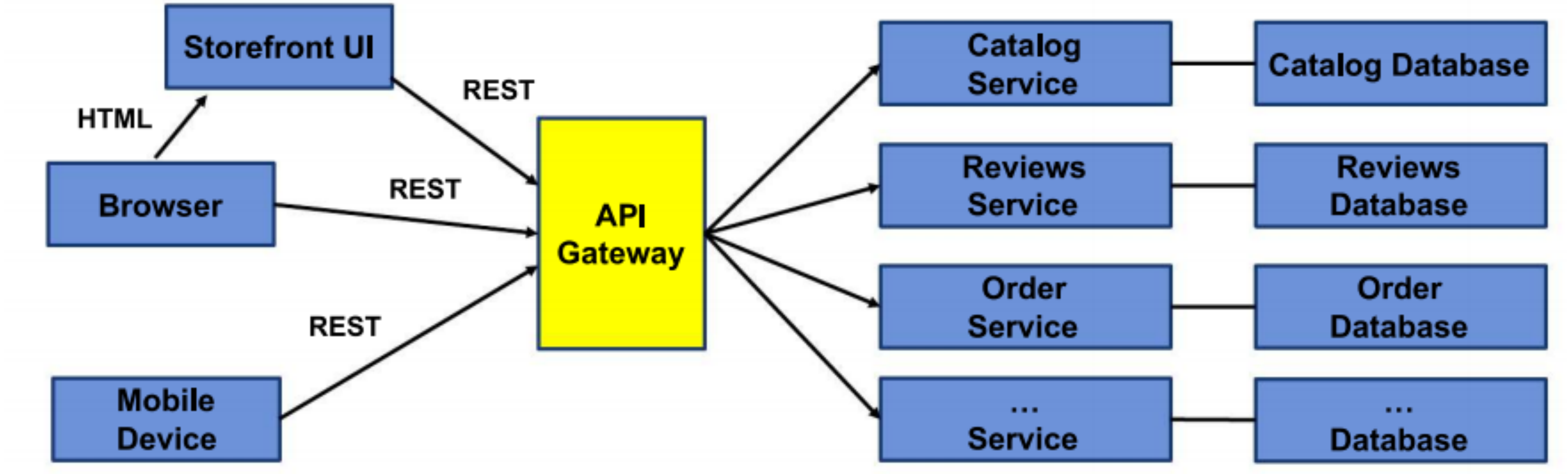
A decade ago

Apps were monolithic
Built on a single stack (e.e. .NET or Java)
Long lived
Deployed to a single server

Today

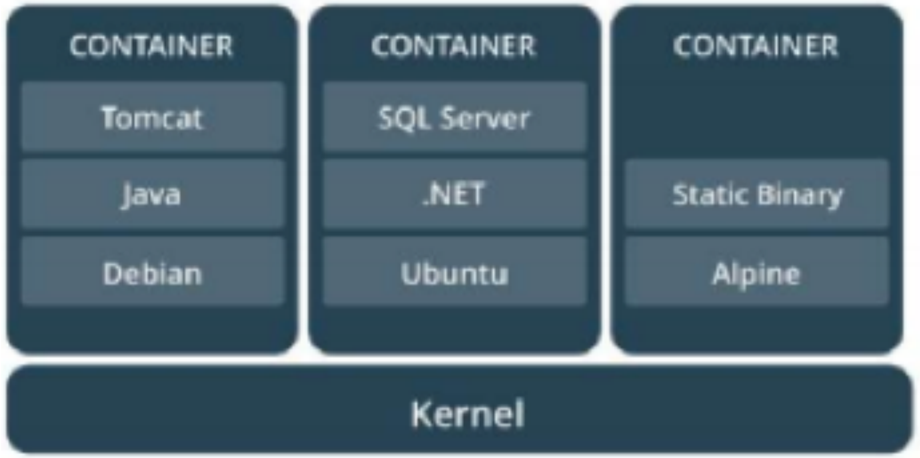
Apps are constantly being developed
Build from loosely coupled components
Newer version are deployed often
Deployed to a multitude of servers

Microservice Architecture



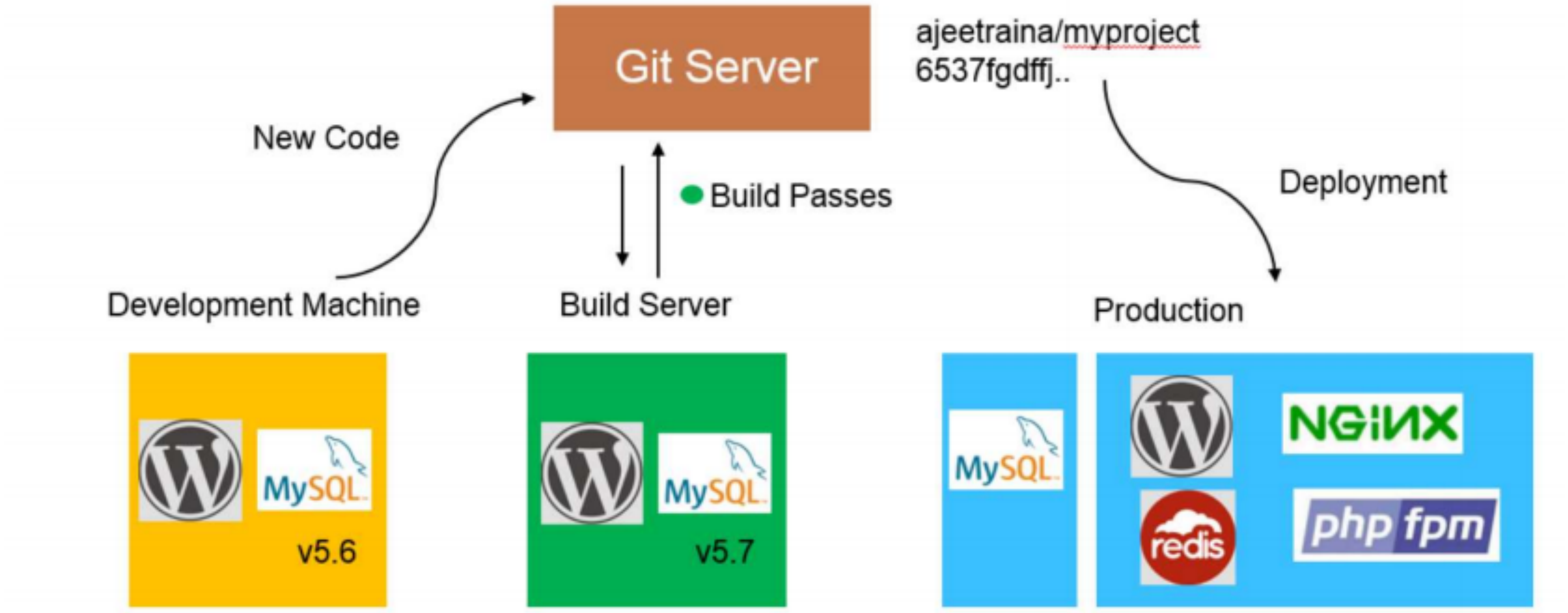
Outline

What is docker



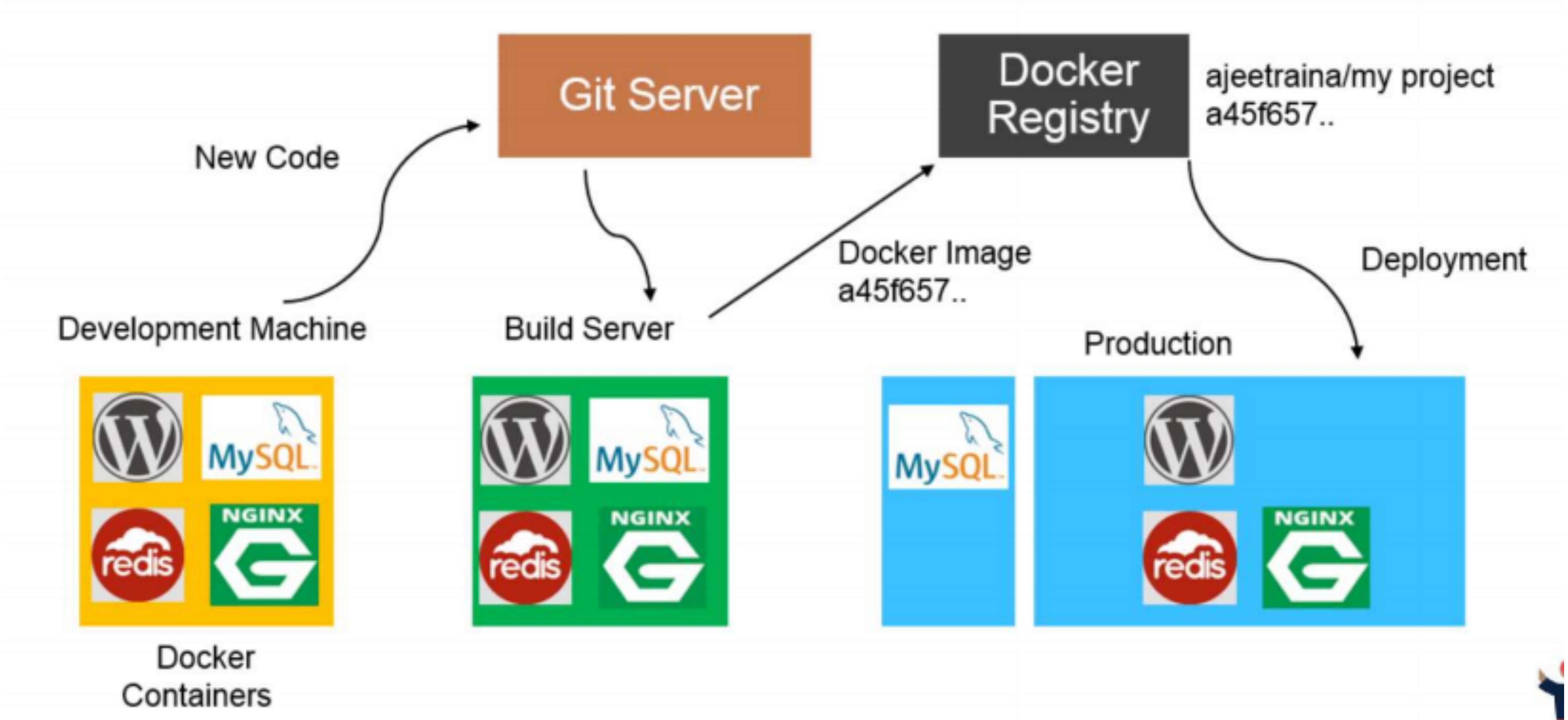
- Standardized packaging for software dependencies
- Isolate apps from each other
- Works for all major Linux distributions, MacOS, Windows

Traditional Software Development Workflow (without Docker)



Slide is tare taken from Ajeet Singh Raine

Traditional Software Development Workflow (with Docker)



Slide is tare taken from Ajeet Singh Raine

Docker Containers are not Virtual Machines

Virtual Machines

Containers

Docker Container vs Virtual Machines (VM)

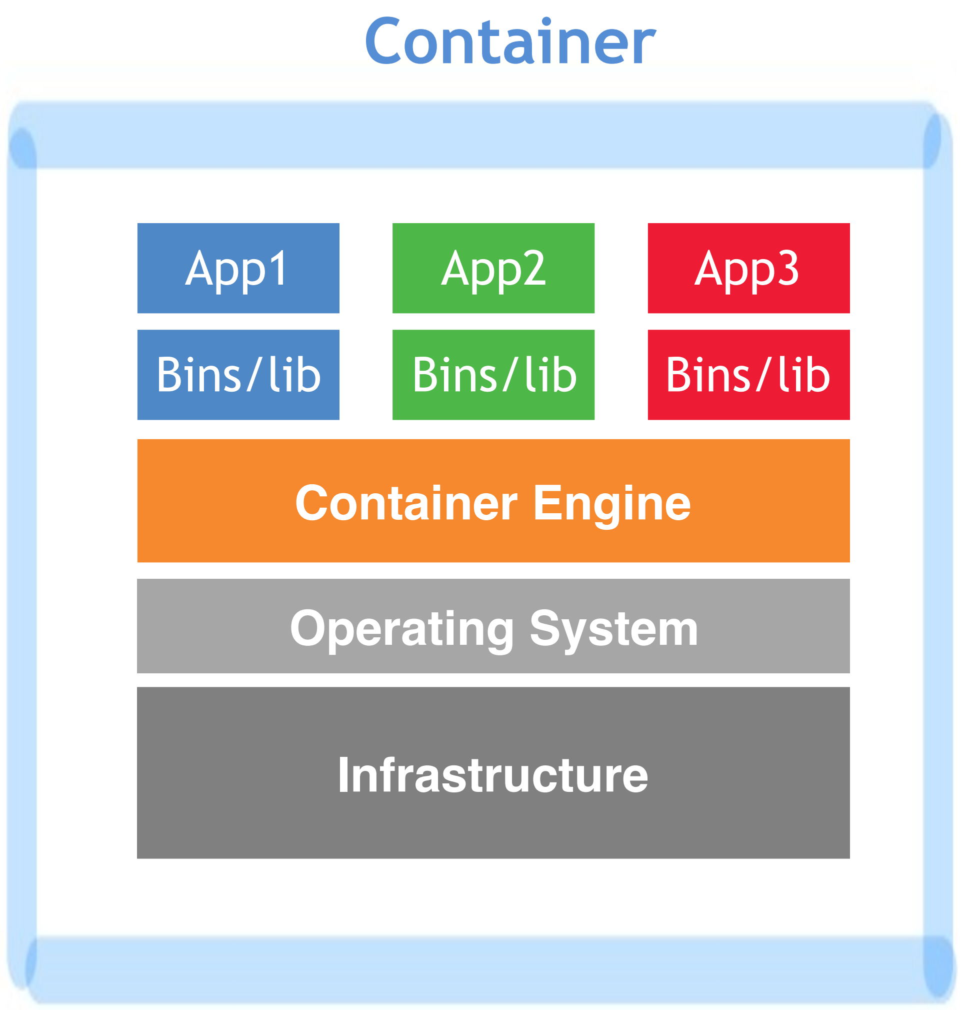
VMs

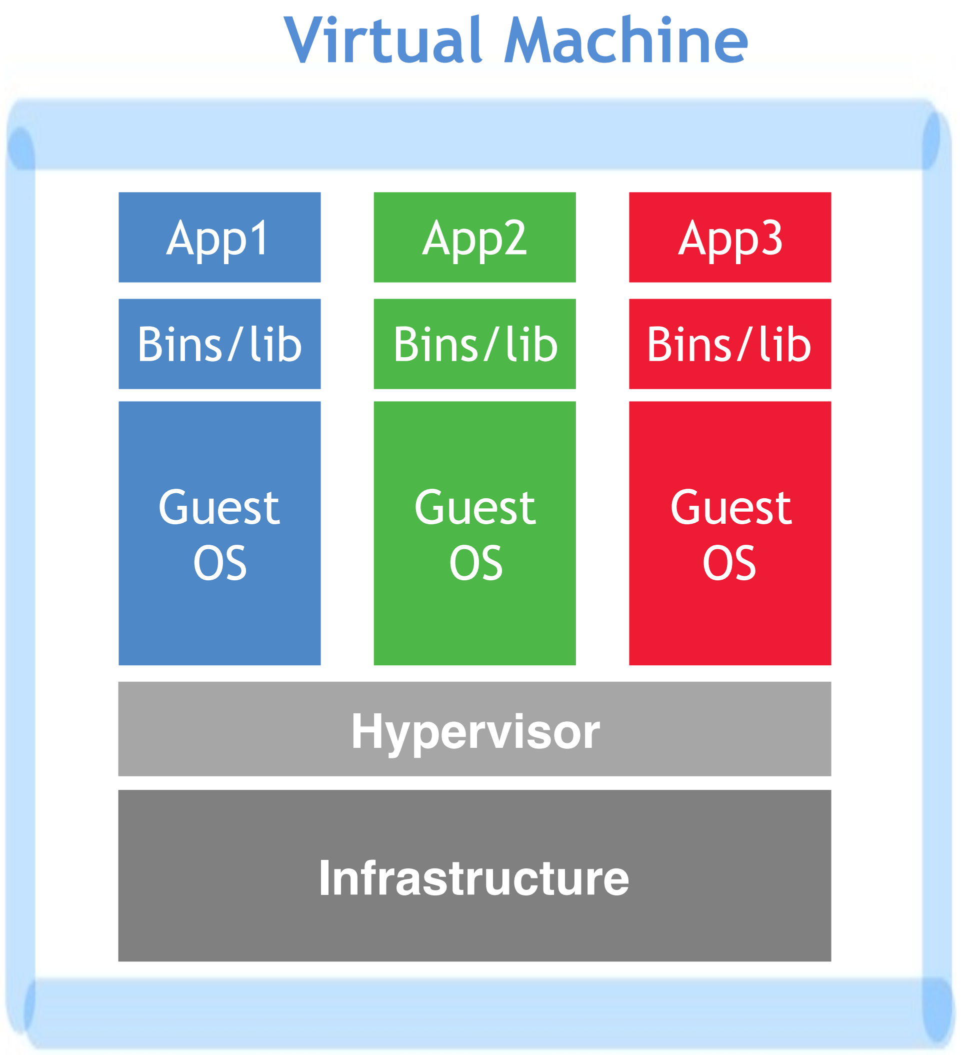
- Each VM runs its own OS
- Boot up time is in minutes
- Not version controlled
- Cannot run more than couple of VMs on an average laptop
- Only one VM can be started from one set of VMX and VMDK files

Docker

- Container is just a user space of OS
- Containers instantiate in seconds
- Images are built incrementally on top of another like layers. Lots of images/ snapshots
- Images can be diffed and can be version controlled. Docker hub is like Github
- Can run many Dockers in a laptop
- Multiple docker containers can be started from one Docker image

Docker Container vs Virtual Machines





What Makes Containers so Small?

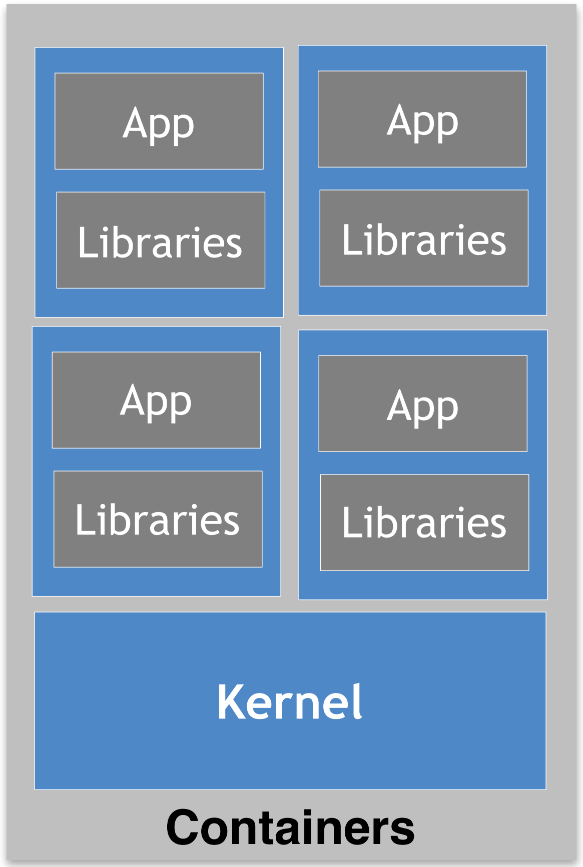
Container = User Space of OS

- User space refers to all of the code in an operating system that lives outside of the kernel

Why should we use containers?

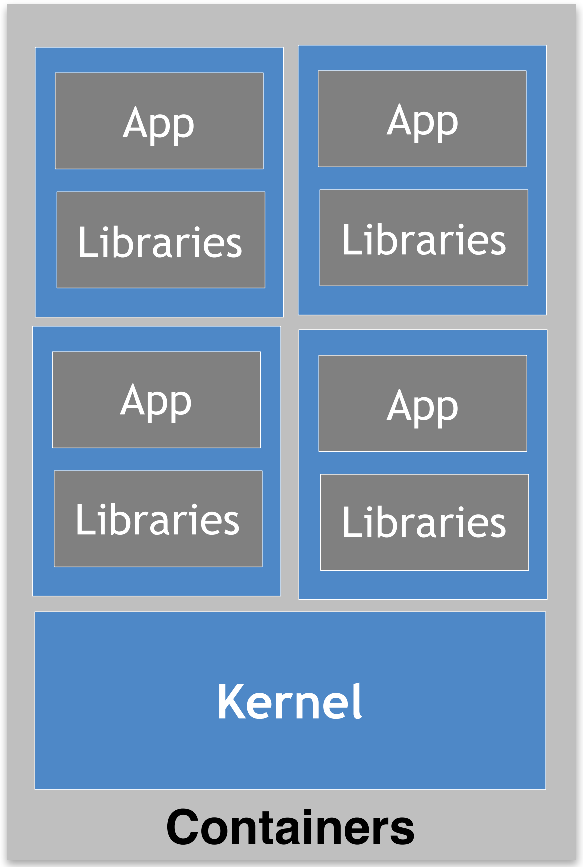
- It has the best of the two worlds because it allows:
  1. to create isolate environment using the preferred operating system
  2. to run different operating system without sharing hardware
- The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.

- Containers give the impression of a separate operating system however, since they're sharing the kernel, they are much cheaper than a virtual machine.



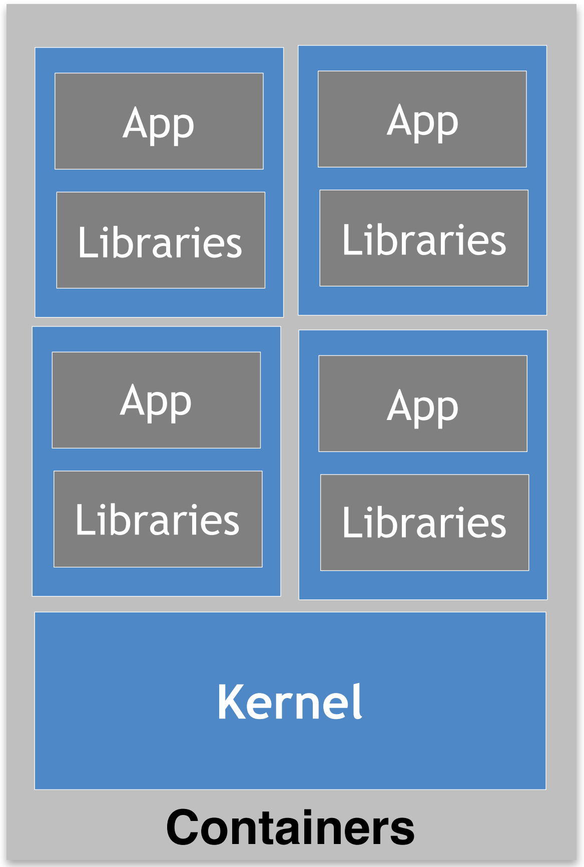
Why should we use containers? (cont)

- With container images, we confine the application code, its runtime, and all its dependencies in a pre-defined format.
- With the same image, you can reproduce as many containers as you wish. Think about the image as the recipe, and the container as the cake ;-) you can make as many cakes as you'd like with a given recipe.
- A container orchestrator (see next lecture) is a single controller/ management unit that connects multiple nodes together.
- You can create a container on a Window but install an image of a Linux OS inside that container. The container still works on the Window



Why should we use containers? (cont)

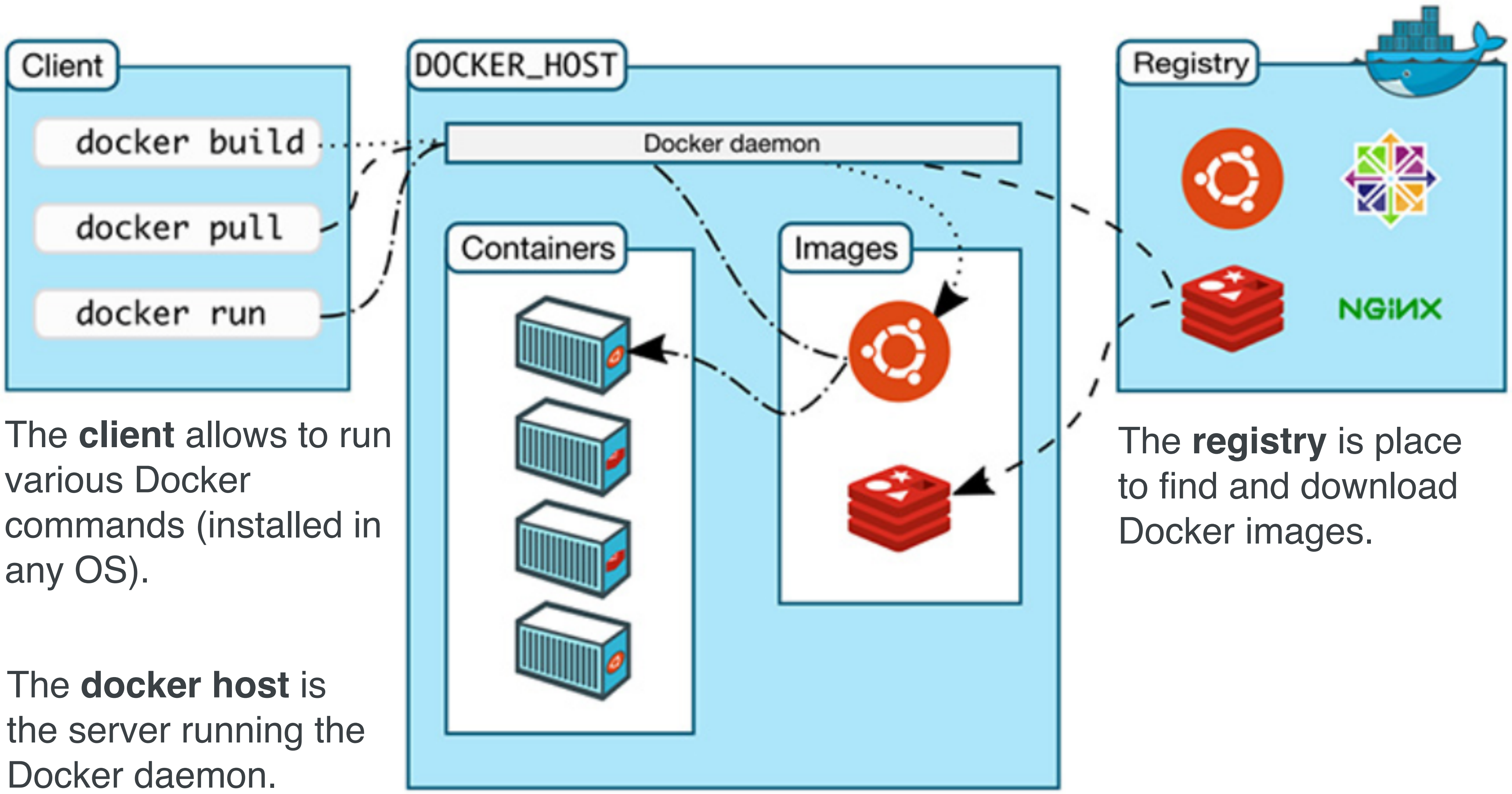
- Containers are application-centric methods to deliver high-performing, scalable applications on any infrastructure of your choice.
- Containers are best suited to deliver **microservices** by providing portable, isolated virtual environments for applications to run without interference from other running applications.
- Containers run container images, it bundles the application along with its runtime and dependencies.
- Because they're so lightweight, you can have many containers running at once on your system.



Why containers recap?

- Their startup time is on the order of seconds (vs. minutes for Virtual Machines).
- They provide pseudo-isolation. This means they're still pretty secure, but not as secure at a Virtual Machines.
- A container is deployed from the container image offering an isolated executable environment for the application.
- Containers can be deployed from a specific image on many platforms, such as workstations, Virtual Machines, public cloud, etc.
- Containers are extremely popular, and their popularity is growing.
- One of the first widely used containers was provided by Docker.
- Docker containers can be used to run websites and web applications.
- Multiple containers can be managed by a service called Kubernetes (see next lecture)

The Docker Engine Architecture



The **client** allows to run various Docker commands (installed in any OS).

The **docker host** is the server running the Docker daemon.

The **registry** is place to find and download Docker images.

https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together

Some Docker Vocabulary

Docker Image
The basis of a Docker container. Represent a full application

Docker Container
The standard unit in which the application service resides and executes

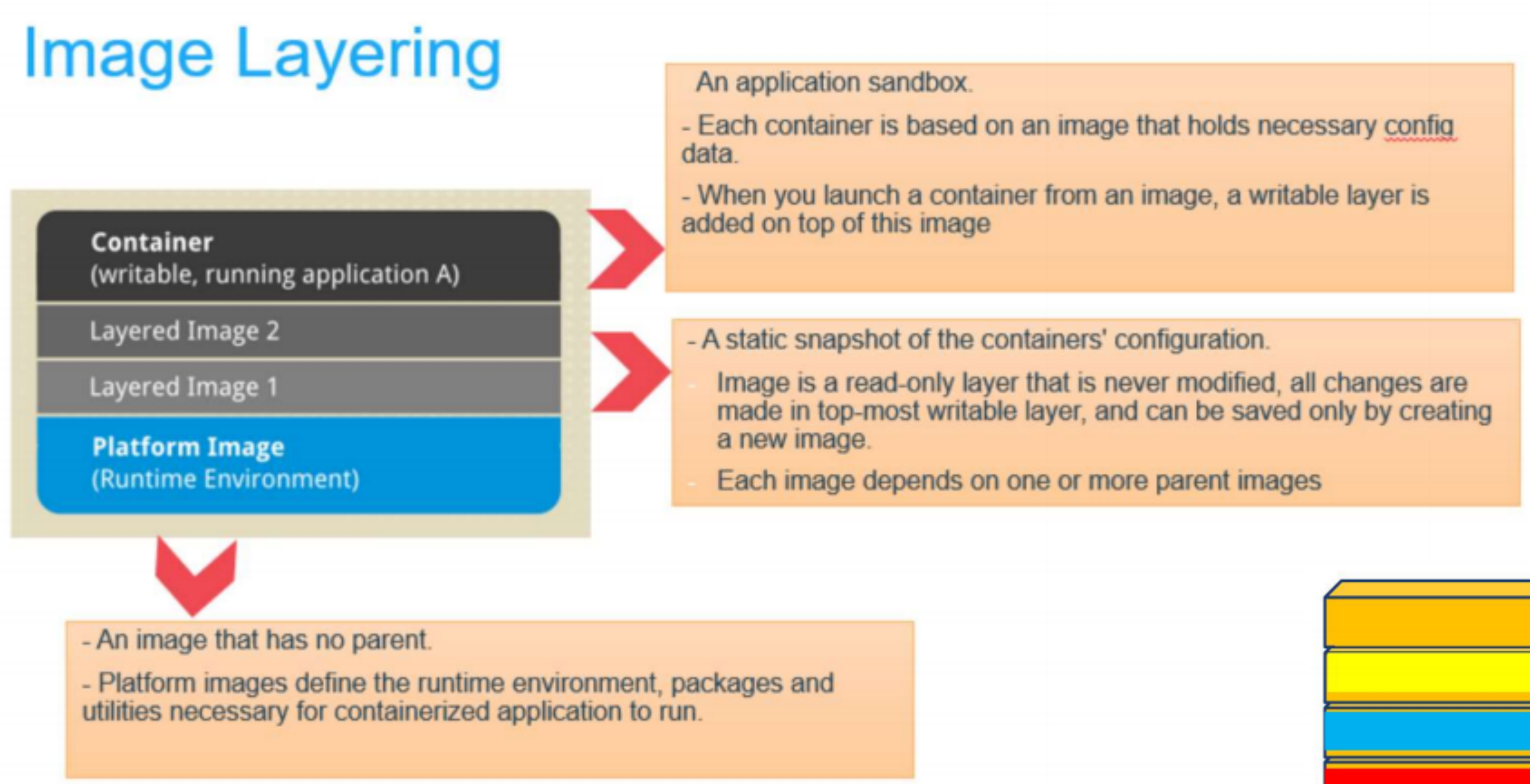
Docker Engine
Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub or Docker Trusted Registry)
Cloud or server-based storage and distribution service for your images

Images
How you **store** your application

Containers
How you **run** your application

Image Layering



An application sandbox.

- Each container is based on an image that holds necessary config data.

- When you launch a container from an image, a writable layer is added on top of this image

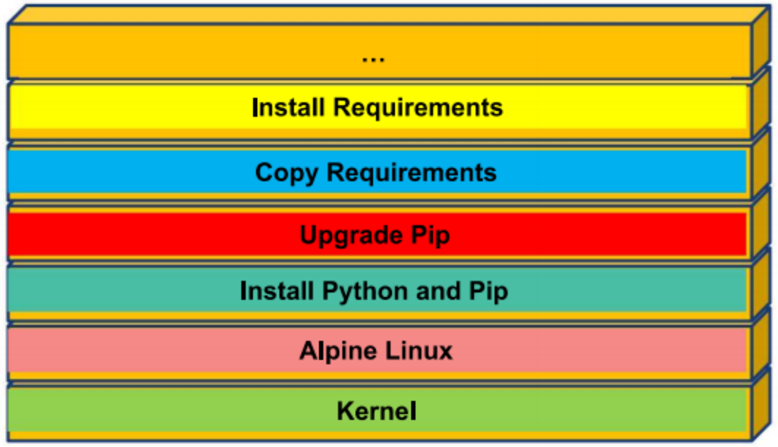
- A static snapshot of the containers' configuration.

- Image is a read-only layer that is never modified, all changes are made in top-most writable layer, and can be saved only by creating a new image.

- Each image depends on one or more parent images

- An image that has no parent.

- Platform images define the runtime environment, packages and utilities necessary for containerized application to run.



Outline

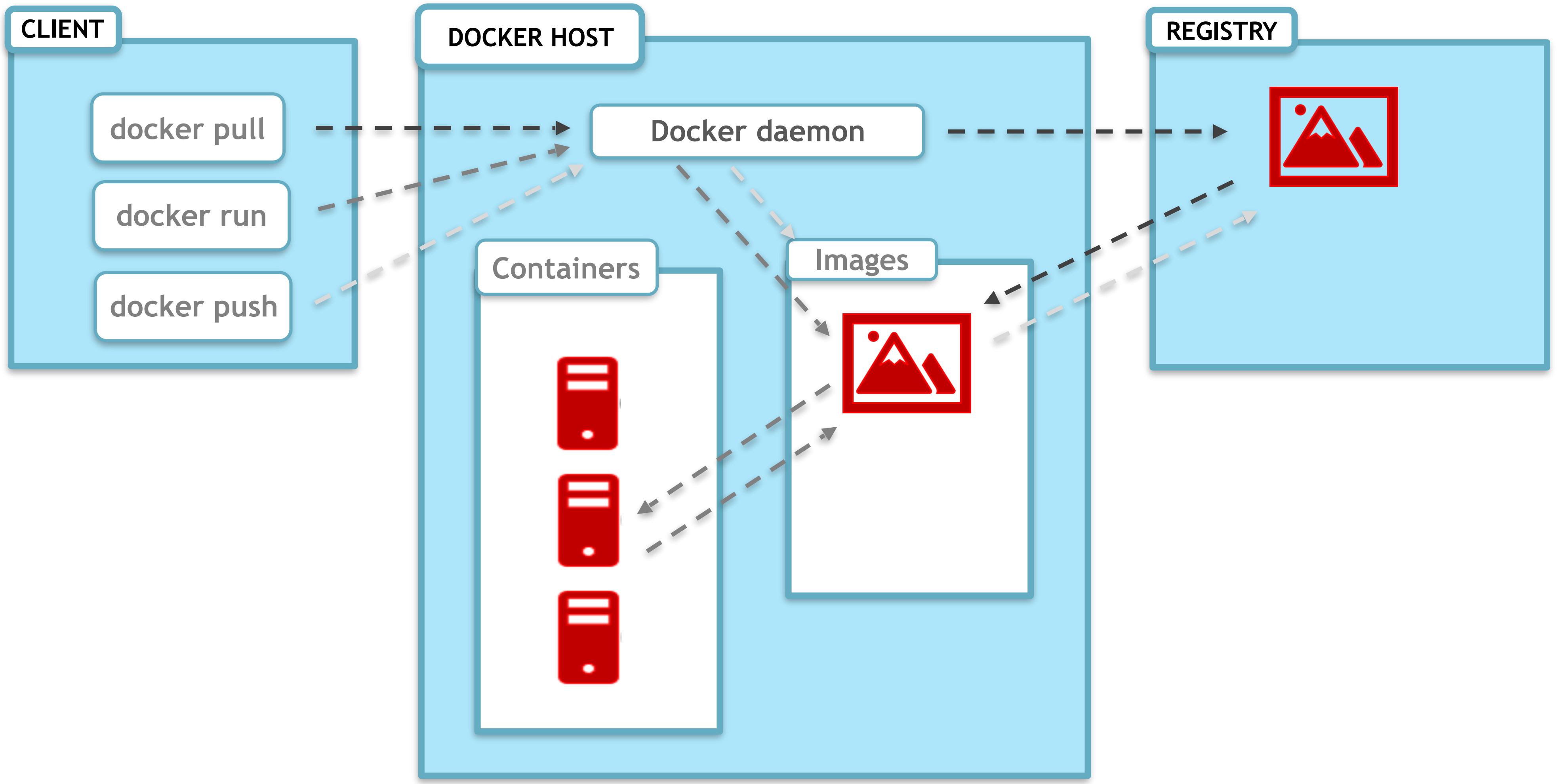
Hands on Containers

Exercise 3; For you to play

Exercise 1: Pull, modify, and push a Docker image from DockerHub

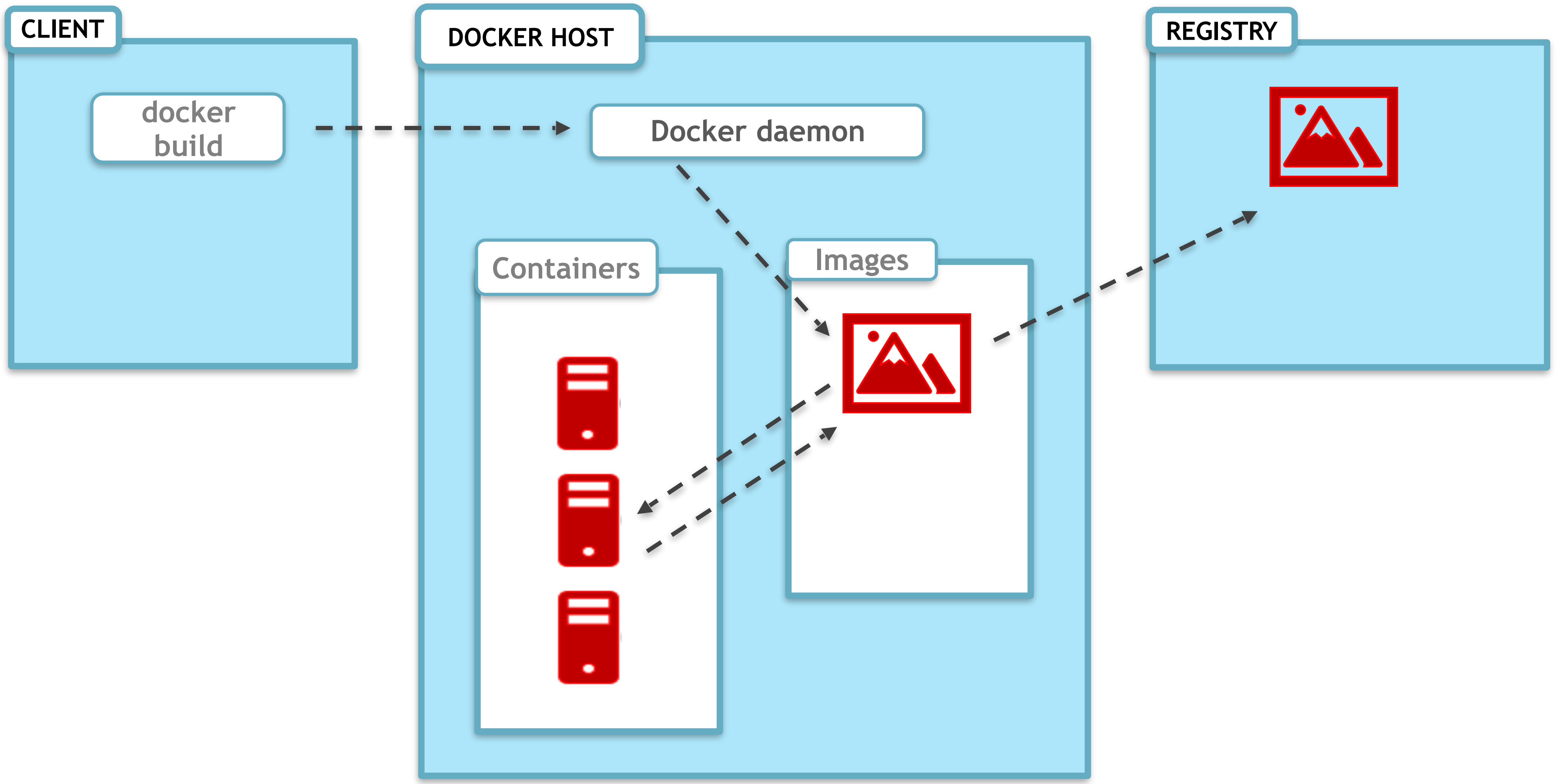
Exercise 2: Build a Docker Image and push it to DockerHub

Exercise 1: pull, modify, and push an image



https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together

Exercise 2: build a Docker Image



https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together

Hands on Containers I Instructions

Exercise set up

- install docker (https://hub.docker.com/)
- have docker up and running
- create a class repository in docker hub (yourhubusername/ac295_playground)

Exercise 1: modify images from Docker Hub

- Step 1 I pull image pavlosprotopapas/ac295_l2:latest
- Step 2 I run container in interactive mode (-it)
- Step 3 I open Readme.txt file and follow instructions to modify image
- Step 4 I push modified image (pavlosprotopapas/ac295_l2)

Exercise 2: Docker Do It Yourself

- Step 1 I pull course repository and cd to lecture2/exercises/exercise1
- Step 2 I build an image using Dockerfile (for MacOs)
- Step 3 I push image to docker hub (yourhubusername/ac295_playground)

THANK YOU