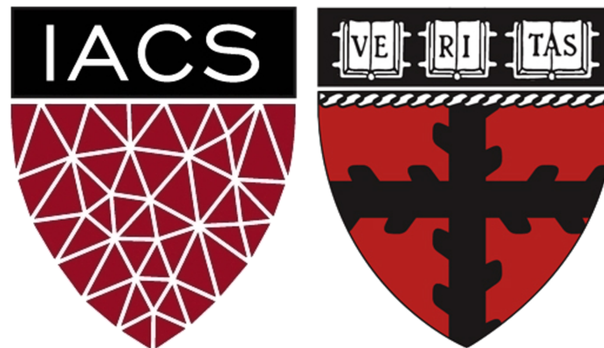
Environments, Virtual Machines and Containers

Guest Lecture for CS107/AC207

Pavlos Protopapas



Outline

1: Virtual environments

2: Virtual machines

3: Containers

4: Demo

Outline

1: Virtual environments

2: Virtual machines

3: Containers

4: Demo

Why should we use virtual environment?

- Virtual environments help to make development and use of code more streamlined.
- Virtual environments keep dependencies in separate “sandboxes” so you can switch between both applications easily and get them running.
- Given an operating system and hardware, we can get the exact code environment set up using different technologies. This is key to understand the trade off among the different technologies presented in this class.

Why should we use virtual environment?

- Maggie took cs109a, she used to run her Jupyter notebooks from anaconda prompt. Every time she installed a module it was placed in the either of `bin, lib, share, include` folders and she could import it in and used it without any issue.

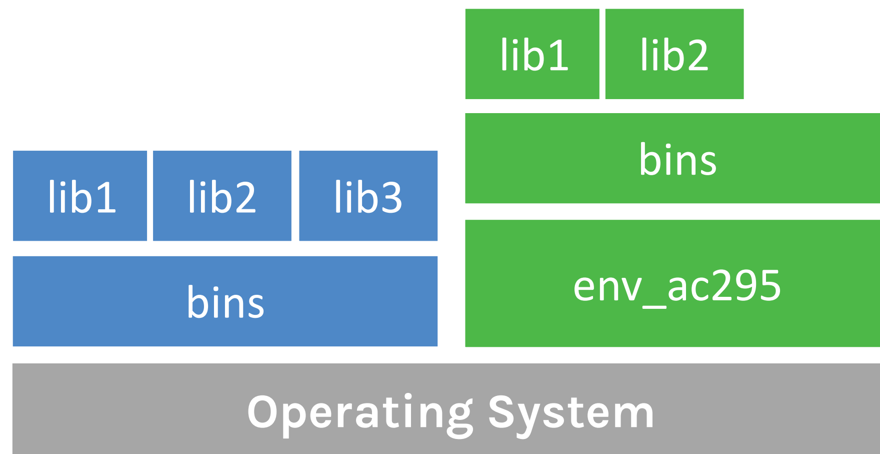
```
$ which python
/c/Users/maggie/Anaconda3/python
```

Maggie

IACS

Why should we use virtual environment?

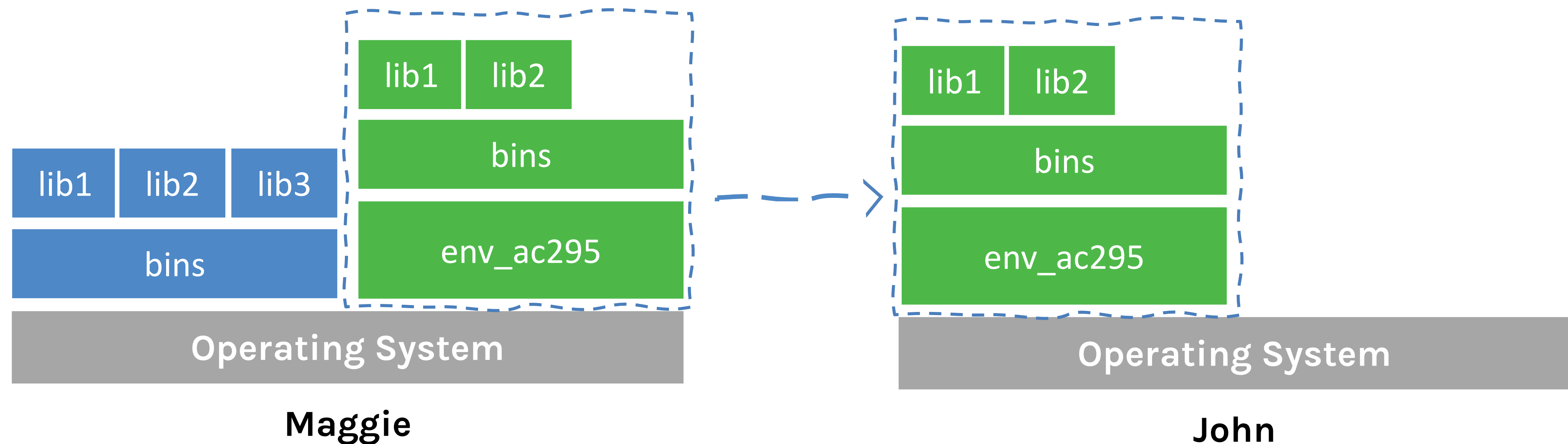
- Maggie starts taking ac295 and she thinks that would be good to isolate the new environment from the previous environments avoiding any conflict with the installed packages. She adds a layer of abstraction called virtual environment that helps her keep the modules organized and avoid misbehaviors while developing a new project.




```
$ which python
/c/Users/maggie/Anaconda3/envs/env_ac295/python
```

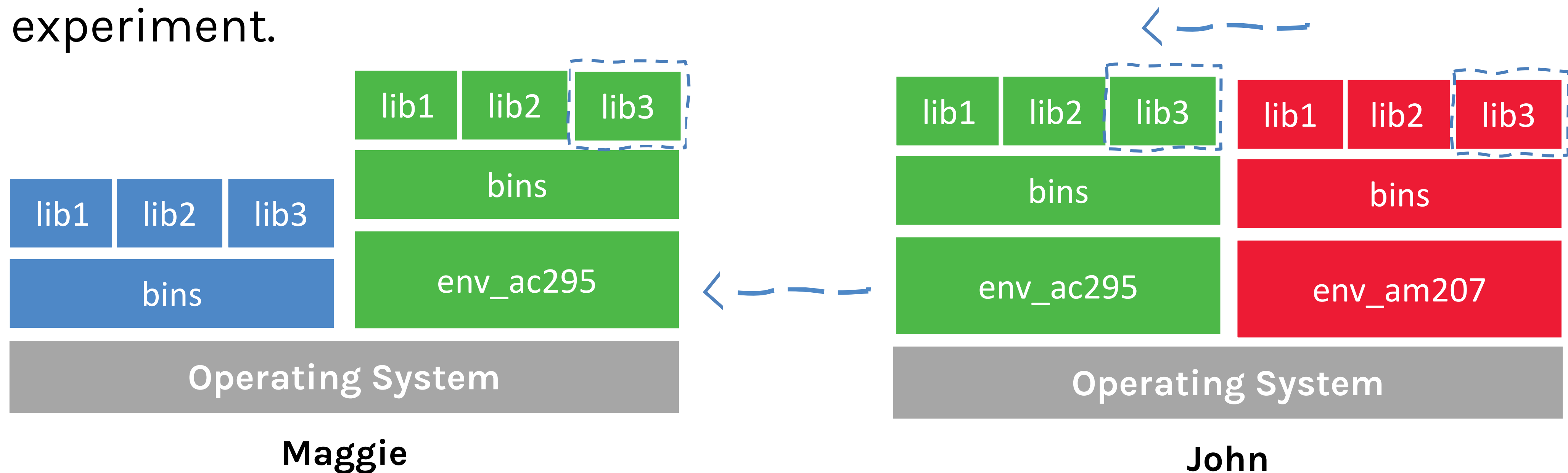
Why should we use virtual environment?

- Maggie collaborates with John for the final project and shares with him the environment she is working on through .yml file.



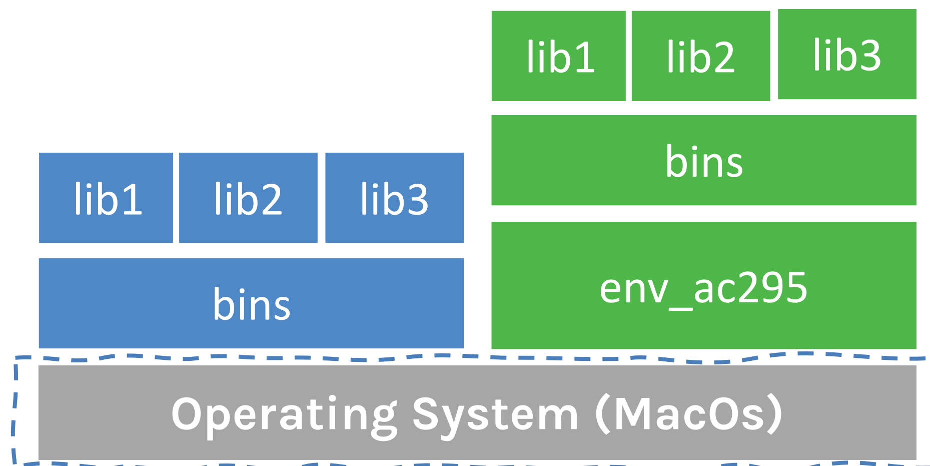
Why should we use virtual environment?

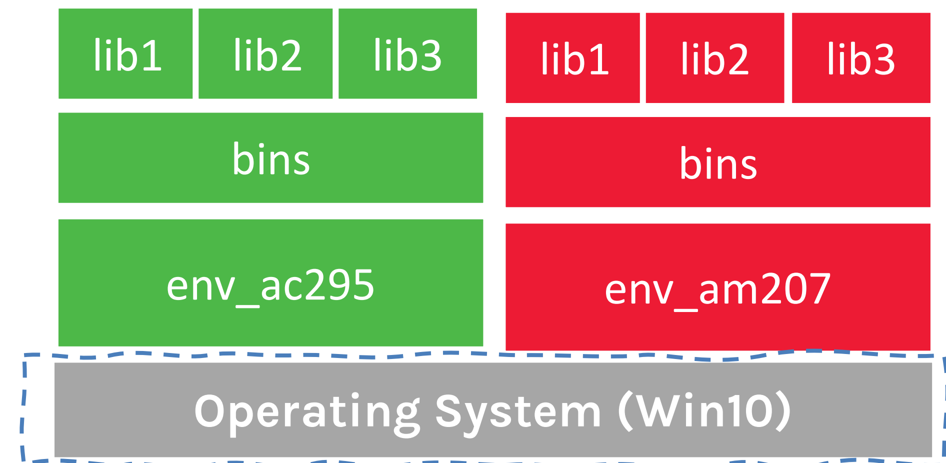
- John experiments with a new method he learned in another class and adds a new library to the working environment. After seeing a tremendous improvements he sends Maggie back his code and a new .yml file. She can now update her environment and replicate the experiment.



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.





IACS

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

What are virtual environments then?

A virtual environment is a directory with the following components:

- site_packages/ directory where third-party libraries are installed
- links [really symlinks] to the executables on your system
- some scripts that ensure that the code uses the interpreter and site packages in the virtual environment

Virtual environments: virtualenv vs conda

virtualenv

- virtual environments manager embedded in Python
- incorporated into broader tools such as `pipenv`
- allow to install modules using pip package manager

how to use **virtualenv**

- create an environment within your project folder `virtualenv your_env_name`
- it will add a folder called environment_name in your project directory
- activate environment: `source env/bin/activate`
- install requirements using: `pip install package_name=version`
- deactivate environment once done: `deactivate`

Virtual environments in practice (virtualenv vs conda)

conda environment

- virtual environments manager embedded in Anaconda
- allow to use both conda and pip to manage and install packages

how to use **conda**

- create an environment `conda create --name your_env_name python=3.7`
- it will add a folder located within your anaconda installation `/Users/your_username/anaconda3/envs/your_env_name`
- activate environment `conda activate your_env_name` (should appear in your shell)
- install requirements using `conda install package_name=version`
- deactivate environment once done `conda deactivate`
- duplicate your environment using YAML file `conda env export > my_environment.yml`
- to recreate the environment now use `conda env create -f environment.yml`

More on Virtual environments

Further readings

- For detailed discussions on similarities and differences among virtualenv and conda
  https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/
- More on venv and conda environments
  https://towardsdatascience.com/virtual-environments-104c62d48c54
  https://towardsdatascience.com/getting-started-with-python-environments-using-conda-32e9f2779307

Outline

1: Virtual environments

2: Virtual machines

3: Containers

4: Demo

Why should we use virtual machines?

Motivation

- we have our isolated systems and after we set up a similar environment into our colleagues' machines we should get similar results, right? Unfortunately, it is not always the case. Why? Most likely because we run it on different operating system.
- even though by using virtual environments we are isolating our computations, we might need to use the same operating system which requires to run "like if" we are in a different machines.
- How can we run the same experiment? Virtual Machines!
- Isolation!

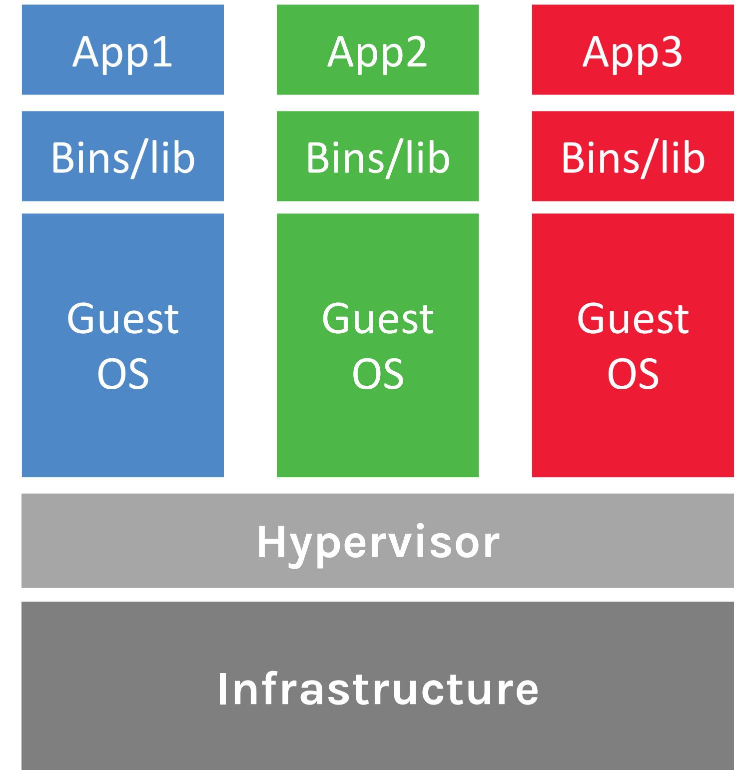
Why should we use virtual machines?(cont)

Advantages

- full autonomy: it works like a separate computer system, it is like run a computer within a computer.
- **very secure**: the software inside the virtual machines can't affect the actual computer.
- lower costs: buy one machine and run multiple operating systems.

What are virtual machines?

- virtual machines have their own virtual hardware: CPUs, memory, hard drives, etc.
- you need a hypervisor that manages different virtual machines on server
- hypervisor can run as many virtual machines as you wish
- operating system is called the "host" while those running in a virtual machine are called "guest“
- You can install a completely different operating system on this virtual machine



Machine Virtualization

https://towardsdatascience.com/how-to-install-a-free-windows-virtual-machine-on-your-mac-bf7cbc05888

Limitations

- Uses hardware in your local machine
- 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

1: Virtual environments

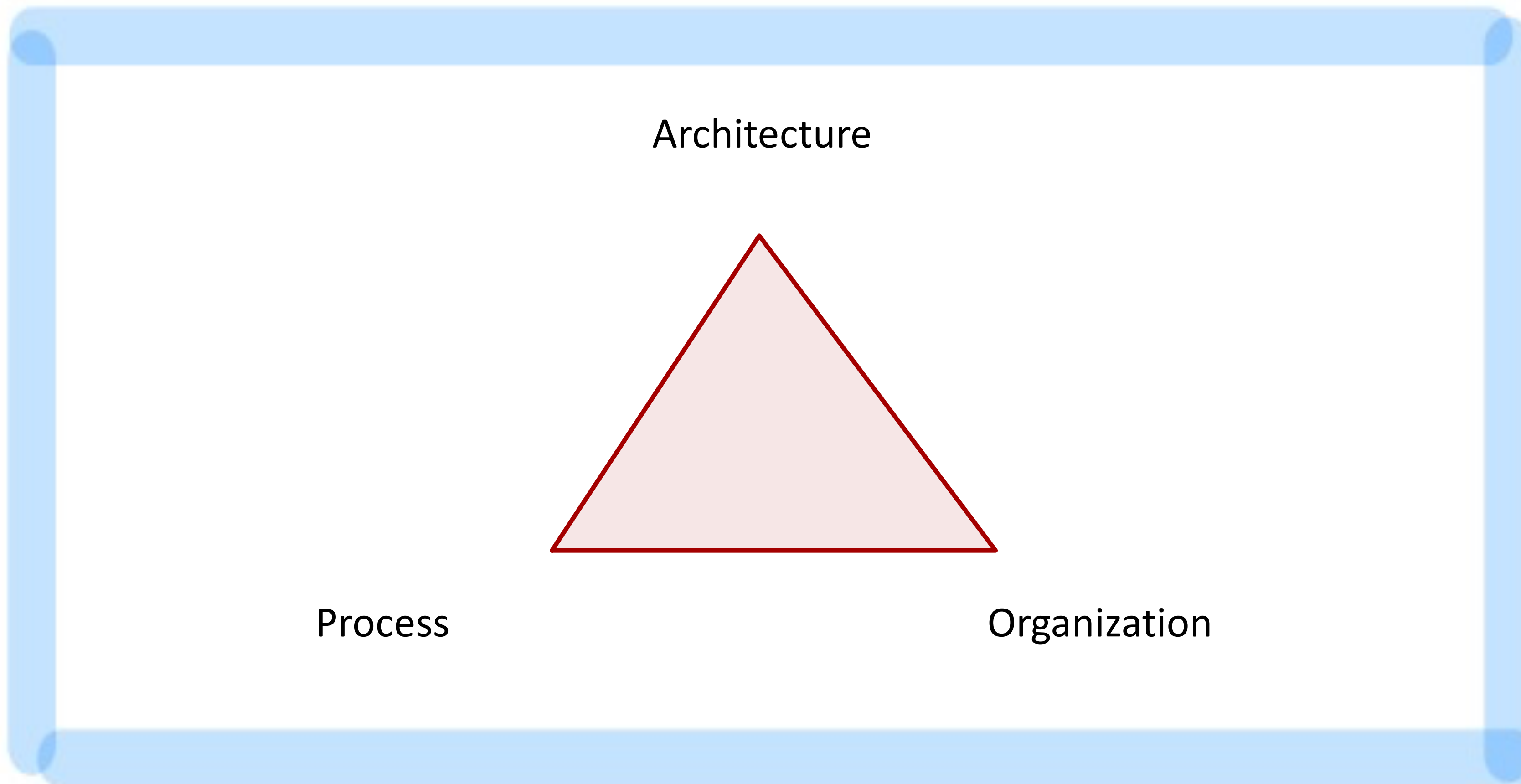
2: Virtual machines

3: Containers

4: Demo

Successful Software Application

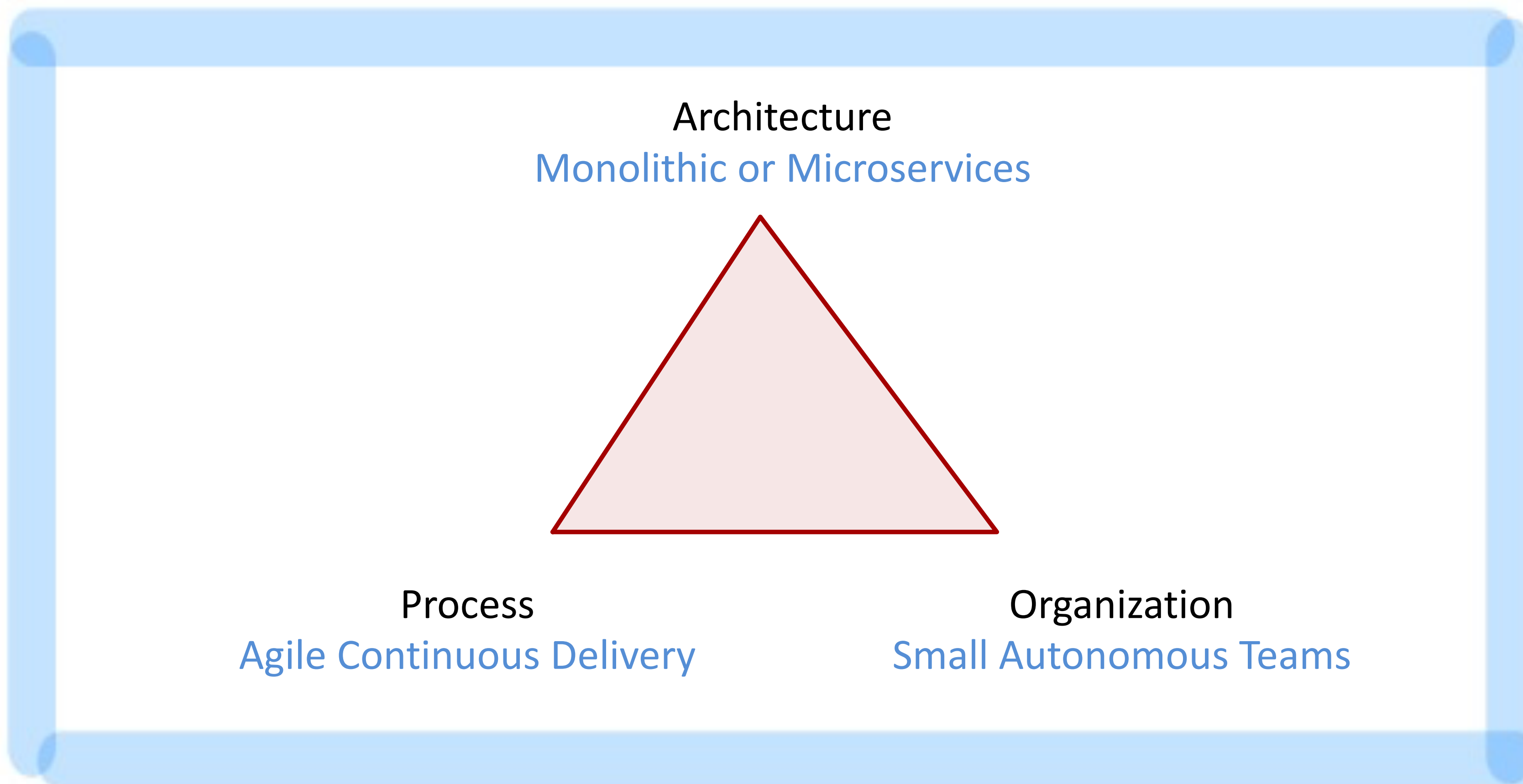
- Imagine you are building a large complex application (e.g. Online Store)



Slide is tare taken from Ajeet Singh Raine

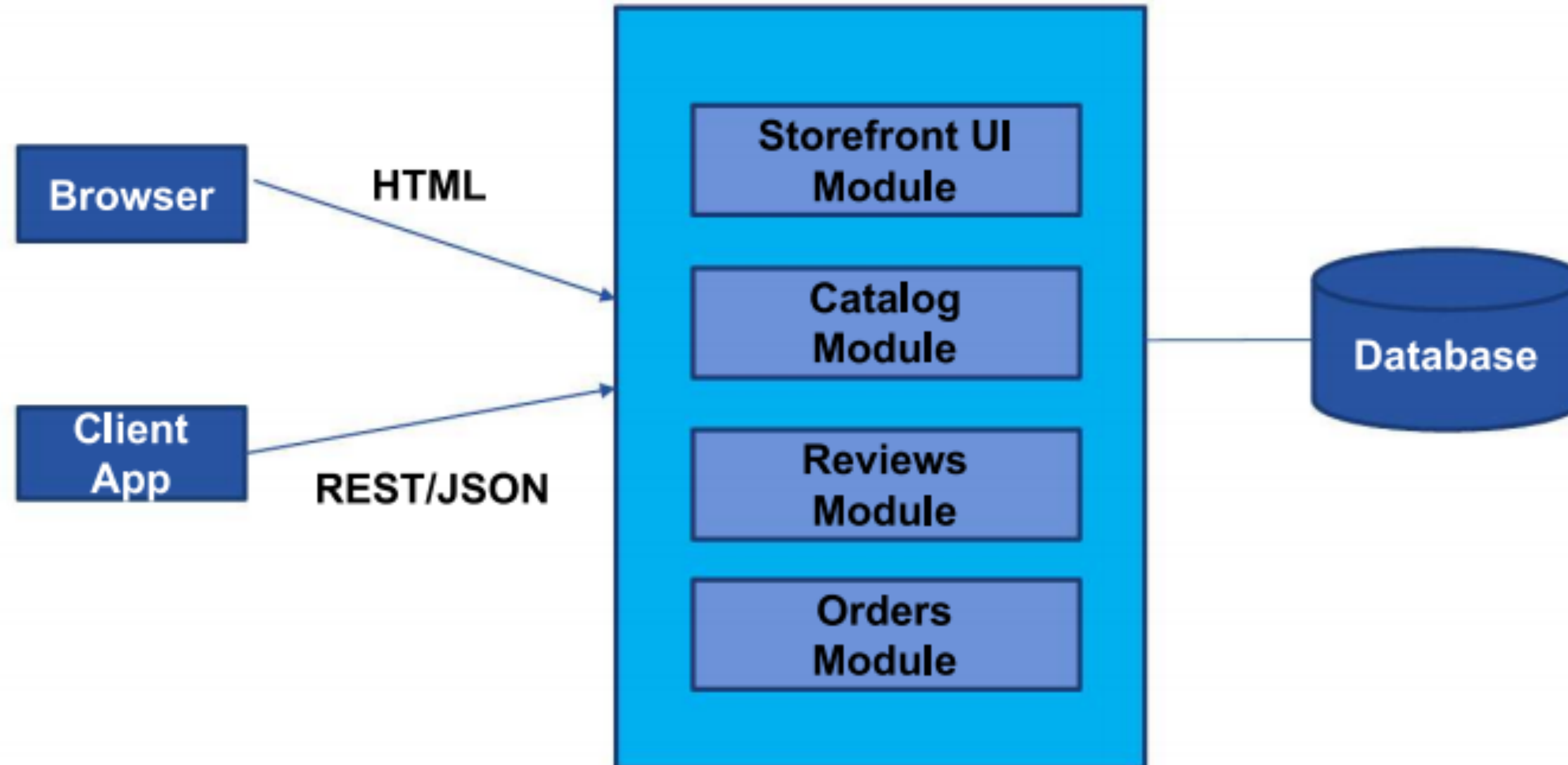
Successful Software Application

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



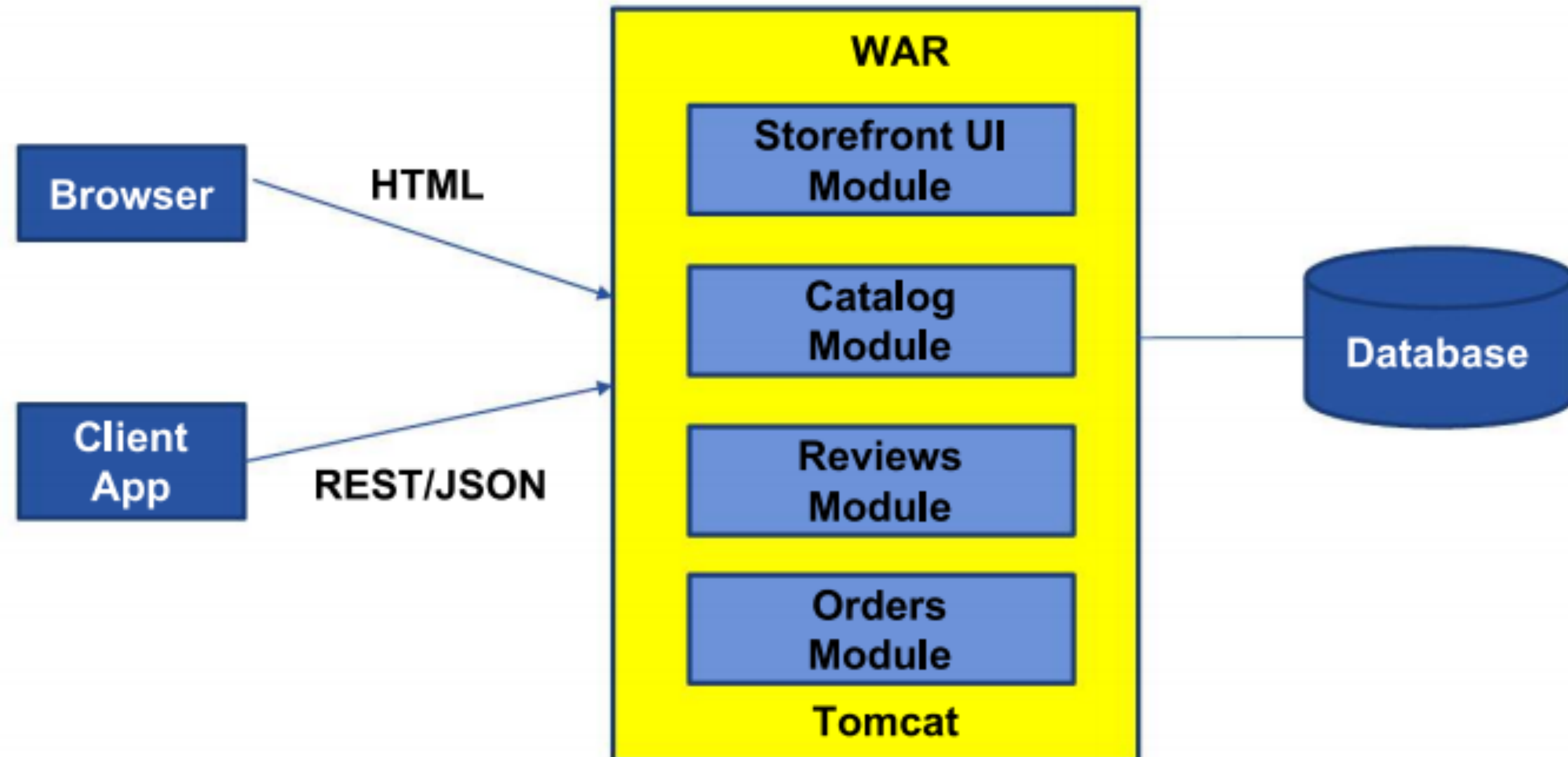
Slide is tare taken from Ajeet Singh Raine

Monolithic Architecture



Slide is tare taken from Ajeet Singh Raine

Monolithic Architecture



Slide is tare taken from Ajeet Singh Raine

Benefits of Monolithic

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 Monolithic

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 dramatically

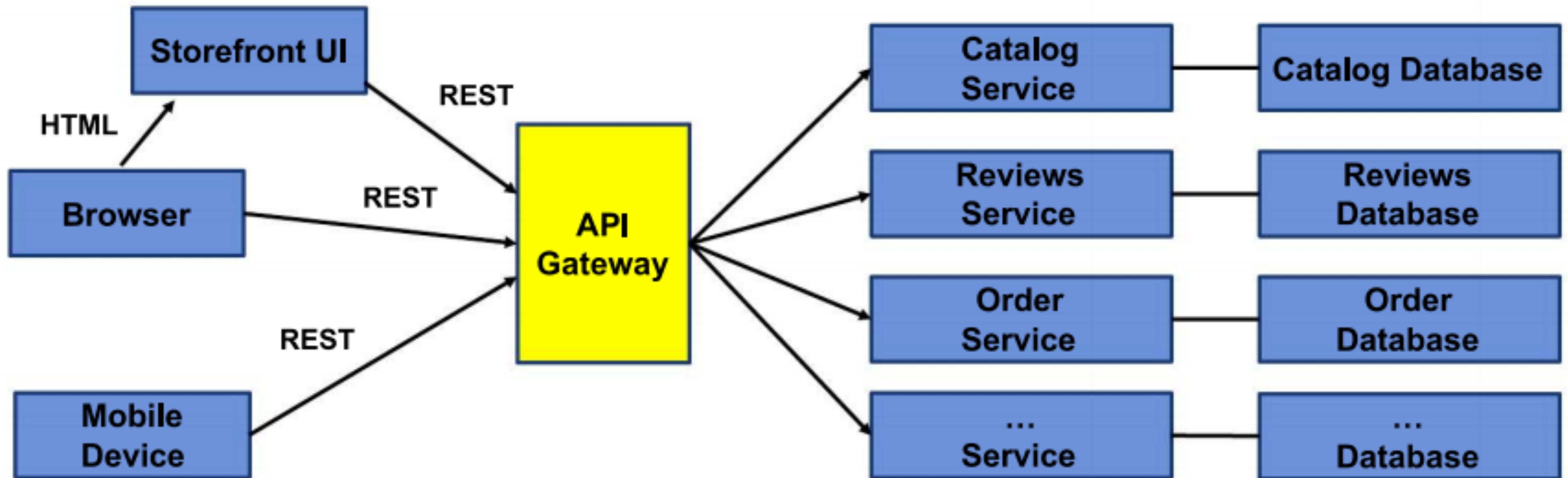
A decade ago

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

Today

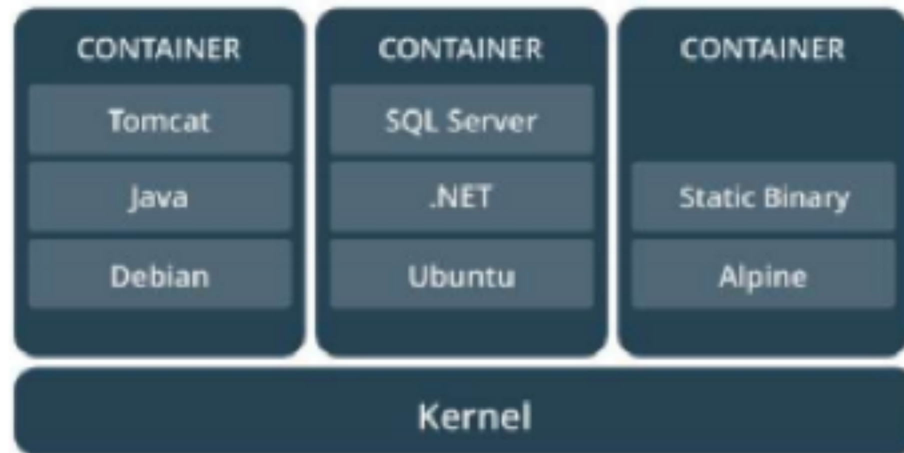
Apps are constantly being developed
Build from loosely coupled components
Newer version are deployed often
Deployed to a multitude of servers

Microservice Architecture



IACS

What is container



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

Images and Containers

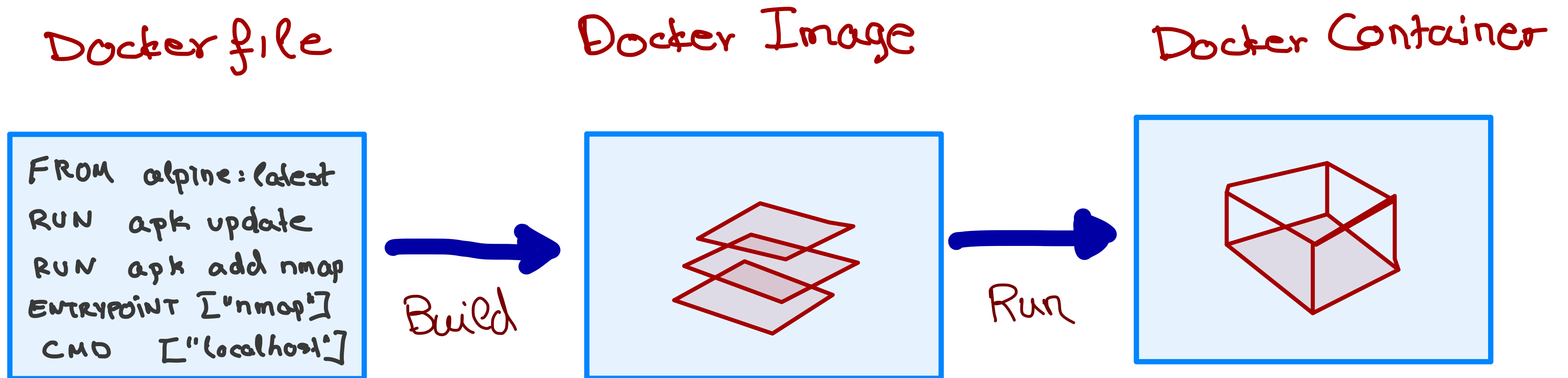
Docker Image is a template aka blueprint to create a running Docker container. Docker uses the information available in the Image to create (run) a container.

Image is like a recipe, container is like a dish

You can think of an image as a class and a container is an instance of that class.

How to build an image

We use the Dockerfile, a simple text file, to build the Docker Image, which are iso files and other files. We run the Docker Image to get Docker Container.



Inside the Dockerfile

Dockerfile

```
FROM alpine:latest
RUN apk update
RUN apk add nmap
ENTRYPOINT ["nmap"]
CMD ["localhost"]
```

FROM: This instruction in the Dockerfile tells the daemon, which base image to use while creating our new Docker image. In the example here, we are using a very minimal OS image called alpine (just 5 MB of size). You can also replace it with Ubuntu, Fedora, Debian or any other OS image.

RUN: This command instructs the Docker daemon to run the given commands as it is while creating the image. A Dockerfile can have multiple RUN commands, each of these RUN commands create a new **layer** in the image.

ENTRYPOINT: The ENTRYPOINT instruction is used when you would like your container to run the same executable every time. Usually, ENTRYPOINT is used to specify the binary and CMD to provide parameters.

CMD: The CMD sets default command and/or parameters when a docker container runs. **CMD can be overwritten** from the command line via the docker run command.

Multiple containers from same image

How can you run multiple containers from the same image? Wouldn't they all be identical?

Yes, you could think of an image as instating a class. You could instate it with different parameters using the CMD and therefore different containers will be different.

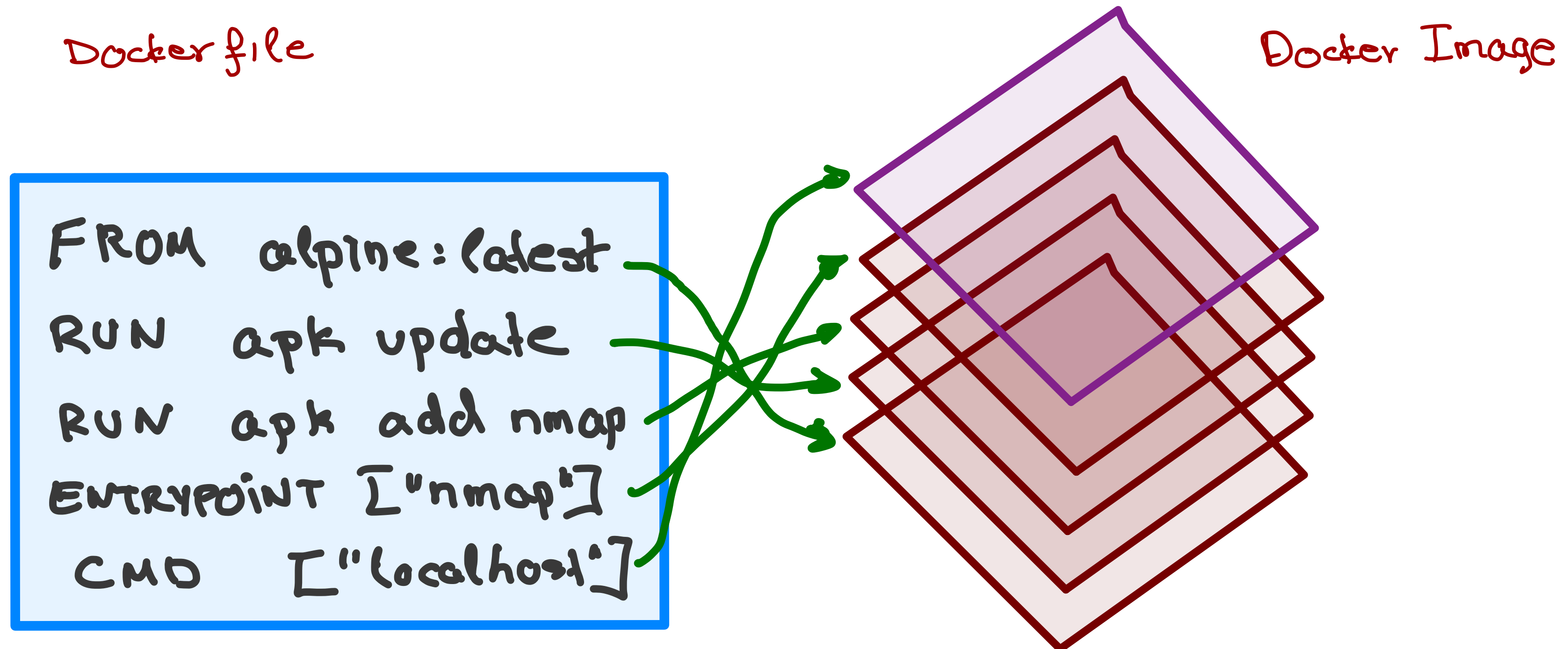
Dockerfile

```
FROM ubuntu:latest
RUN apt-get update
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["world"]
```

```
> docker build -t hello_world_cmd:first -f Dockerfile_cmd .

> docker run -it hello_world_cmd:first
> Hello world
> docker run -it hello_world_cmd:first Pavlos
> Hello Pavlos
```

Docker Image as Layers

When we execute the build command, the daemon reads the Dockerfile and creates a layer for every command.



```
>docker build -t hello_world_cmd -f Dockerfile_cmd .

Sending build context to Docker daemon   34.3kB
Step 1/4 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
54ee1f796a1e: Already exists
f7bfea53ad12: Already exists
46d371e02073: Already exists
b66c17bbf772: Already exists
Digest: sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Status: Downloaded newer image for ubuntu:latest
 ---> 4e2eef94cd6b
Step 2/4 : RUN apt-get update
 ---> Running in e3e1a87e8d6e
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [67.5
kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [231 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [1078
B]
…
```

Step1: Instruction 1

Step2: Instruction 2

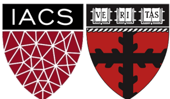
```
>docker build -t hello_world_cmd -f Dockerfile_cmd .

….
Step 3/4 : ENTRYPOINT ["/bin/echo", "Hello"]
 ---> Running in 52c7a98397ad
Removing intermediate container 52c7a98397ad
 ---> 7e4f8b0774de
Step 4/4 : CMD ["world"]
 ---> Running in 353adb968c2b
Removing intermediate container 353adb968c2b
 ---> a89172ee2876
Successfully built a89172ee2876
Successfully tagged hello_world_cmd:latest
```

Step3: Instruction 3

Step4: Instruction 4

```
> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello_world_cmd     latest              a89172ee2876        7 minutes ago       96.7MB
ubuntu              latest              4e2eef94cd6b        3 weeks ago         73.9MB
```

```
> docker image history hello_world_cmd
IMAGE               CREATED             CREATED
BY                                      SIZE                COMMENT
a89172ee2876        8 minutes ago       /bin/sh -c #(nop)  CMD ["world"]                0B
7e4f8b0774de        8 minutes ago       /bin/sh -c #(nop)  ENTRYPOINT ["/bin/echo" "…   0B
cfc0c414a914        8 minutes ago       /bin/sh -c apt-get update                       22.8MB
4e2eef94cd6b        3 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           3 weeks ago         /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>           3 weeks ago         /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   811B
<missing>           3 weeks ago         /bin/sh -c [ -z "$(apt-get indextargets)" ]     1.01MB
<missing>           3 weeks ago         /bin/sh -c #(nop) ADD file:9f937f4889e7bf646…   72.9MB
```



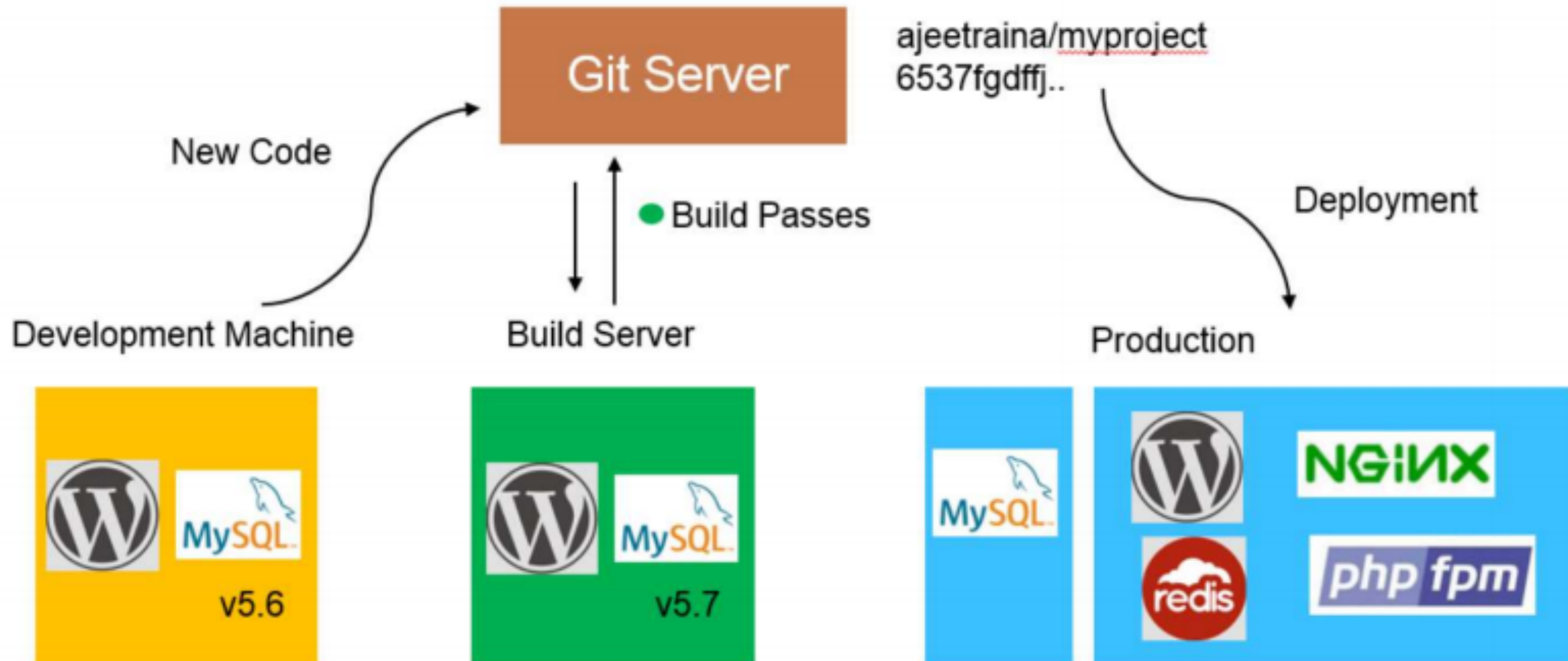
Why Layers

Why build an image with multiple layers when we can just build it in a single layer?
Let’s take an example to explain this concept better, let us try to change the Dockerfile_cmd we created and rebuild a new Docker image.

```
> docker build -t hello_world_cmd -f Dockerfile_cmd .
Sending build context to Docker daemon   34.3kB
Step 1/4 : FROM ubuntu:latest
 ---> 4e2eef94cd6b
Step 2/4 : RUN apt-get update
 ---> Using cache
 ---> cfc0c414a914
Step 3/4 : ENTRYPOINT ["/bin/echo", "Hel
 ---> Using cache
 ---> 7e4f8b0774de
Step 4/4 : CMD ["world"]
 ---> Using cache
 ---> a89172ee2876
Successfully built a89172ee2876
Successfully tagged hello_world_cmd:latest
```

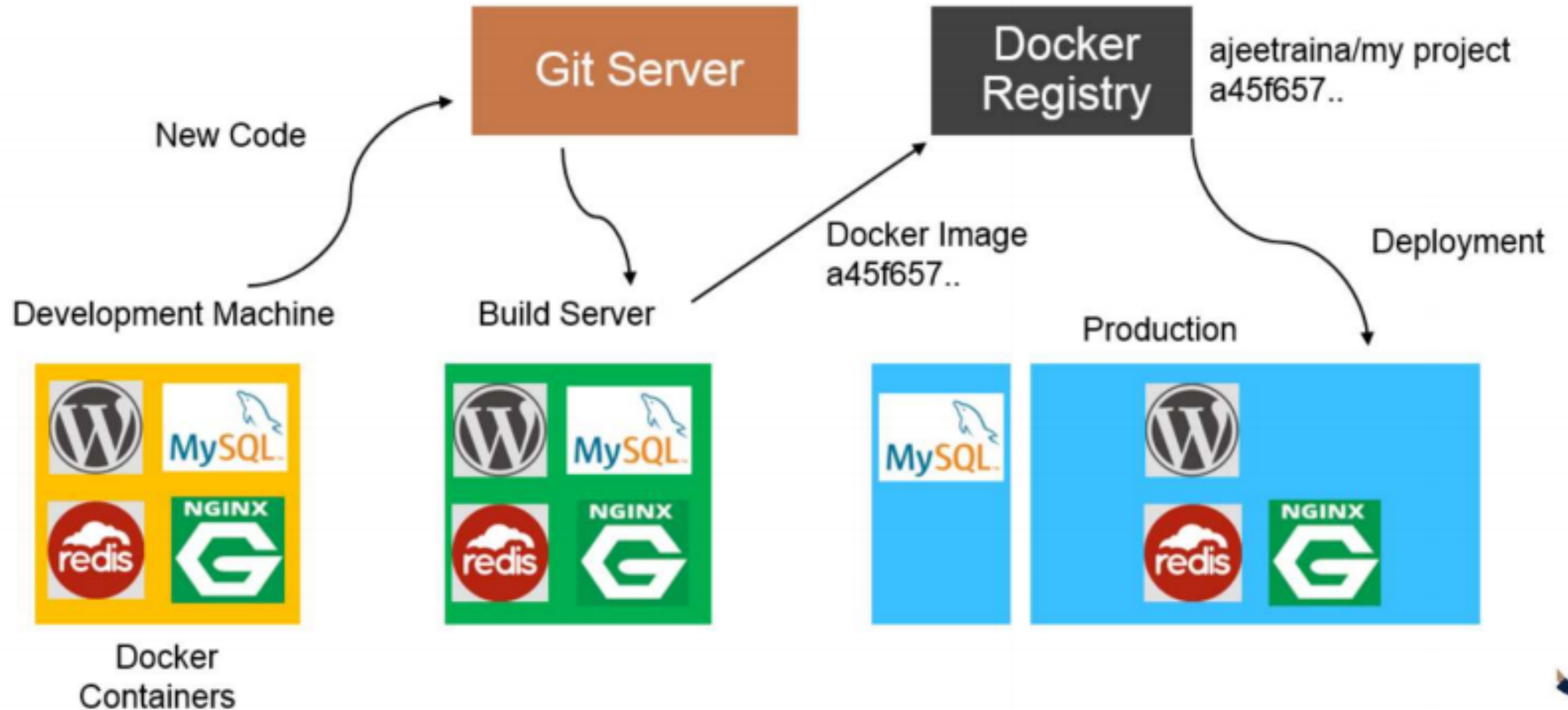
As you can see that the image was built using the existing layers from our previous docker image builds. If some of these layers are being used in other containers, they can just use the existing layer instead of recreating it from scratch.

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 Registry Services

DOCKER REGISTRY SERVICES

DOCKER HUB

Docker hub is the official image repository of the docker.Its helps to store , share and distribute the docker image

QUAY

It is the docker registry owned by Red hat. Its helps to create on premises and cloud repository

GOOGLE CONTAINER REGISTRY

It is the docker registry created by the google.Its used to setup the private registeries

It is docker registry created

Docker Containers are not Virtual Machines

Virtual Machines

Containers

Docker Container vs Virtual Machines (VM)

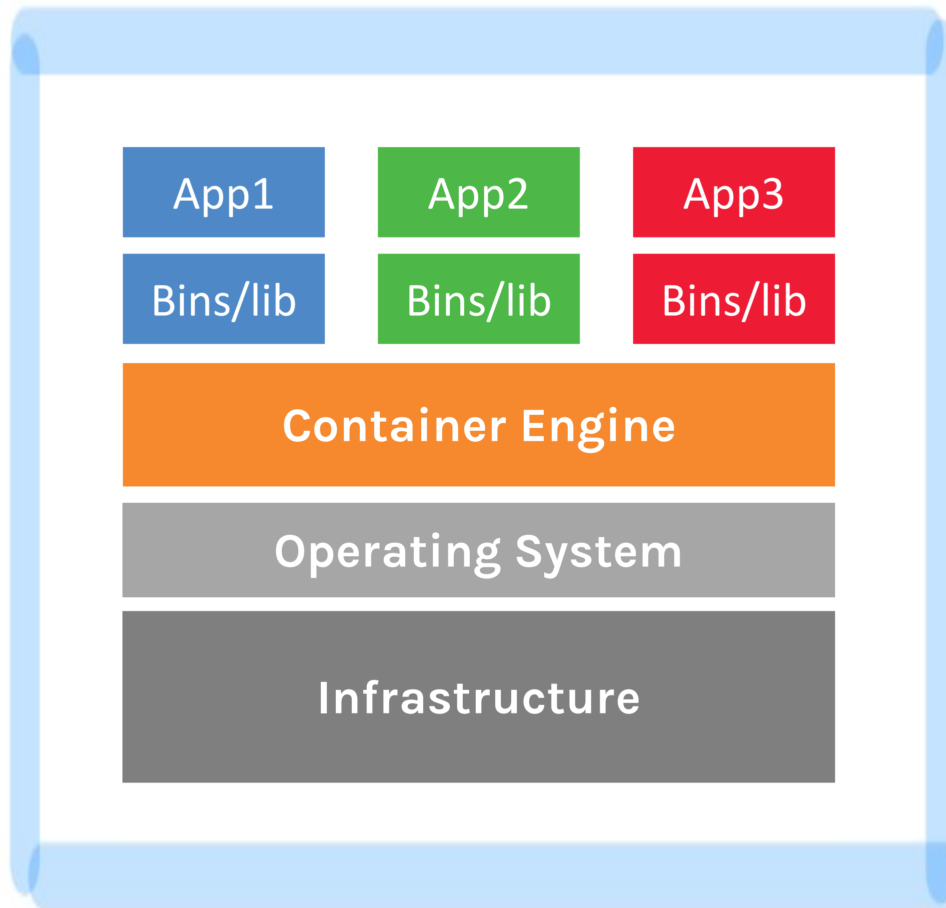
VMs

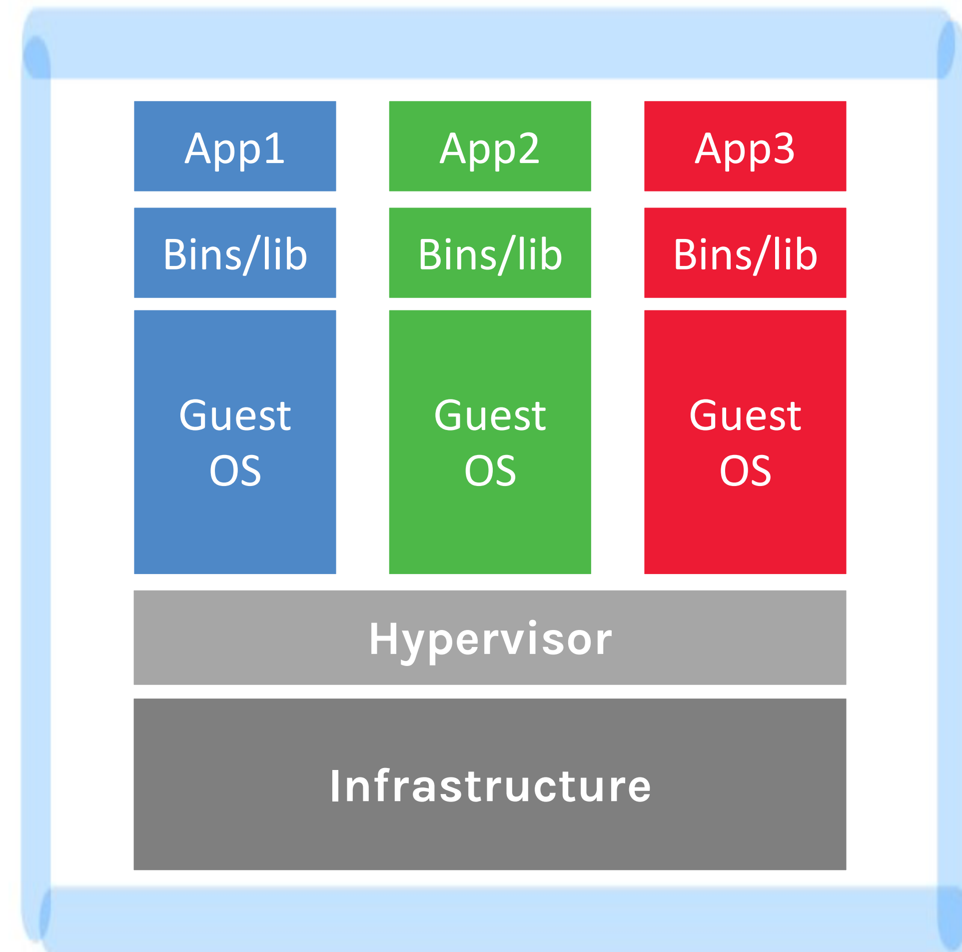
- Each VM runs its own OS
- Boot up time is in minutes
- Not version controlled
- Cannot run more than couple of VMs on an average laptop
- Only one VM can be started from one set of VMX and VMDK files

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

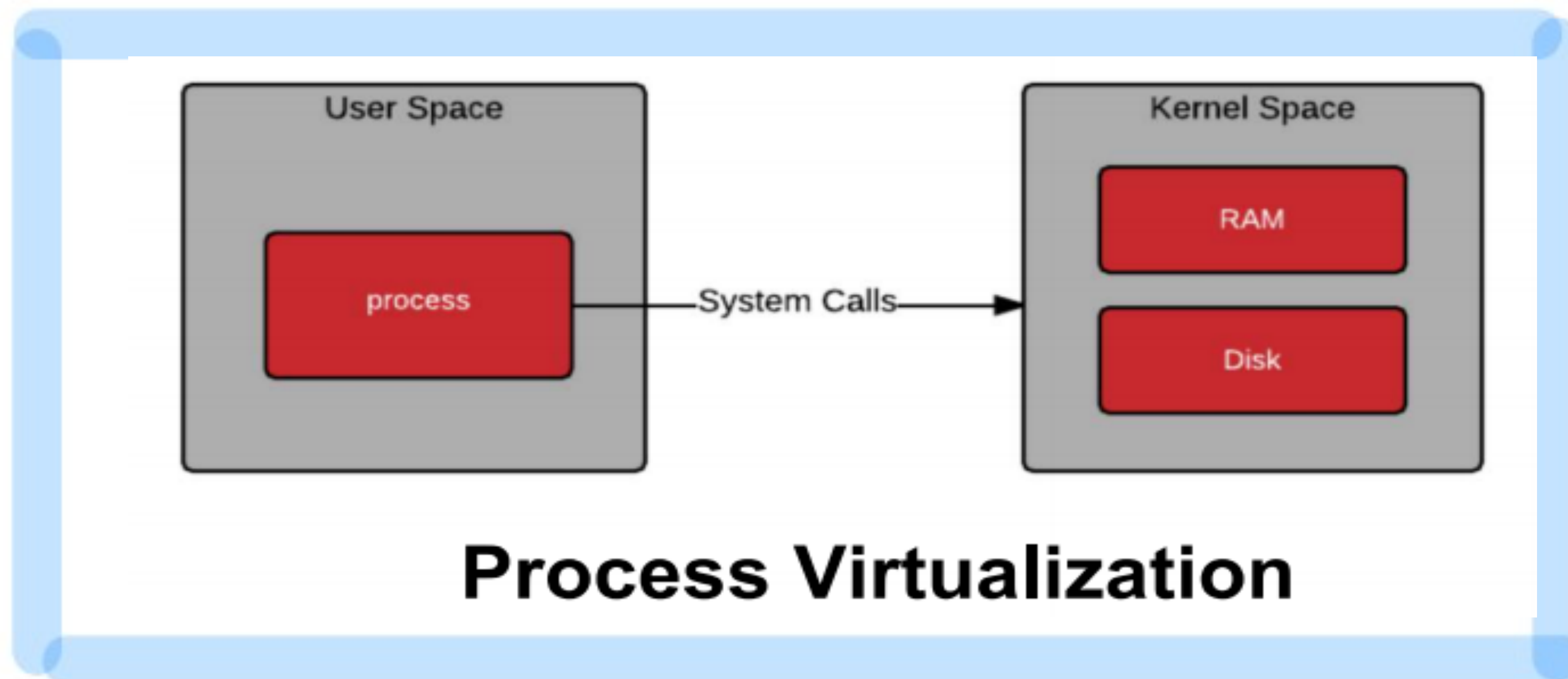




What Makes Containers so Small?

Container = User Space of OS

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

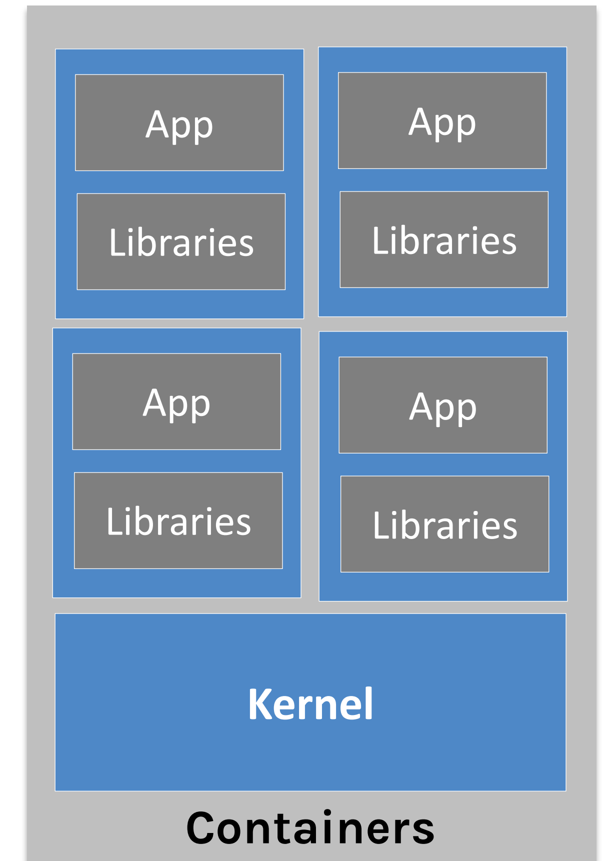


Why should we use containers?

- 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

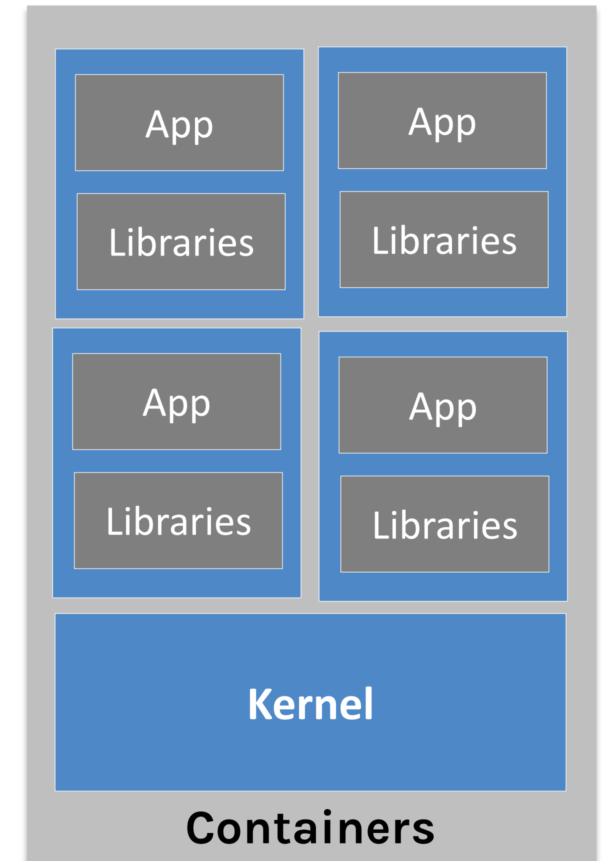
- The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.

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



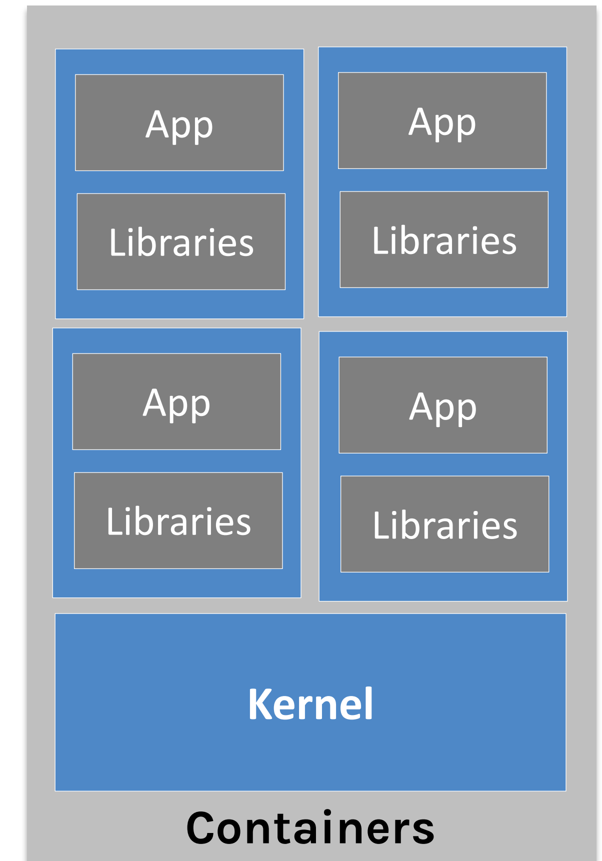
Why should we use containers? (cont)

- 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)

Comparison

	VIRTUAL ENV	DOCKER	VM	JH
COMPUTATIONAL COST MEMORY FOOTPRINT	LOW	MEDIUM LOW	HIGH	?
DEPLOYMENT	EASY	MEDIUM	INIT HIGH THEN EASY	N/A
VERSATILITY (TYPES OF APPS)	MEDIUM	MEDIUM HIGH	MEDIUM HIGH	LOW
PORTABILITY	MEDIUM	HIGH	HIGH	HIGH

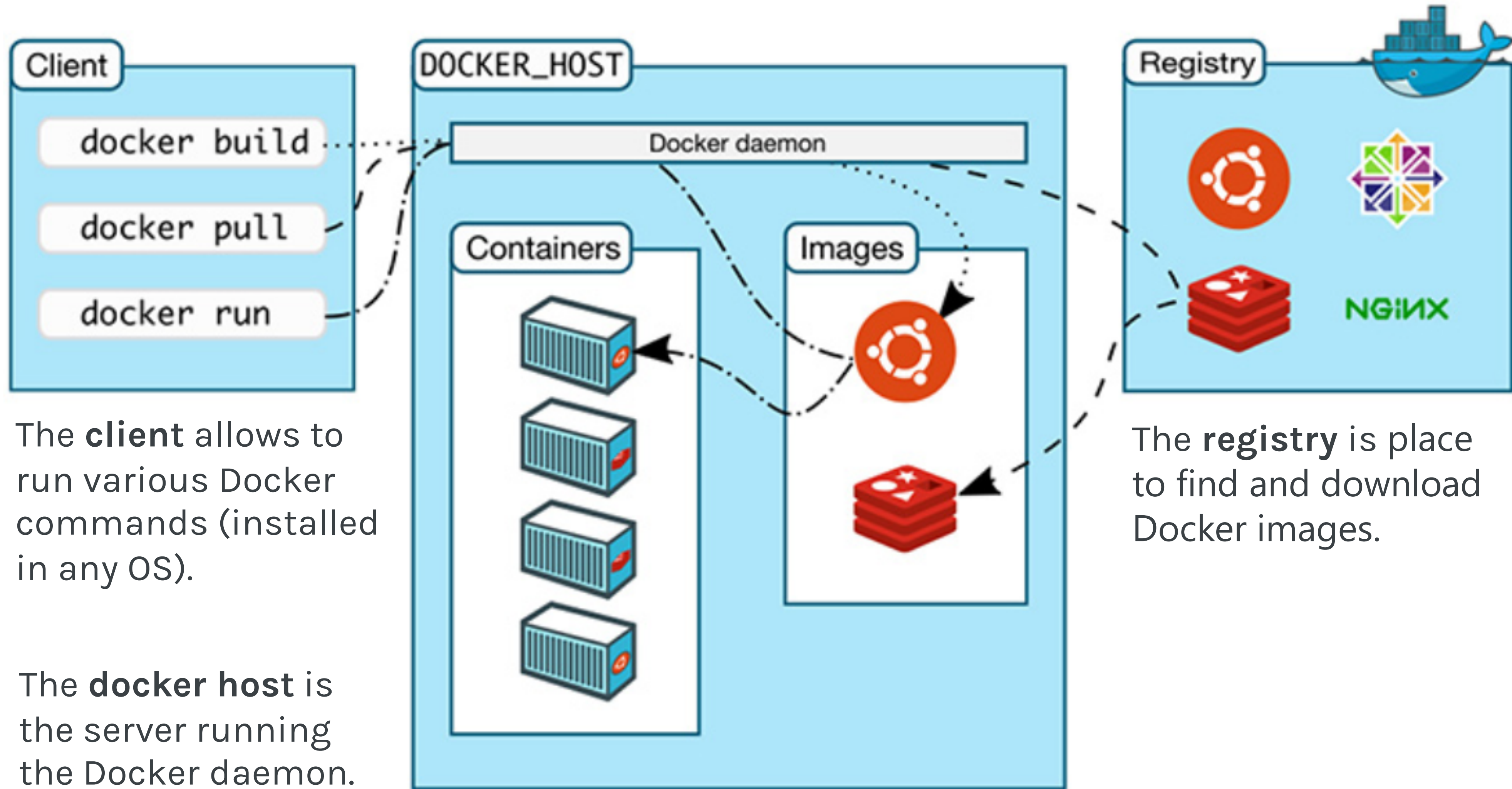
COMPUTATIONAL SCIENCE

DEVOPS

DATA SCIENCE (NO PIPELINE)

DATA SCIENCE (PIPELINES)

The Docker Engine Architecture



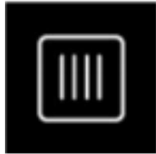
The **client** allows to run various Docker commands (installed in any OS).

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

Outline

1: Class organization

2: Recap

3: Software Development

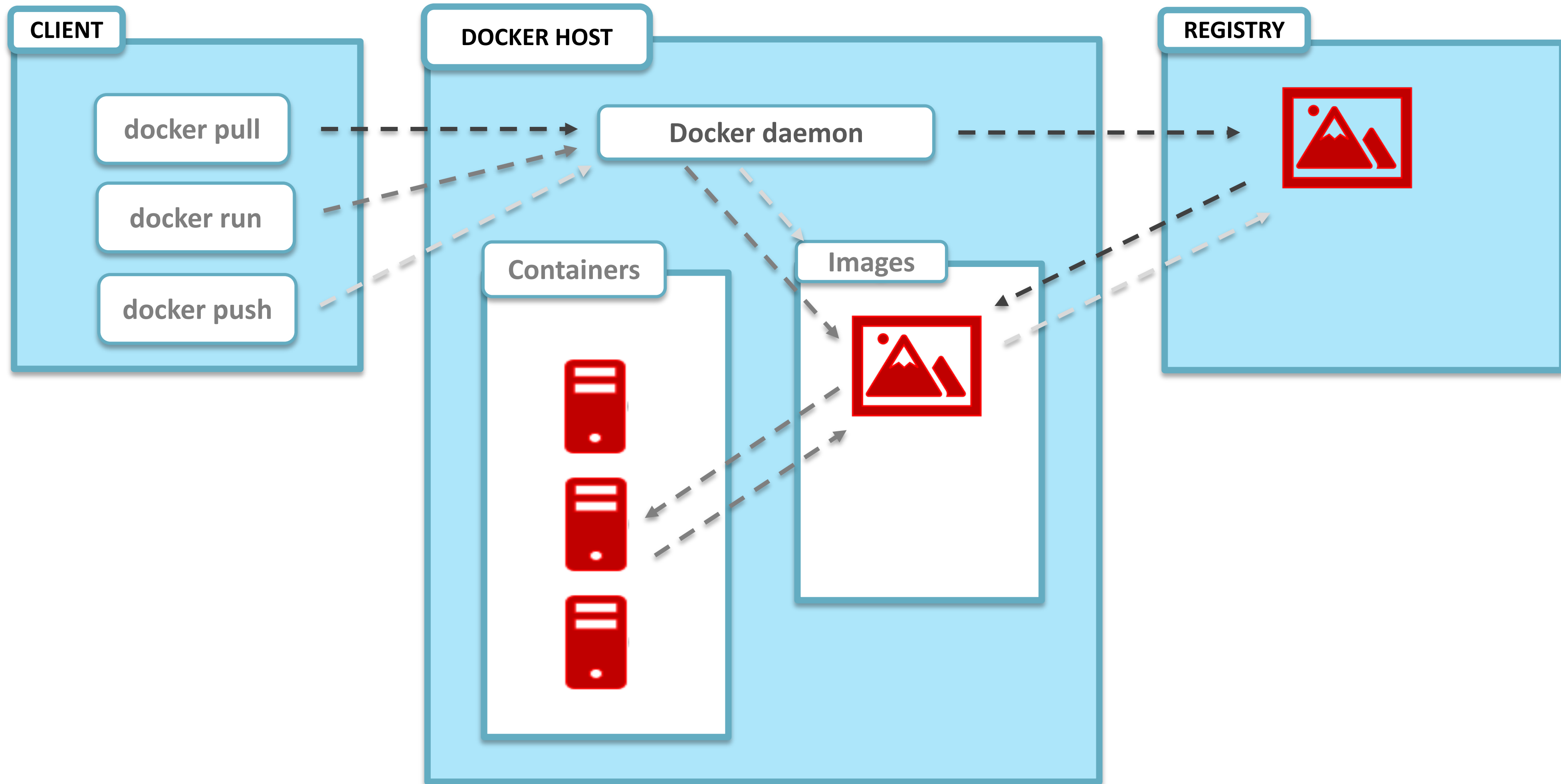
4: **Demo**

Hands on Containers

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

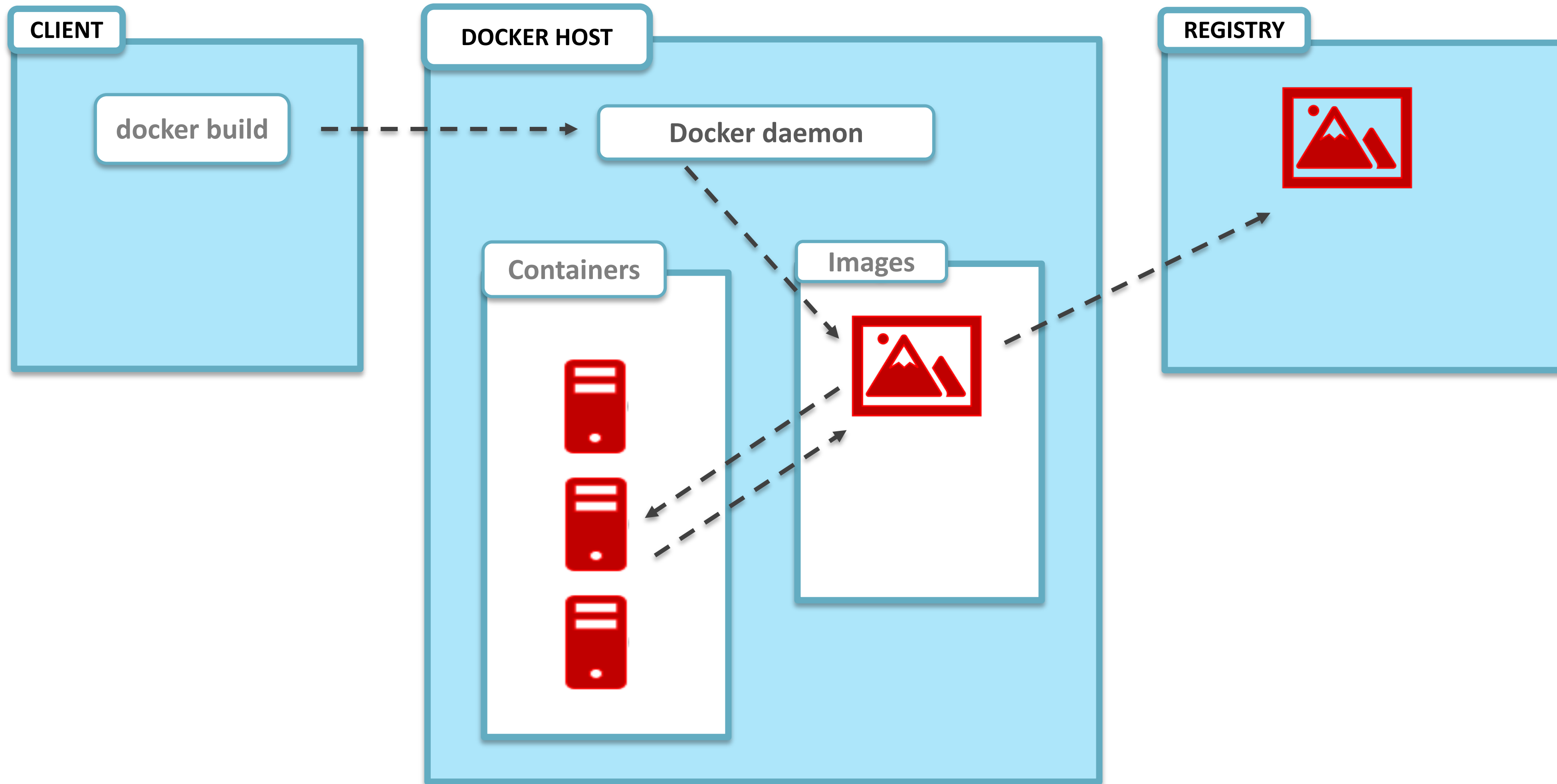
Exercise 2: Build a Docker Image and push it to DockerHub

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

Exercise 2: Docker Do It Yourself

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

THANK YOU