Lecture 2: Environments, Virtual Machines and Containers

AC295

Advanced Practical Data Science

Pavlos Protopapas

Outline

1: Class organization

2: Virtual environments

3: Virtual machines

4: Containers

5: Advanced topics

Class organization

Github for this class (Michael)

Group formation

Presentation schedule

Review class flow

Auditors

Outline

1: Class organization

2: **Virtual environments**

3: Virtual machines

4: Containers

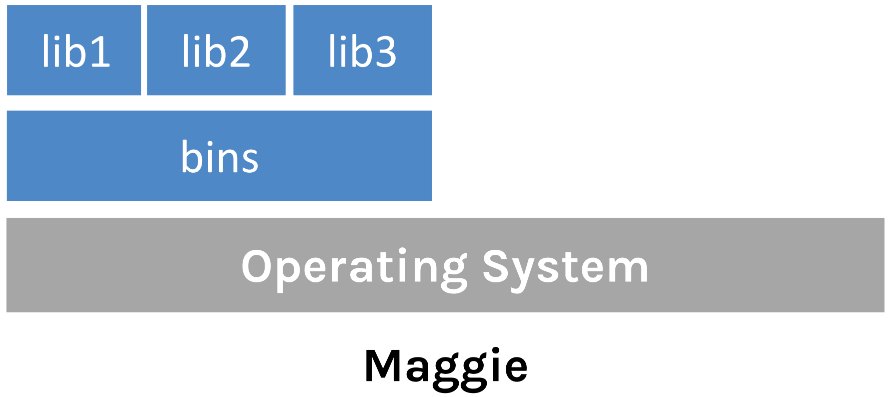
5: Advanced topics

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?

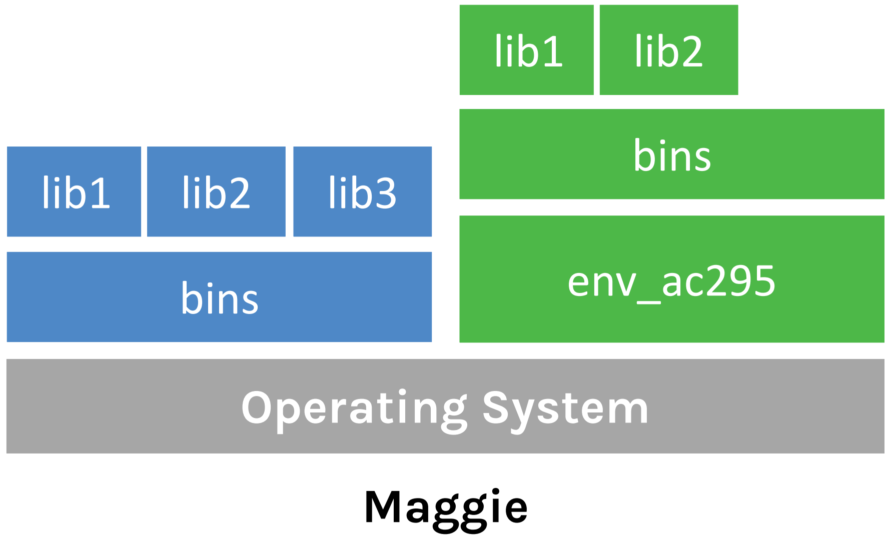
- Maggie took cs109a, she used to run her Jupyter notebooks from anaconda prompt. Every time she installed a module it was placed in the either of `bin`, `lib`, `share`, `include` folders and she could import it in and used it without any issue.



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

Why should we use virtual environment?

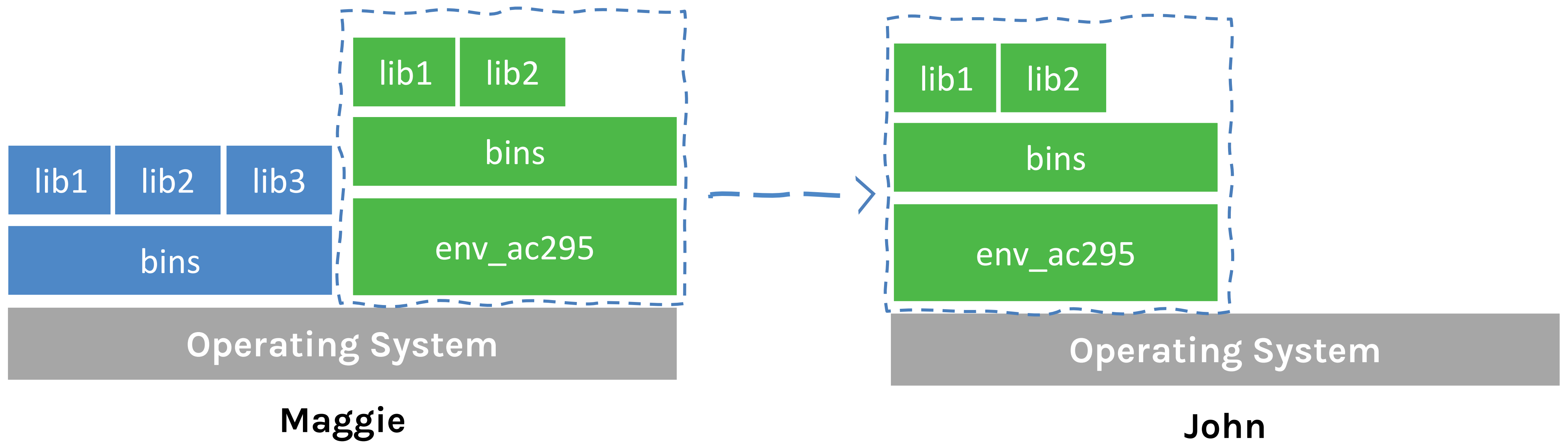
- Maggie starts taking ac295 and she thinks that would be good to isolate the new environment from the previous enviroments avoiding any conflict with the installed packages. She adds a layer of abstraction called virtual environment that helps her keep the modules organized and avoid misbehaviors while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



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

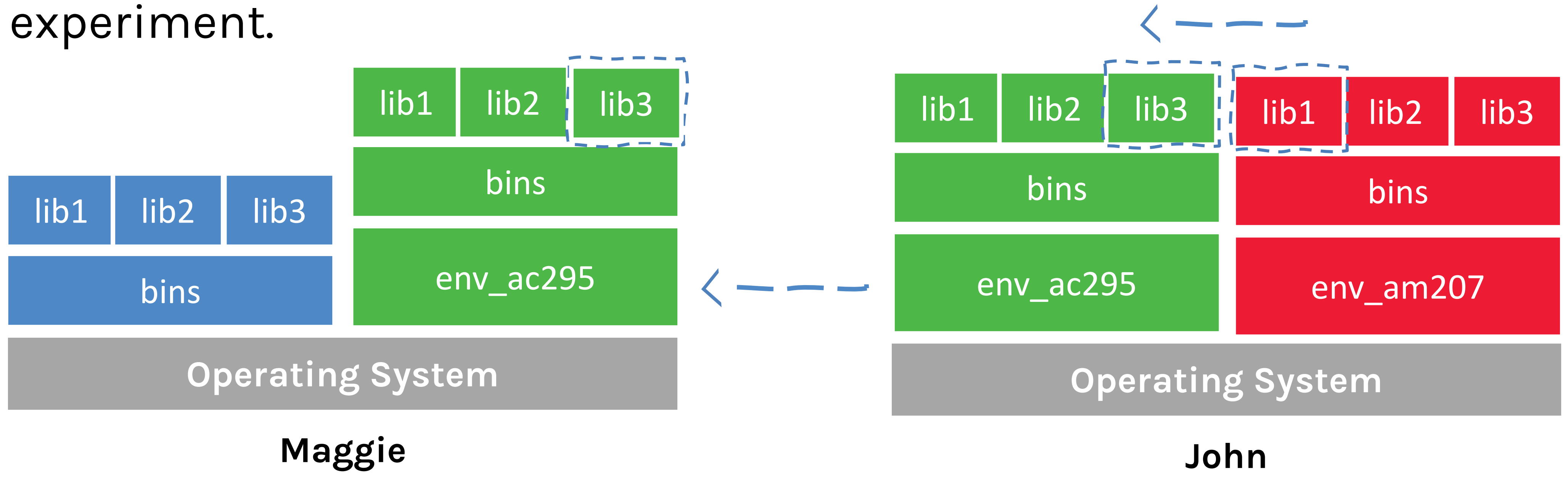
Why should we use virtual environment?

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



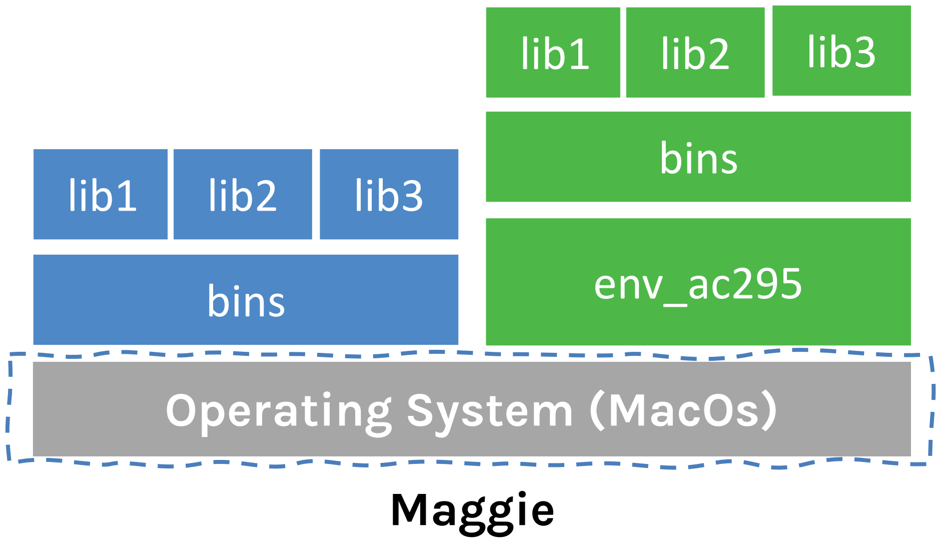
Why should we use virtual environment?

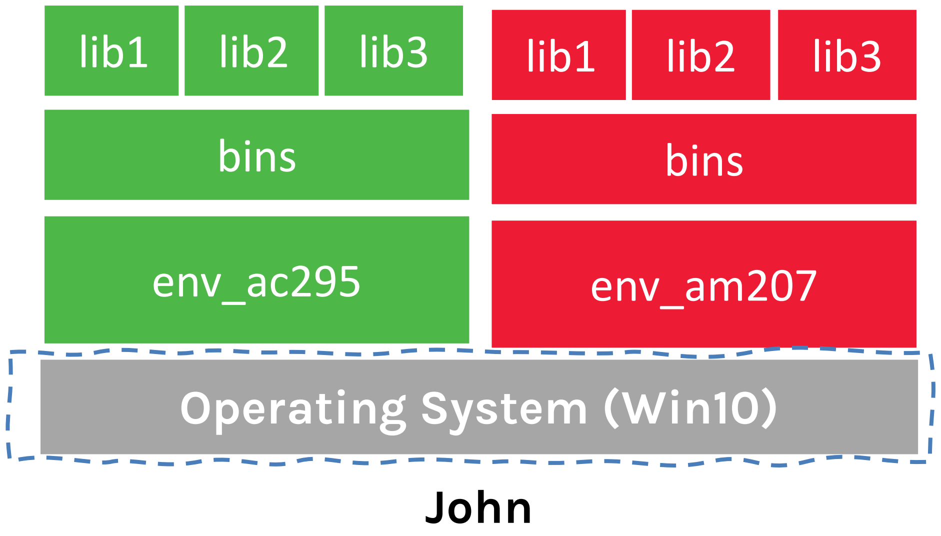
- John experiments a new method he learned in another class and adds a new library to the working environment. After seeing a tremendous improvements he sends Maggie back his code and a new .yml file. She can now update her environment and replicate the experiment.



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.





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

> Adapted from CS207 <

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

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)

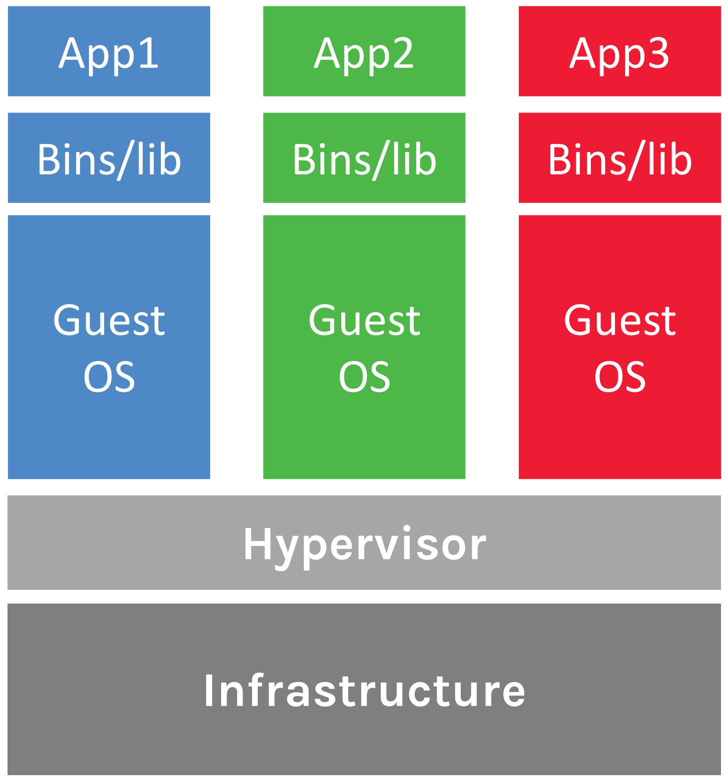
Advantages

- 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 much 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

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



Machine Virtualization

Limitations

- Uses hardware in your local machine
- There is overhead associated with virtual machines
    1. guest is not as fast as the host system (and take some time to start up)
    2. may not have the same graphics capabilities

Outline

1: Class organization

2: Virtual environments

3: Virtual machines

4: Containers

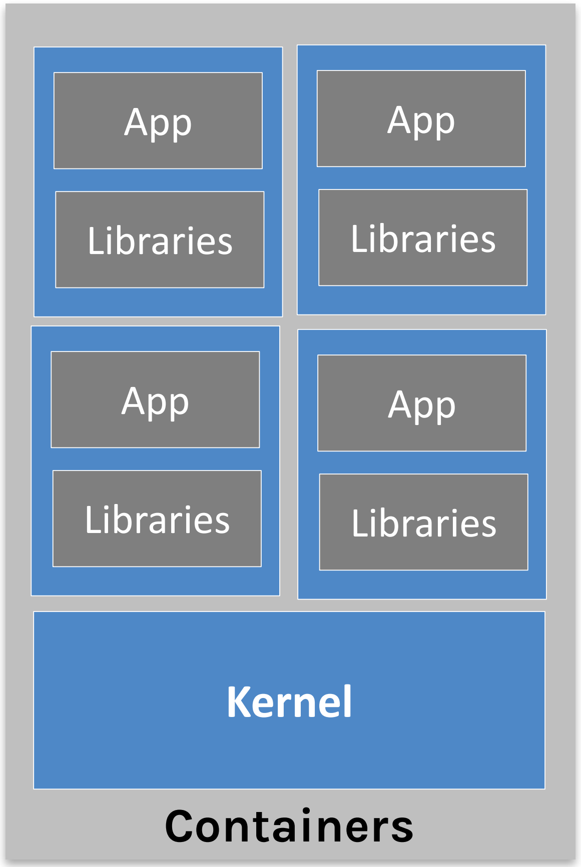
5: Advanced topics

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

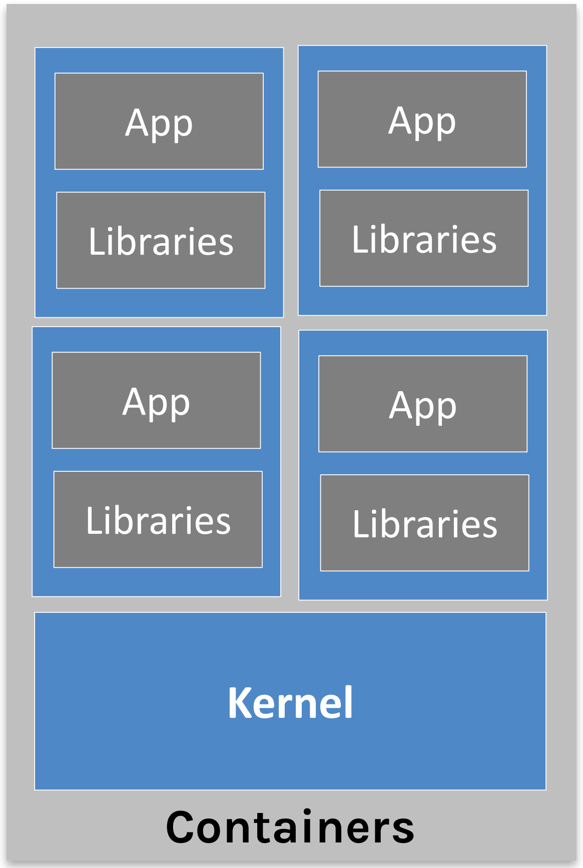
- The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.

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



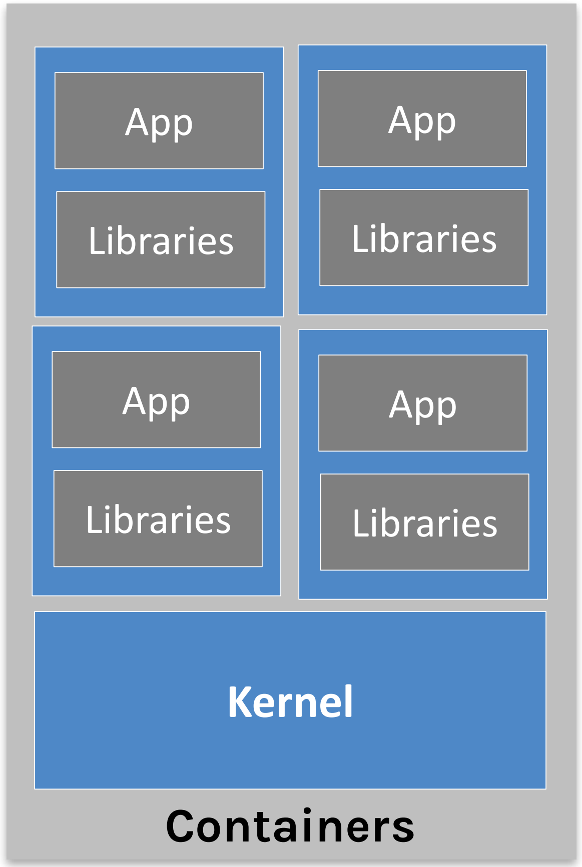
Why should we use containers? (cont)

- 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 much 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 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)

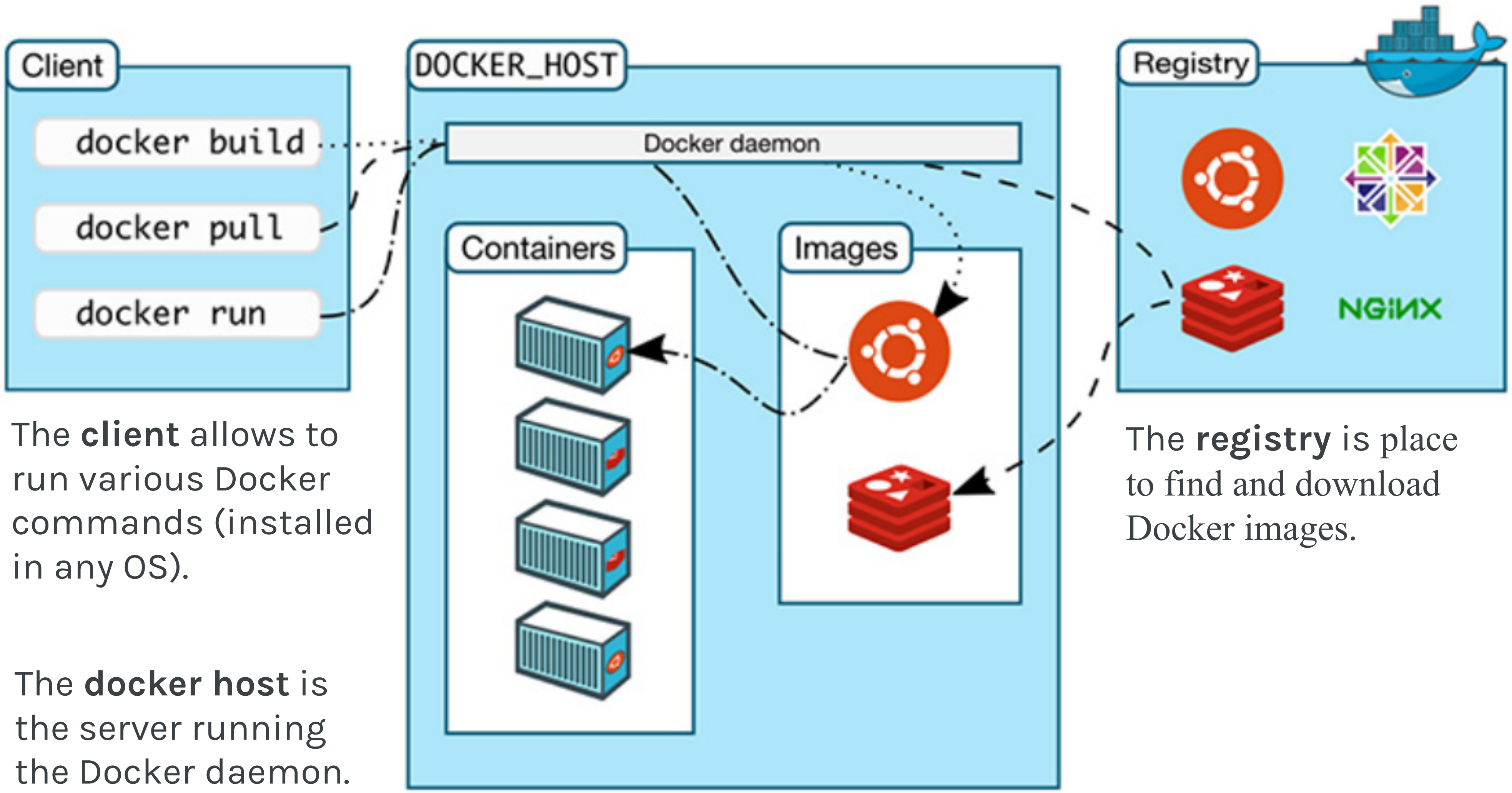
- Containers are application-centric methods to deliver high-performing, scalable applications on any infrastructure of your choice
- Containers are best suited to deliver microservices by providing portable, isolated virtual environments for applications to run without interference from other running applications
- Containers encapsulate microservices and their dependencies but do not run them directly
- 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.
- Containers are lightweight (they cost an order of magnitude less than a virtual machine in memory footprint).



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 Ecosystem



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

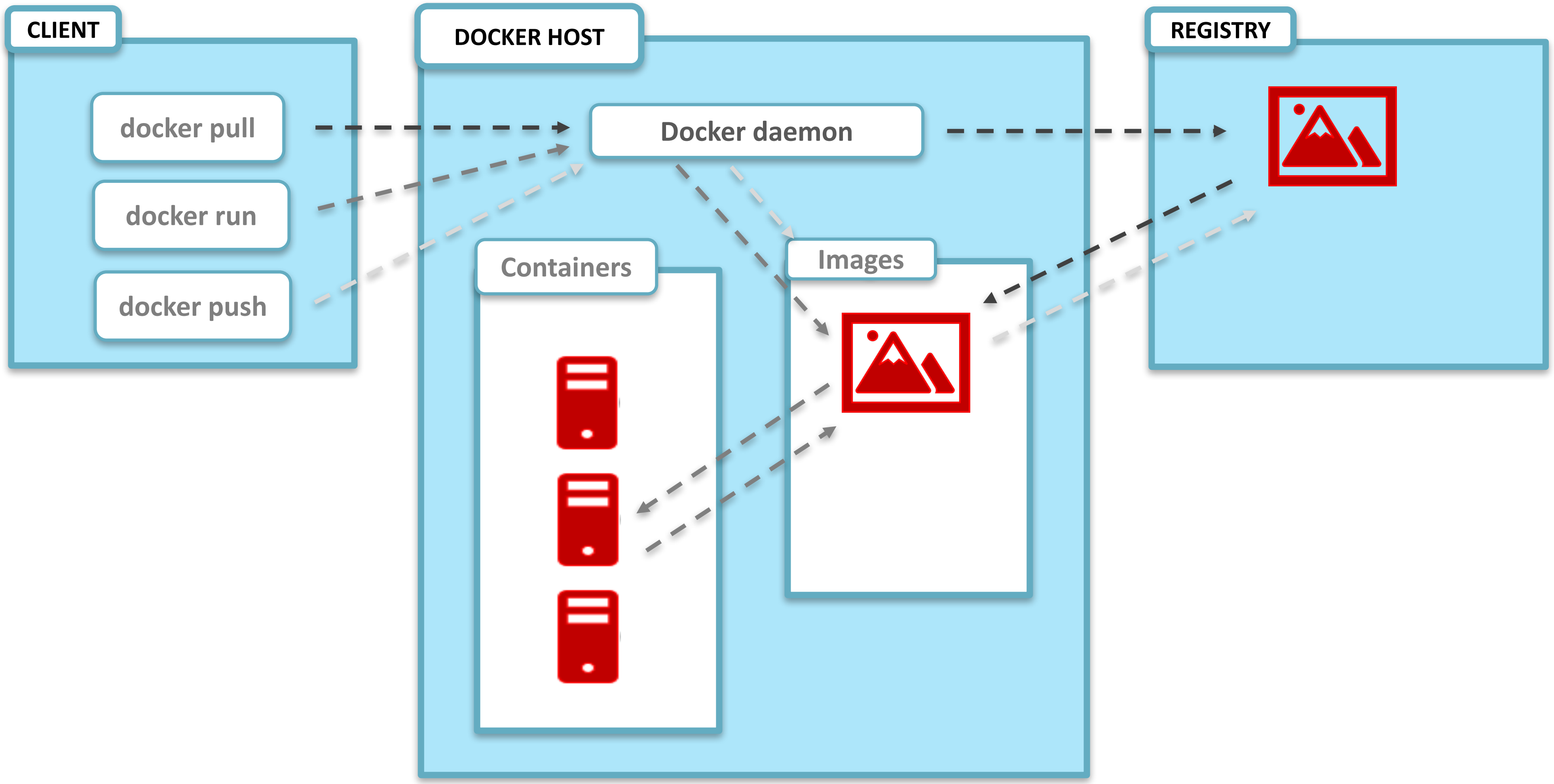
Hands on Containers

Exercise set up

Exercise 1: pull, modify, and push a Docker image from Docker Hub

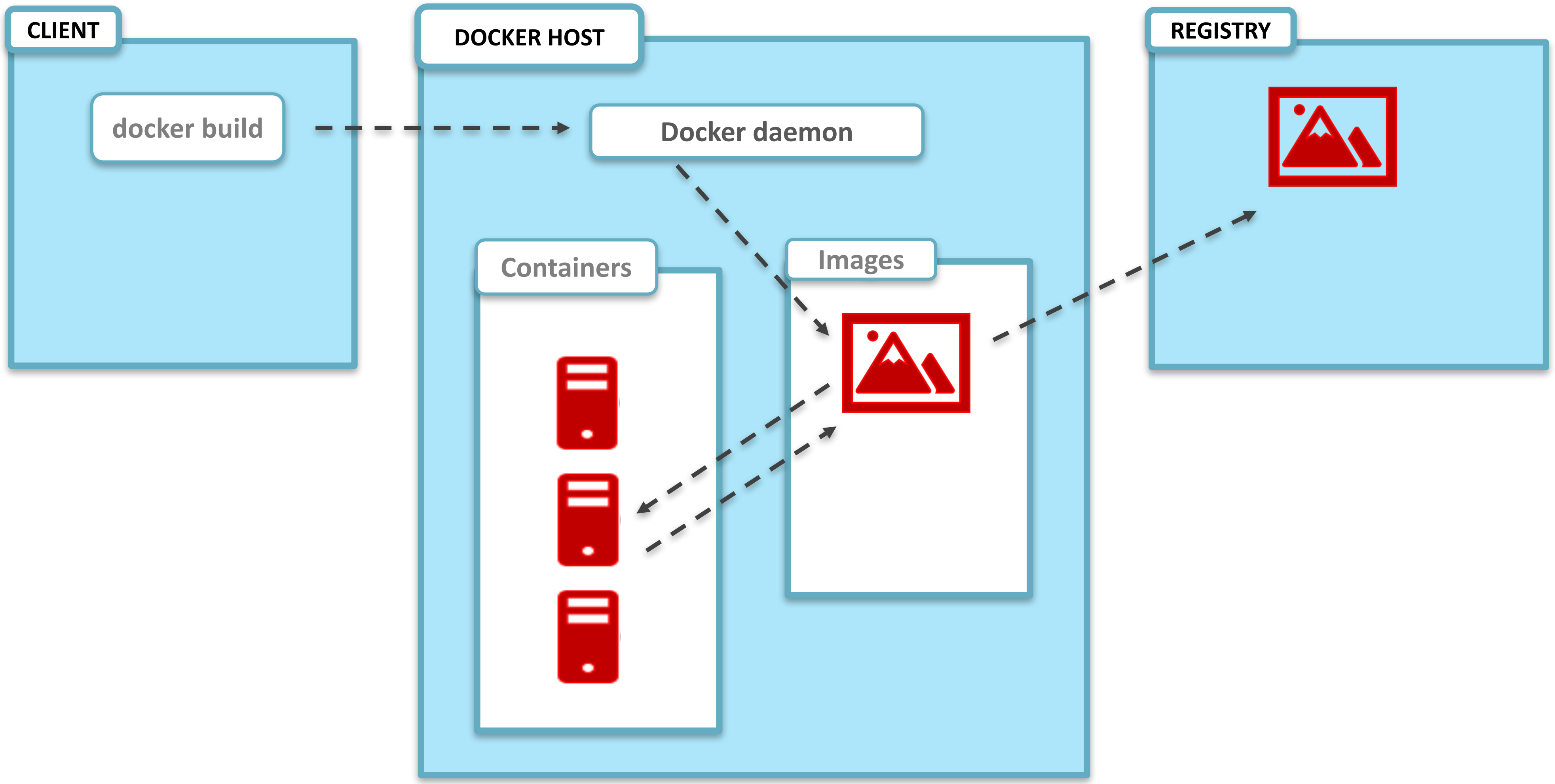
Exercise 2: build a Docker Image and push it to Docker Hub

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 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`)

Outline

Advanced Topic

Dirk Merkel, Docker: Lightweight Linux Containers for Consistent Development and Deployment Light

- "Dependency hell": conflicting and missing dependencies, platform differences
- Docker under the hood (worth reading)
- Containers vs. Other Types of Virtualization
- *Docker Workflow*

Ákos Kovács , Comparison of Different Linux Containers

- Measure containers performance (including Singularity container system)
- Containers considered 1) LXC (Linux Containers), 2) Docker, 3) Singularity (HPC container platform), and measurement: CPU and Networking benchmark
- Containers are almost at the level of the native performance. Containers are also hardening the security. The networking performances are also satisfying, but the greatest area of improvement.

Advanced Topic (cont)

Carlos Arango, Remy Dernat, John Sanabria, Performance Evaluation of Container-based Virtualization for High Performance Computing Environments

- Same containers and measurement used by Kovács
- Singularity containers are more suitable for HPC implementation than Docker or LXC
- For I/O-intensive workloads, Container images can be much more optimal than running against shared storage
- Message: choose technology based on the use

Gregory M. Kurtzer, Vanessa Sochat, Michael W. Bauer, Singularity: Scientific containers for mobility of compute

- Game changer for computational sciences
- The goals of singularity are to Mobility of compute, Reproducibility. User freedom. Support on existing traditional HPC resources.
- Example use cases: academic researcher, server administrator, eliminate redundancy in container technology, running at scale
- Take away is security. No root access.

THANK YOU