Carnegie Mellon University

14-848 Cloud Infrastructure

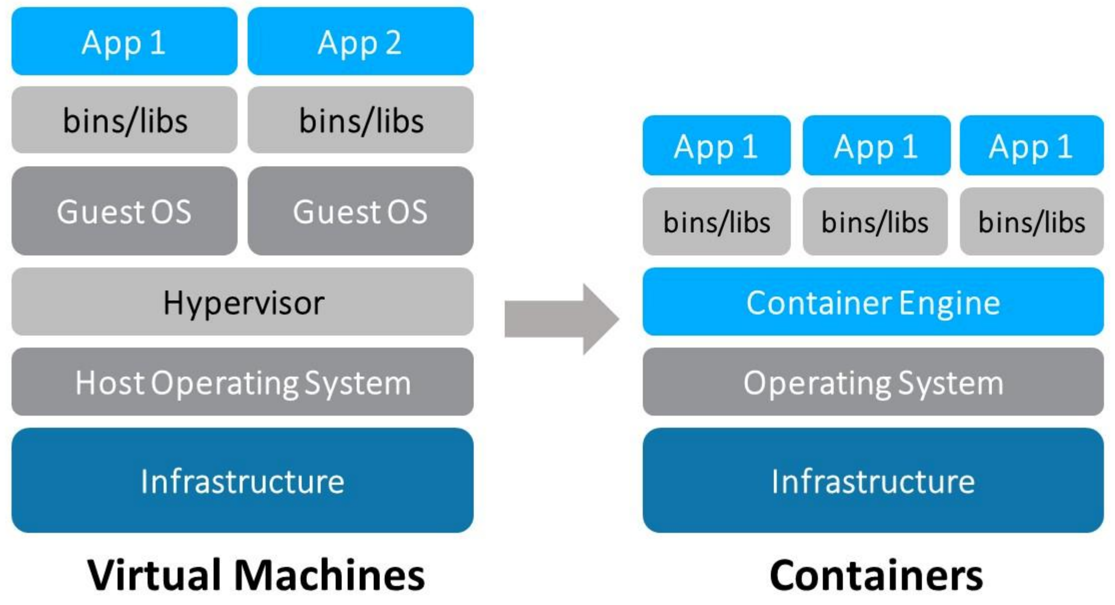
LECTURE 3: FROM VIRTUALIZATION TO CONTAINERIZATION

Agenda

- From Virtualization to Containerization
- Virtual Machine vs Containers
- Why do we need Containerization?
- Understanding Docker
- Docker Lab
- Docker Limitations
- Considerations for M1/M2 Chip Users

Virtual Machines to Containers

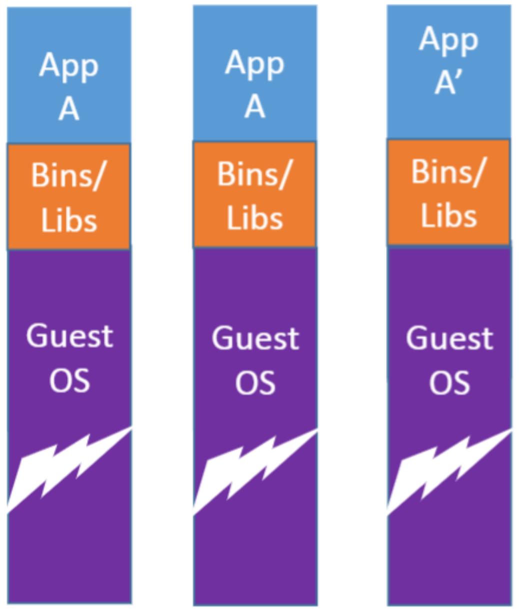
- VM is too heavy for a simple process since it requires the whole OS to be installed
- Containers are isolated, but share OS, and, where appropriate, bins/libs



Carnegie Mellon University

From Virtualization to Containerization

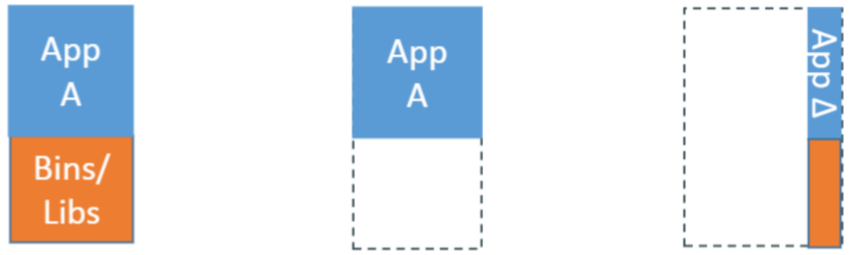
VMs



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App
(No OS to take up space, resources, or require restart)

Copy of App
No OS. Can Share bins/libs

Modified App

Union file system allows us to only save the diffs Between container A and container A'

Carnegie Mellon University

Virtual Machines Vs. Containers

Virtual machines	Containers
Heavyweight	Lightweight
Fully isolated; hence more secure	Process-level isolation; hence less secure
No automation for configuration	Script-driven configuration
Slow deployment	Rapid deployment
Easy port and IP address mapping	More abstract port and IP mappings
Custom images not portable across clouds	Completely portable

Examples:
Citrix Xen,
Microsoft Hyper-V,
VMWare ESXi,
VirtualBox,
KVM

Examples:
Docker,
Google container,
LXC – Linux kernel container,
FreeBSD jails,
Solaris Zones

Why do we need Containerization?

Development Challenges

This app runs on mobile, tablet or Desktop

How to use an application on your machine without having to install it?

Assembled by Developers using best available services

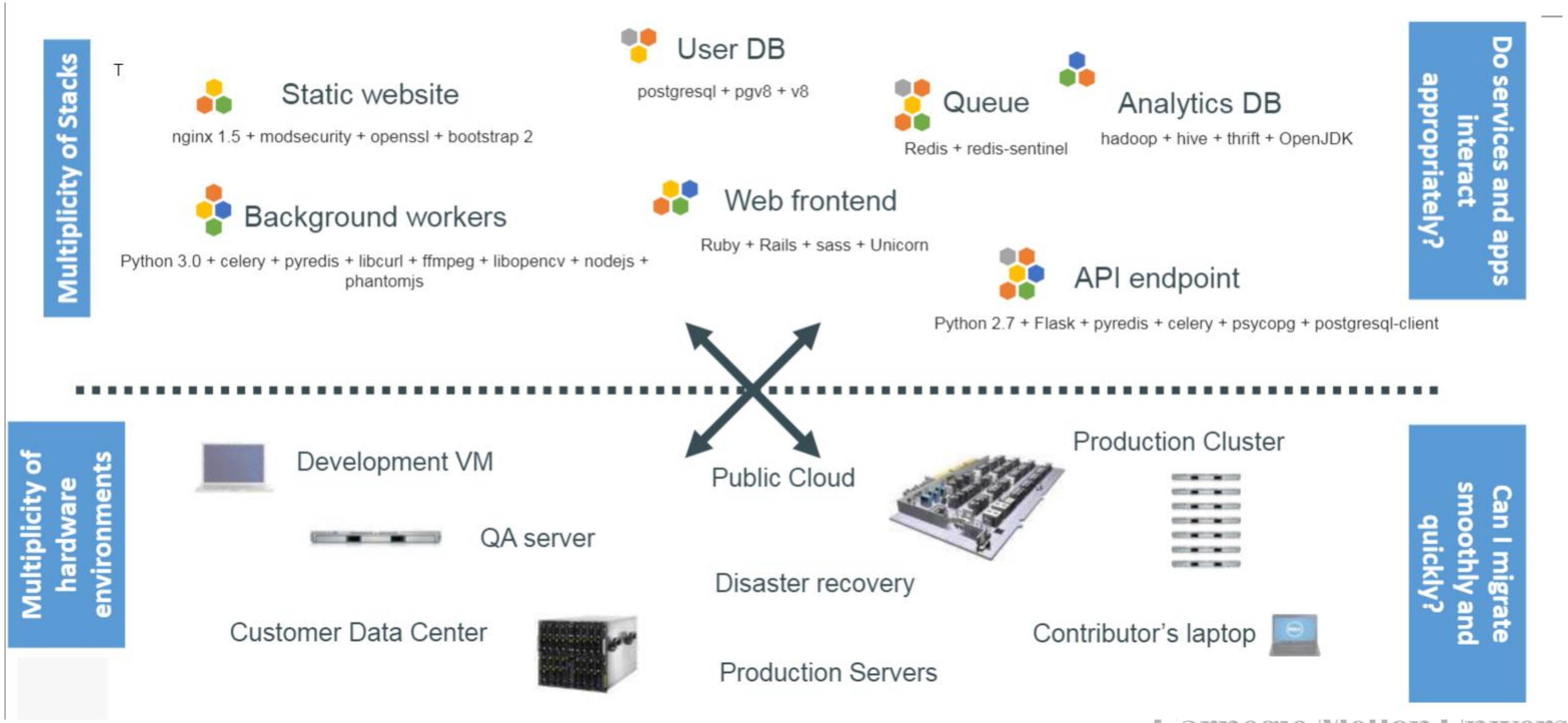
How to ensure services interact consistently, avoid dependency?

How to avoid "n x m" different configurations?

Running on any available resources

How to migrate & scale quickly, ensure compatibility?

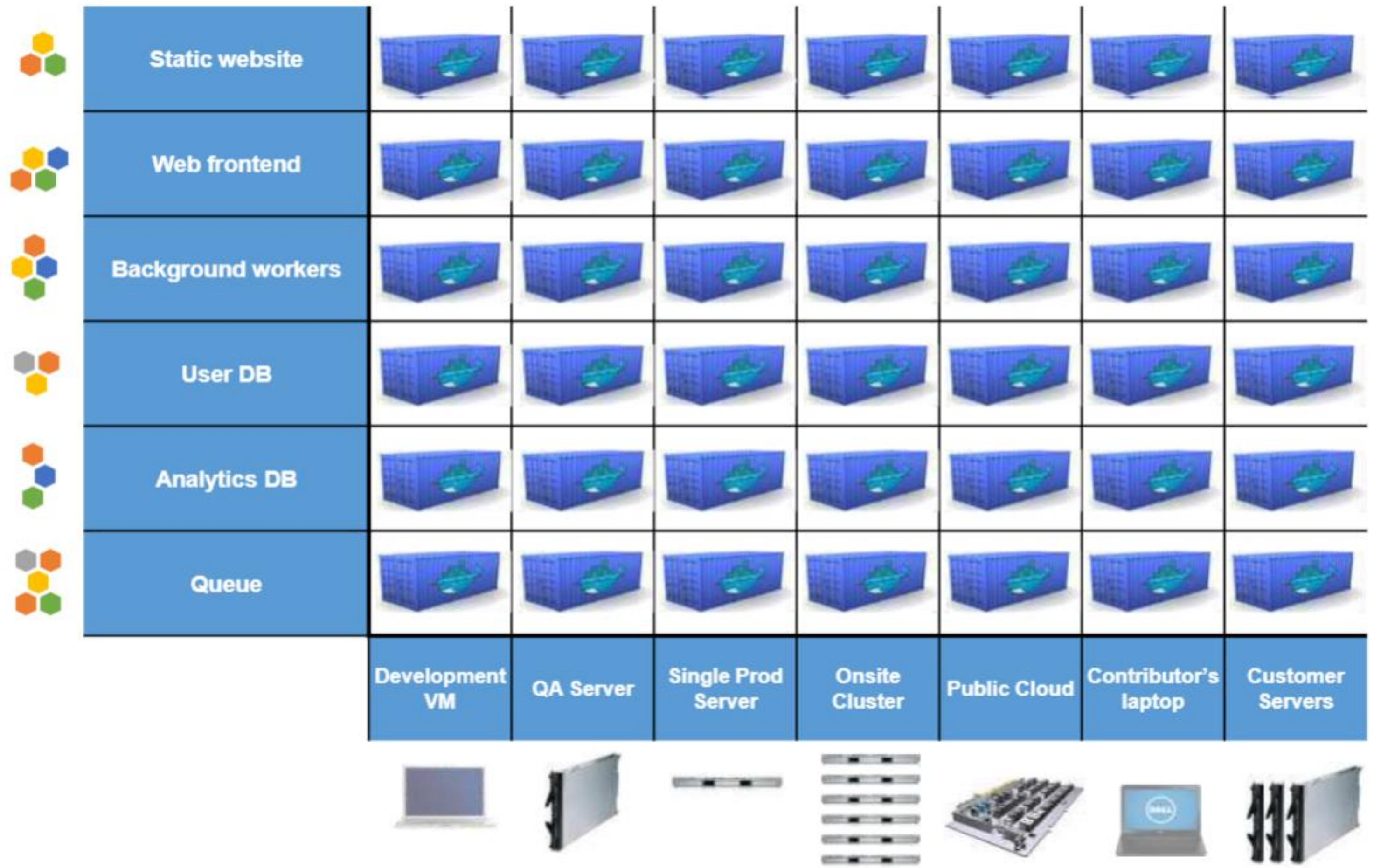
Development Challenges – Cont'd


Multiplicity of Stacks
T
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Do services and apps interact appropriately?
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Disaster recovery
Production Servers
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly?

Development Challenges – Cont'd
NxM Configurations

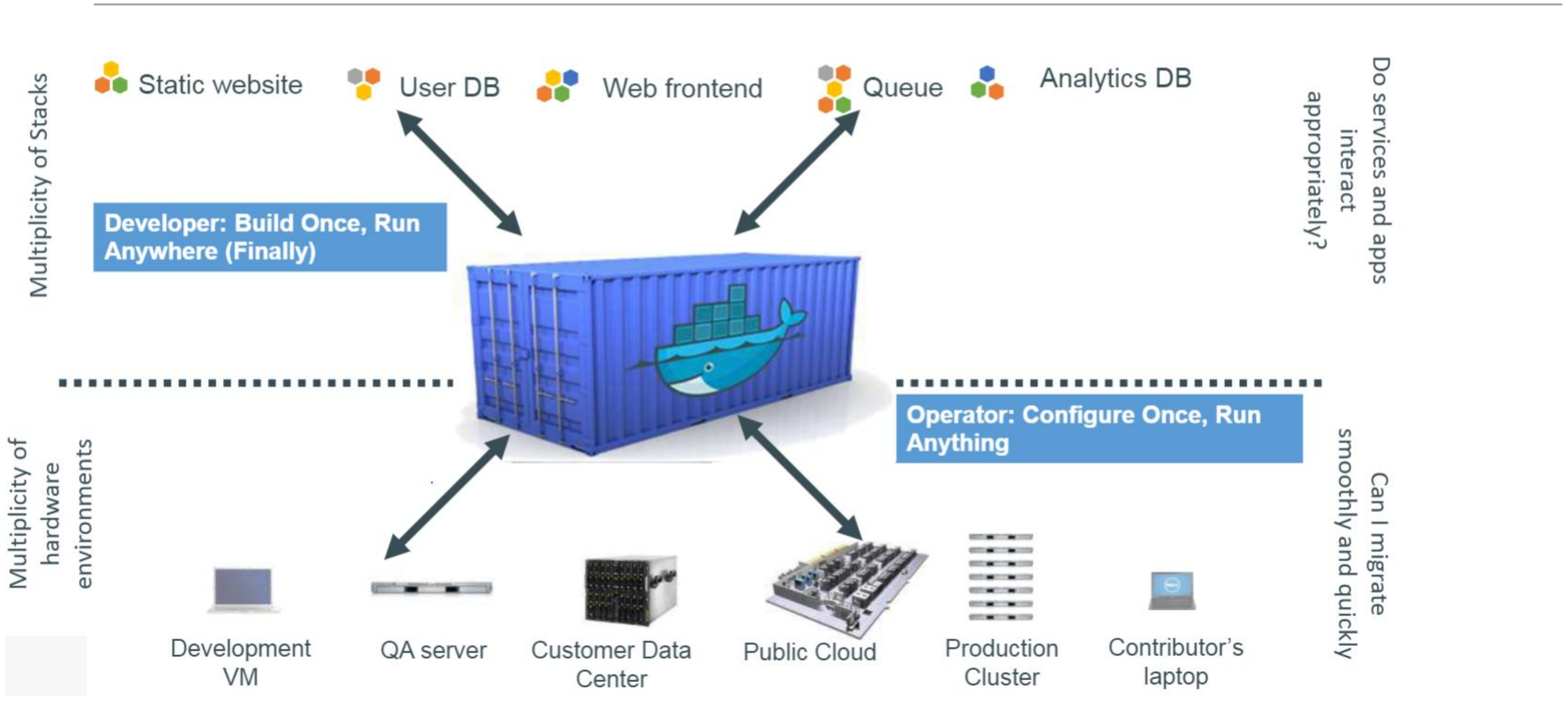
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

The Solution - Containerization



Carnegie Mellon University

The Solution – Cont'd



Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
Developer: Build Once, Run Anywhere (Finally)
Operator: Configure Once, Run Anything
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly

Docker – Leading Container

- Most widely known, and used
- Easy to download and install
- Free

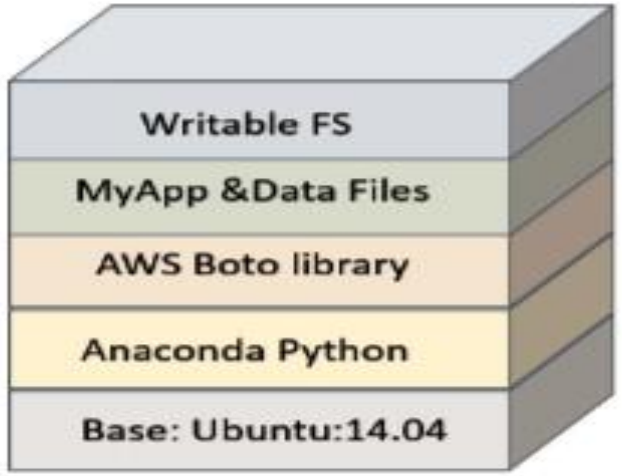
- By mid-2013, **dotCloud** company released a tool that used these ideas to provide a better way to deploy encapsulated applications
- The tool is later became **Docker**
- dotCloud became **Docker Inc.**

What makes Docker popular?!

- Ease of use
  - Command line, Docker compose, Kubernetes
- Speed
  - Load fast - sharing library among containers
- Docker Hub – for sharing images
  - http://hub.docker.com/
- Modularity and Scalability
- And .. its filesystem!!!

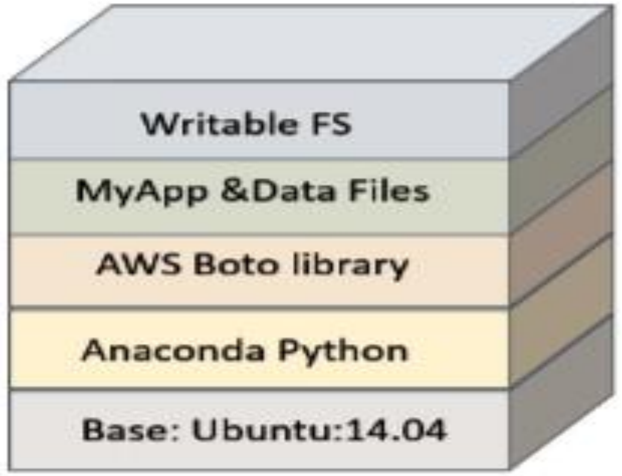
Docker’s Union File System (UFS)

- Union File Systems basically allow you to take different file systems and create a union of their contents with the top-most layer superseding any similar files found in the file systems
- Docker images are composed of layers in the Union File System. The image is itself a stack of read-only directories. The base is a simplified Linux file system.
    - Additional tools that the container needs are then layered on top of that base, each in its own layer.
- All containers with the same image see the same directory tree
    - Load the directory tree in the memory only once among all instances



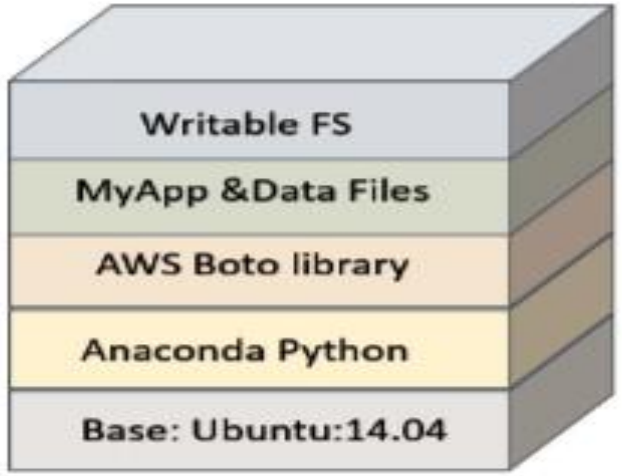
Docker's Union File System (UFS) – Cont'd

- When the container is run, a final writable file system is layered on top.
- As an application in the container executes, it uses the writable layer. If it needs to modify an object in the read-only layers, it copies those objects into the writable layer.
- Otherwise, it uses the data in the read-only layer, which is shared with other container instances.
- Thus, typically only a little of the container image needs to be actually loaded when a container is run, which means that containers can load and run much faster than virtual machines.
- In fact, launching a container typically takes less than a second, while starting a virtual machine can take minutes

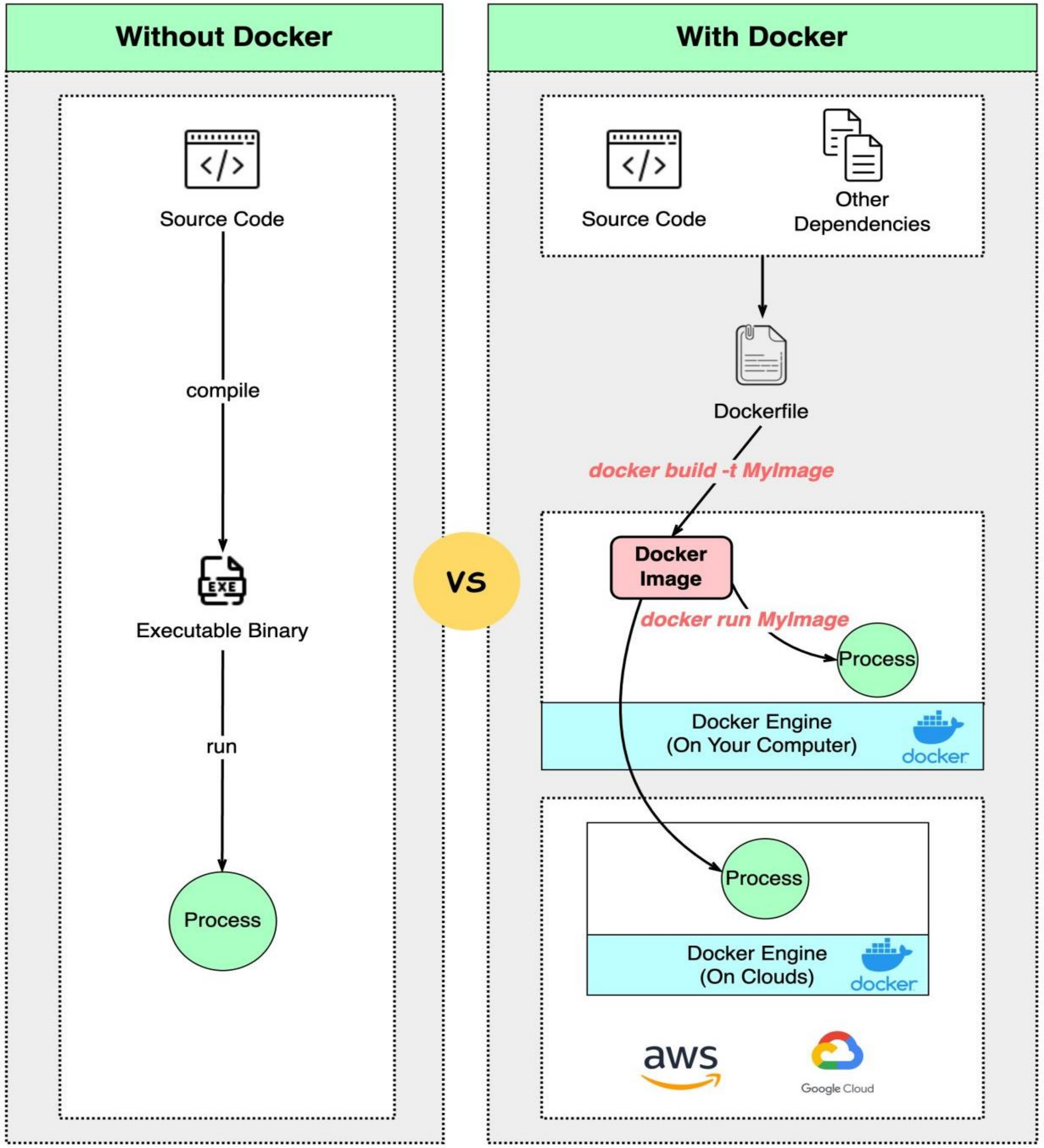


Docker's Union File System (UFS) – Cont'd

- In addition to the file system layers in the container image, you can mount a host machine directory as a file system in the container's OS.
- In this way, a container can share data with the host. Multiple containers can also share these mounted directories and can use them for basic communication of shared data



Docker Development Workflow



Getting Started with Docker

- Download Docker from https://docs.docker.com/get-started/ and install it on your machine.

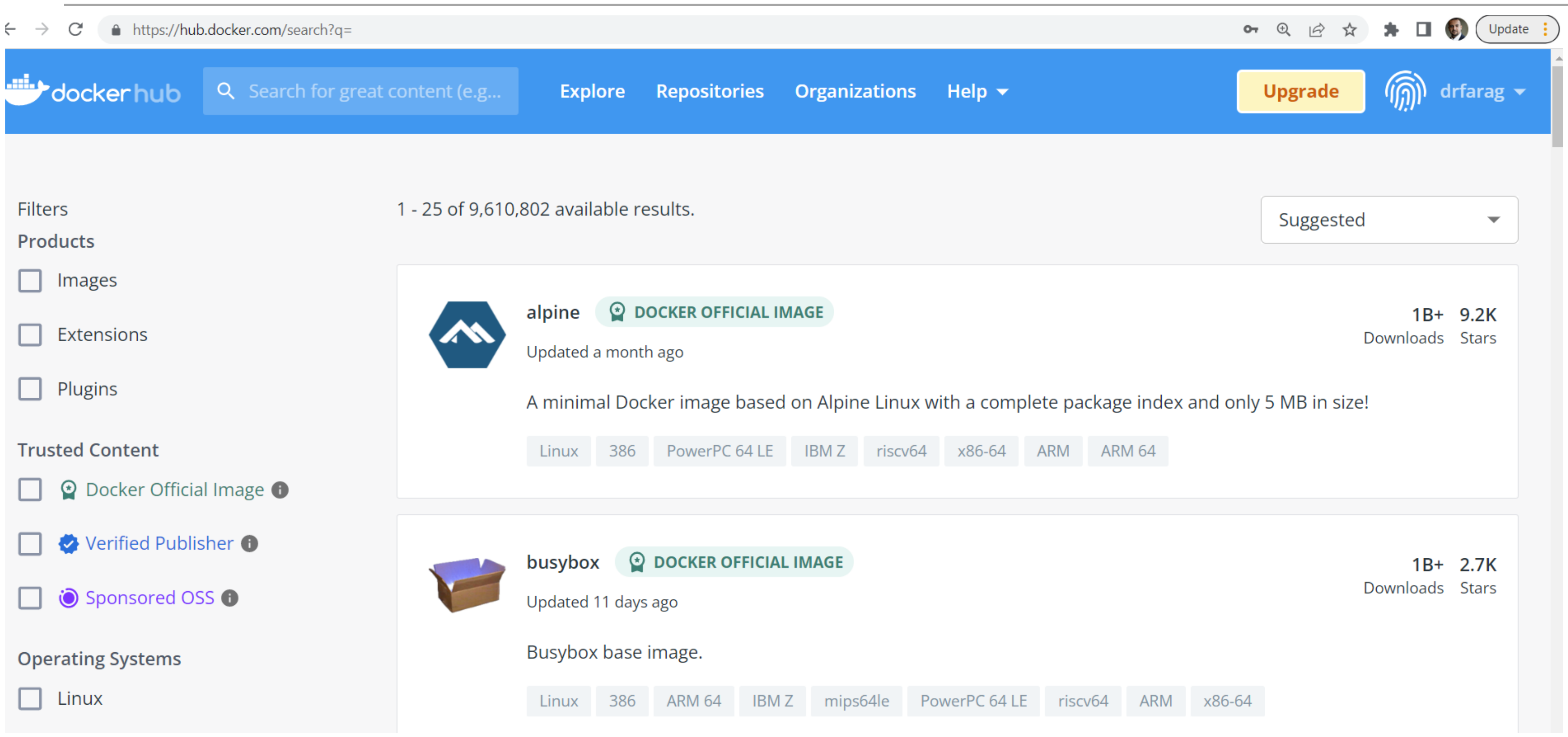
```
## List Docker CLI commands
docker
docker container --help
## Display Docker version and info
docker --version
docker version
docker info
## List Docker images
docker image ls
## List Docker containers (running, all)
docker container ls
docker container ls --all
```

Carnegie Mellon University

Do you need to write a Dockerfile every time you would like to use Docker?

No, you may pull images from Docker Hub!

Docker Hub


https://hub.docker.com/search?q=

docker hub | Search for great content (e.g... | Explore | Repositories | Organizations | Help | Upgrade | drfarag

Filters

Products

- [ ] Images
- [ ] Extensions
- [ ] Plugins

Trusted Content

- [ ] Docker Official Image
- [ ] Verified Publisher
- [ ] Sponsored OSS

Operating Systems

- [ ] Linux

1 - 25 of 9,610,802 available results.

Suggested

alpine DOCKER OFFICIAL IMAGE
Updated a month ago
1B+ Downloads 9.2K Stars

A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

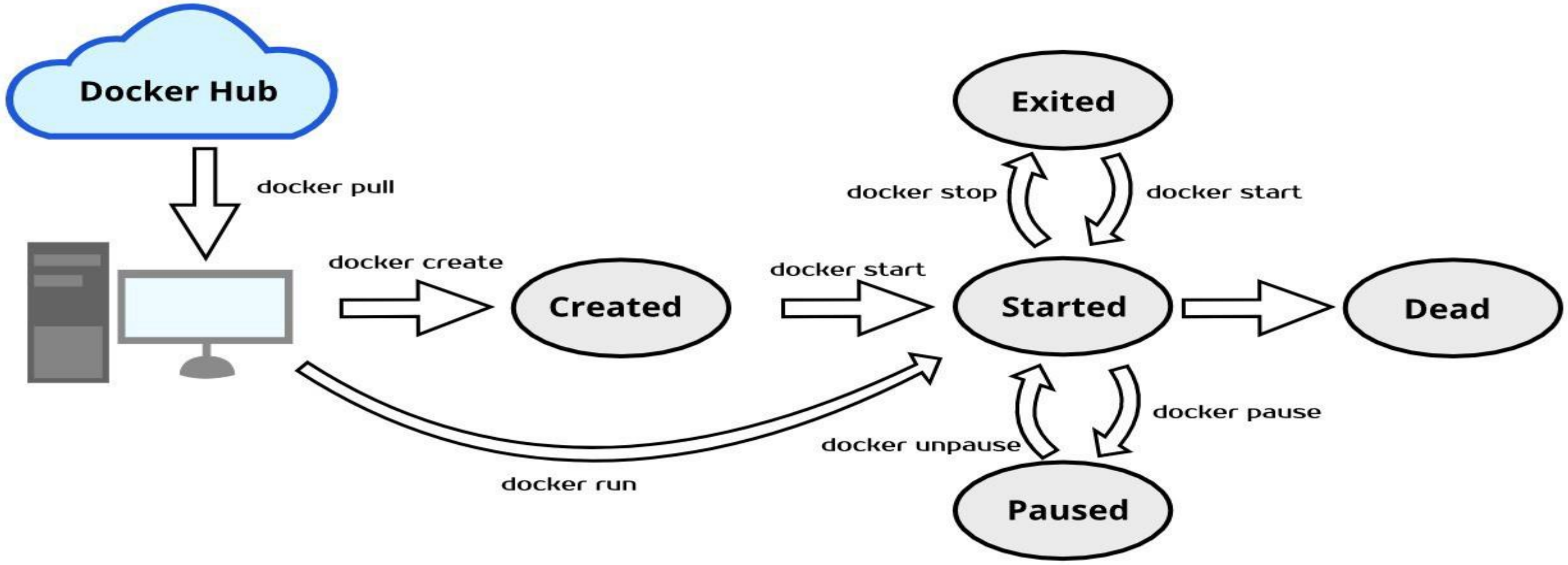
Linux | 386 | PowerPC 64 LE | IBM Z | riscv64 | x86-64 | ARM | ARM 64

busybox DOCKER OFFICIAL IMAGE
Updated 11 days ago
1B+ Downloads 2.7K Stars

Busybox base image.

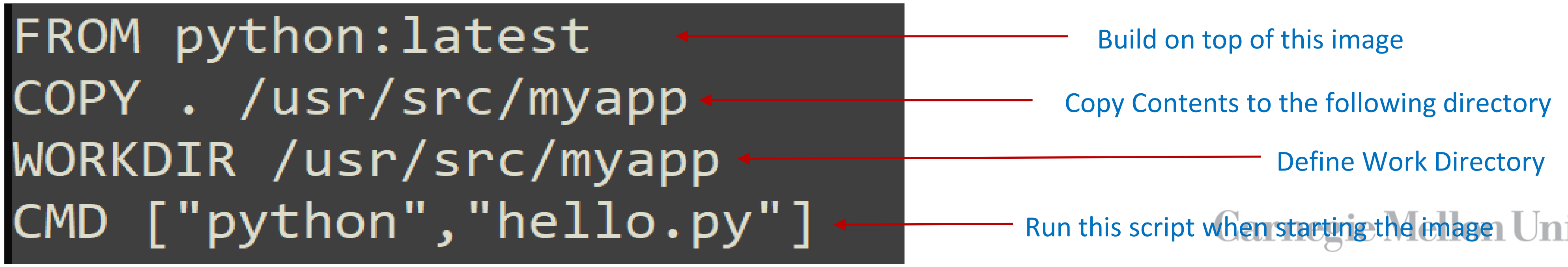
Linux | 386 | ARM 64 | IBM Z | mips64le | PowerPC 64 LE | riscv64 | ARM | x86-64

Docker Lifecycle Overview



How to write a Dockerfile?

- **Scan Docker Hub for an image that has most of what you need. No need to "reinvent the wheel".**
- **Find the correct version/tag of the image you would like to use as a base image.**
- **Create a dummy container of the Docker Image to understand its structure, the apps it needs, etc. Use " docker exec -it *containerId* "**
- **Create your Dockerfile (<u>with no file extensions</u>) to include any missing applications or commands that should be part of your final image.**
- **Build Your Image using " docker build " command (shown in a later slide)**
- **A sample Dockerfile is shown below:**



```
FROM python:latest
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
CMD ["python","hello.py"]
```

Build on top of this image
Copy Contents to the following directory
Define Work Directory
Run this script when starting the image

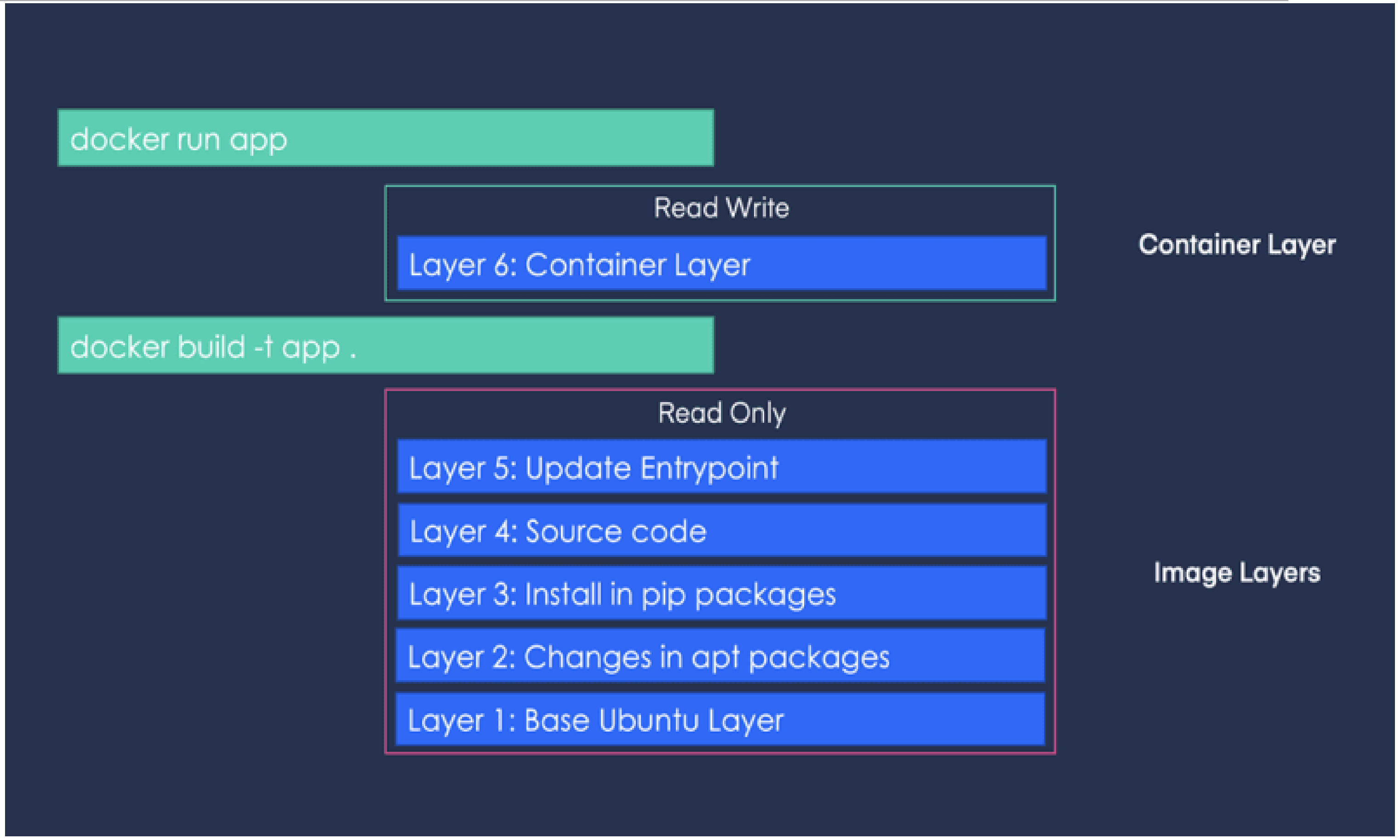
Carnegie Mellon University

What are the most important commands used in a Dockerfile?

Command	Description
FROM	The base image can be Ubuntu, Redis, MySQL, etc.
LABEL	Labeling like EMAIL, AUTHOR, etc.
RUN	used to tell the container what to do after creating the container from the image. Such as apt-get update, apt-get install, apk-add, etc.
COPY	Copy the files from our host system to a container destination path
ADD	like a COPY command, but it downloads tar, zip, or web file and extracts and copies inside of our image
WORKDIR	used to set the directory that we are going to work. If we are adding some files from host local machine and saves in the container, the working directory path is the default directory
EXPOSE	Documents which ports are exposed (It is only used for documentation)
ENV	Sets environment variables inside the image
ENTRYPOINT	The command that executes inside of the container when the container is started

Dockerfile vs. Docker Union Filesystem

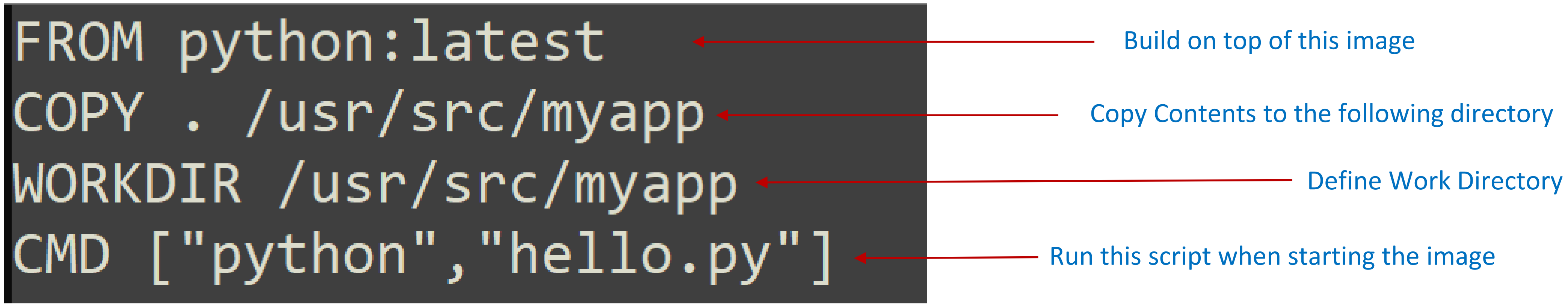
```
#Basic Dockerfile
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Lab: Run Your First Docker Container

We will run our first Hello World in Python

- Create a **hello.py** file that prints “Hello World” to the Screen.
- In the same directory where hello.py is stored, create your Dockerfile (with no extensions)
- Dockerfile content would look like this:



```
FROM python:latest
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
CMD ["python","hello.py"]
```

Build on top of this image
Copy Contents to the following directory
Define Work Directory
Run this script when starting the image

Lab: Build Your Image and Create Your Container

- Build Your Docker Image (and tag it) – Notice the period at the end

  docker build -t *yourDockerHubId/chooseName* **.**

- Test Your Image (you may not need the arguments depending on the case)

  docker run –d –p 8000:8000 *yourDockerHubId/chooseName*

  Or

  docker run *yourDockerHubId/chooseName*

- Upload to Docker Hub to share with others (make sure to have Docker Hub account)

  docker push *yourDockerHubId/chooseName*

You may need to perform "docker login" before pushing your image to Docker Hub

Lab: Run Your First Docker Container – Cont'd

Don't get confused with the following Docker commands:

RUN

- Executes command(s) and creates new image layers. E.g., it is often used for installing software packages.

CMD

- Sets default command and its parameters that gets executed when the container starts. This command can be overwritten from command line by passing parameters from docker run

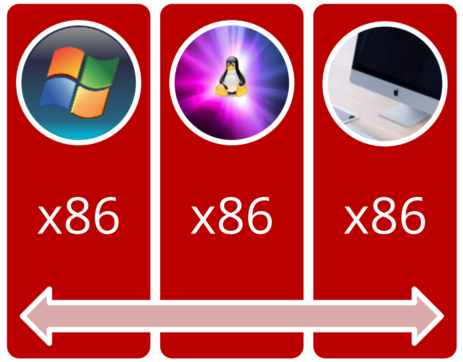
ENTRYPOINT

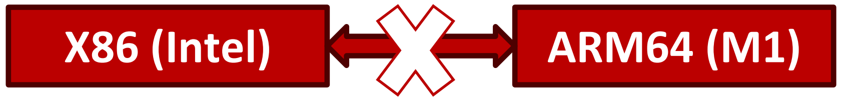
- Configures the command to run when the container starts. This command is similar to CMD from functionality perspective.
  You cannot override an ENTRYPOINT when starting a container unless you add the --entrypoint flag.

Carnegie Mellon University

Docker Limitations

- Docker offers cross-platform portability, but it does not offer cross-hardware-architecture portability





- Docker is not easy to be used with Desktop applications that require rich GUI
- It’s challenging to manage large number of docker containers **manually**.

What about M1/M2 Chip Users?

One of these strategies may work for you!

1. Enable Rosetta M1/M2 Emulation in your Docker Desktop Settings.
2. Search for Docker images that are tagged with arm64 or their documentation provides compatibility information with M1/M2 chips.
3. Use buildx tool to build your custom docker images that are compatible with your M1/M2 chip from the basic images that are compatible with x86 (intel) chips.
   - https://blog.jaimyn.dev/how-to-build-multi-architecture-docker-images-on-an-m1-mac/