Docker 101

IBM **Developer**

IBM

Welcome to Docker 101

We will start soon...

Pre-requirements:

Sign up for a Docker account at https://hub.docker.com/

Use your Docker login at https://labs.play-with-docker.com/

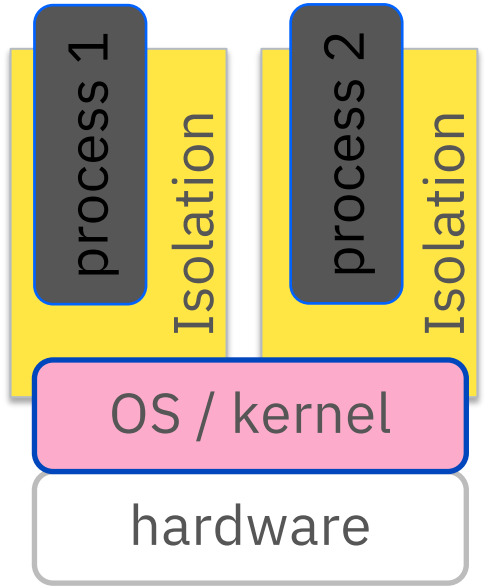
Labs are at https://github.com/IBM/intro-to-docker-lab

Table of Content

Introducing containers and Docker

Containers – not a new idea

- (1979) `chroot` command, changes apparent root for running process to isolate system processes
- (1982) `chroot` added to Unix v7
- (1990s) `jail` command created
- (2000) `jail` added to FreeBSD to isolate filesystems, users, networks etc.
- (2001) Linux VServer is a jail mechanism that partitions resources (file systems, network addresses, memory)
- (2004) Solaris Containers using Solaris Zones, application has a full user, processes, and file system, and access to the system hardware, but can only see within its own zone.
- (2005) Open VZ (Virtuzzo), OS-level virtualization for Linux using a patched Linux kernel for virtualization, isolation, resource management and checkpointing
- (2006) Google launches `Process Containers` for limiting resource usage, renamed `cgroups` in 2007
- (2008) `cgroups` merged into Linux kernel 2.6.24, becoming LinuX Containers (LXC)
- (2009) Cloud Foundry developed by VMWare called Project B29 (Warden, Garden)
- (2011) Cloud Foundry started Warden
- (2013) Docker released as open source
- (2014) LXC replaced by `libcontainer`, Google contributes container stack `LMCTFY` (Let Me Contain That For You) to Docker project as `libcontainer` in 2015



What are Containers?

Similar to VMs but managed at the **process level**
"VM-like" isolation achieved by set of "**namespaces**" (isolated view)

- PID –isolated view of process IDs
- USER- user and group IDs
- UTS - hostname and domain name
- NS - mount points
- NET - Network devices, stacks, ports
- IPC - inter-process communications, message queues

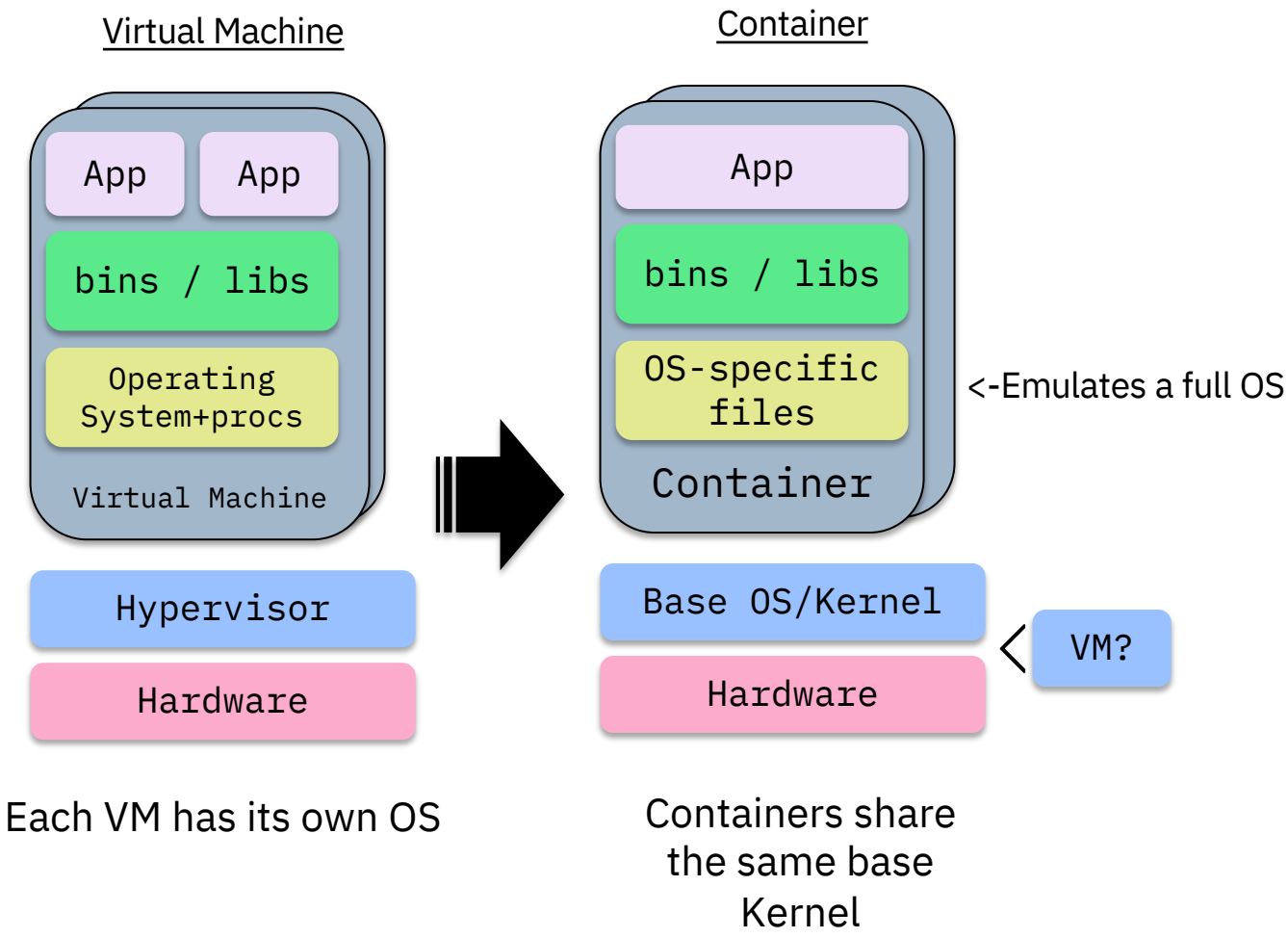
cgroups - controls limits and monitoring of resources

The key statement: **A container is a process(es) running in isolation**

VM vs Container

Before containers, OS environment:

- Entanglement between OS and App
- Entanglement between runtime environment and OS
- Version dependency and conflicts between Apps
- Breaking updates
- Updates require full app stop and downtime
- Deployment and Maintenance of HA system is complex
- Slow startup time



Each VM has its own OS

Containers share the same base Kernel

Containers vs Docker

Containers is the technology, Docker is the **tooling** around containers

Without Docker, containers would be **hard to use** (for most people)

Docker **simplified** container technology

Added value: Lifecycle support, setup file system, etc

For extra confusion: **Docker Inc.** is a company, which is different than Docker the technology…

Docker's ecosystem approach transformed the perception of containers,
- Building application-centric containers
- Mechanism for sharing images (Docker Registry)
- Open-source enabled

Our First Container

```
$ docker run ubuntu echo Hello World
Hello World
```

What happened?

- Docker created a directory with a "ubuntu" filesystem (image)
- Docker created a new set of namespaces
- Ran a new process: echo Hello World
  - Using those namespaces to isolate it from other processes
  - Using that new directory as the "root" of the filesystem (chroot)
- That's it!
  - Notice as a user I never installed "ubuntu"
- Run it again - notice how quickly it ran

"ssh-ing" into a container

```
$ docker run -ti ubuntu bash
root@62deec4411da:/# pwd
/
root@62deec4411da:/# exit
$
```

- Now the process is "bash" instead of "echo"
- But its still just a process
- Look around, mess around, its totally isolated
  - rm /etc/passwd – no worries!
  - MAKE SURE YOU'RE IN A CONTAINER!

A look under the covers

```
$ docker run ubuntu ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 14:33 ?        00:00:00 ps -ef
```

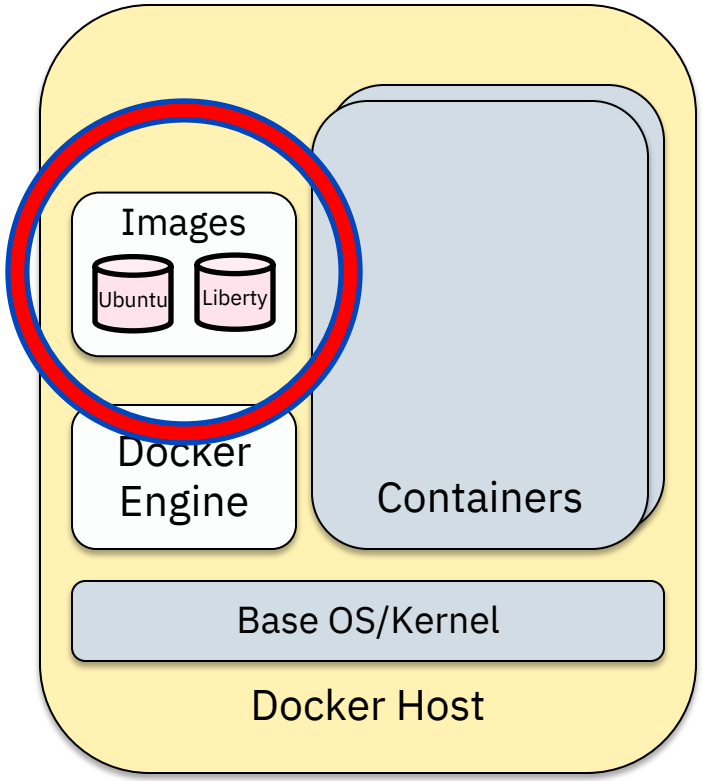
Things to notice with these examples:

- Each container only sees its own process(es)
- Each container only sees its own filesystem
- Running as "root"
- Running as PID 1

Docker Images

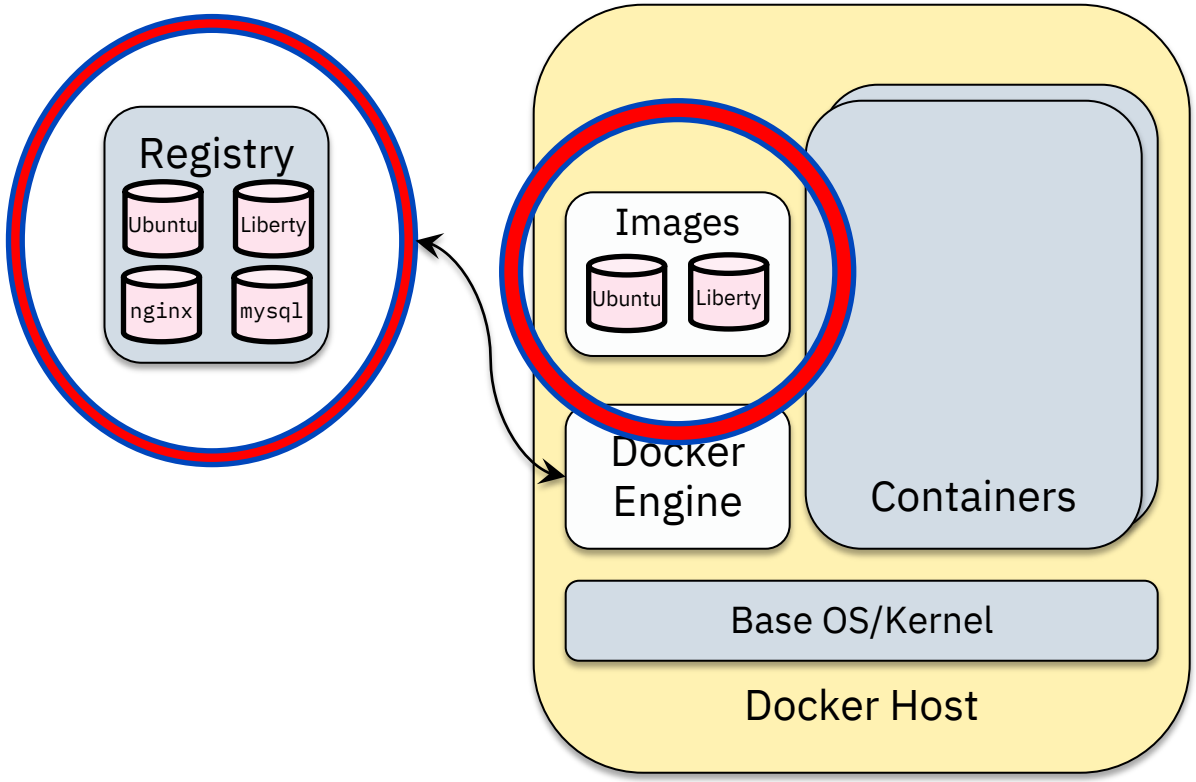
- Tar file containing a container's filesystem + metadata

- For sharing and redistribution
  - Global/public registry for sharing: DockerHub



Docker Registry

- DockerHub (https://hub.docker.com)
- Public registry of Docker Images
- The central place for sharing images with friends or coworkers!
- Also useful to find prebuilt images for web servers, databases, etc
- Enterprises will want to find a private registry to use (such as Artifactory)



Build your own image with a Dockerfile!

Step 1) Create Dockerfile to script how you want the image to be built

```
FROM java:8 # This might be an ubuntu or...
COPY *.jar app.jar
CMD java -jar app.jar
```

Step 2) **docker build** to build an image

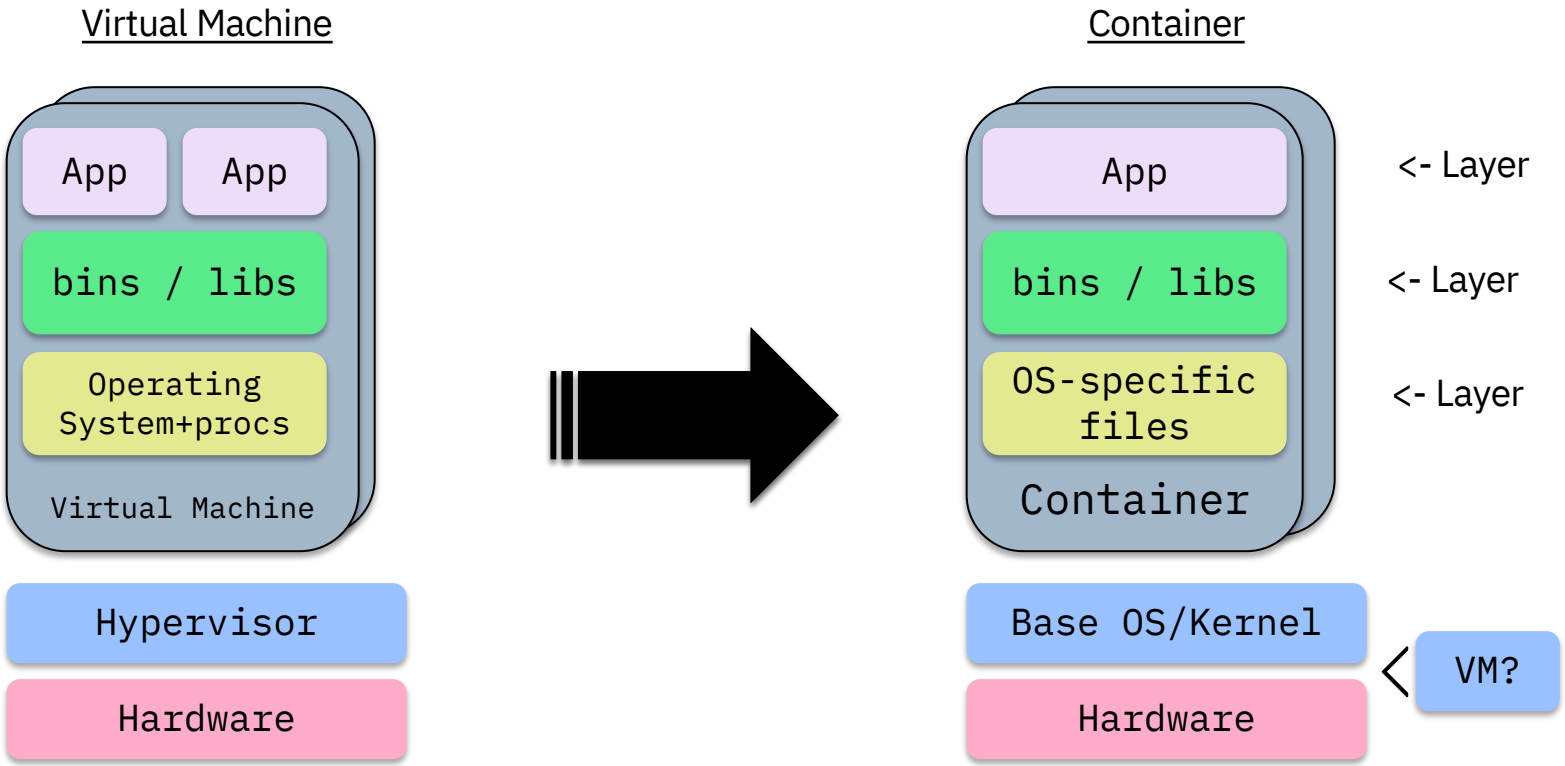
Step 3) **docker push** to push to registry

Step 4) From another location, **docker pull** to download an image

Docker special sauce: Layers

Let's compare VMs and Containers one more time...

VM vs Container: Notice the layers!



Each VM has its own OS

Containers share the same base Kernel

Shared/Layered/Union Filesystems

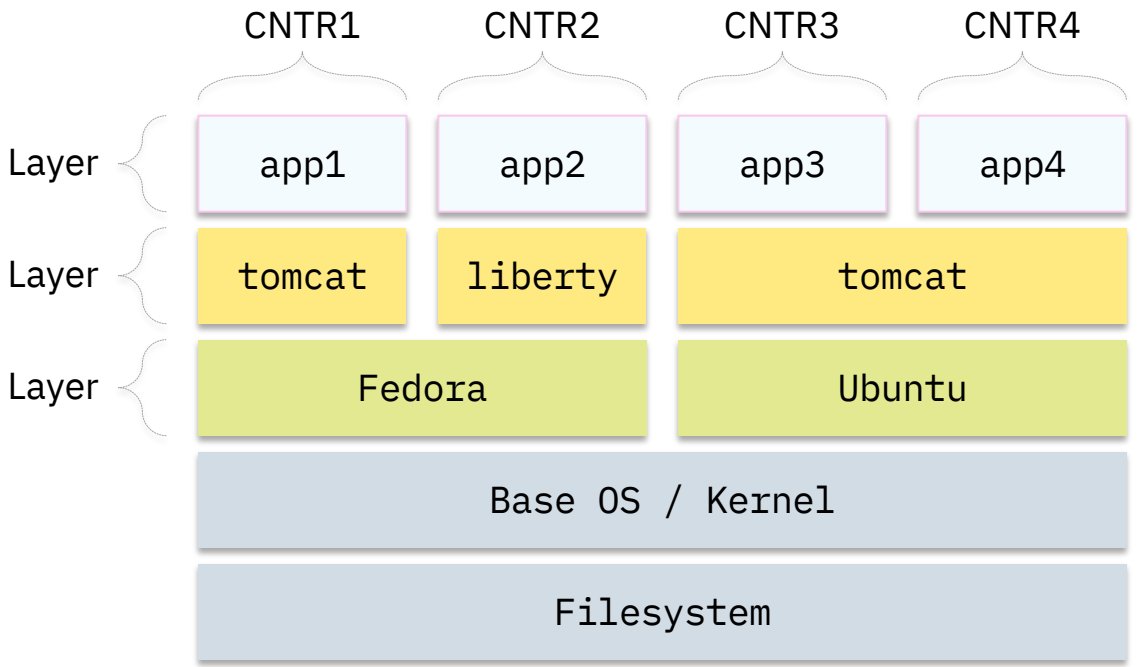
- Docker re-uses common layers between containers and images
- A single writeable layer is added on the top every time a new layer is created
- Layers are “smushed” with **union file system** (think transparencies on a projector)
- Files are copied up when writes need to be made (copy-on-write)

Bottom Line

- More containers per host
- Faster downloads and uploads
- Faster container startups

ls /var/lib/docker/overlay2

```
0016ac03f0de110bd315ea3cd03546d4192ddd6a4a4c75ea1908c7edee69e9d3
0016ac03f0de110bd315ea3cd03546d4192ddd6a4a4c75ea1908c7edee69e9d3-init
16a28614760c68941fbd193fad753965943d35de3dfe5ebe059a1ba6d770fc10
37b3b057d9f04a0849dd74c3183604204e2bb745bd88d3b6c429a520fad8fb45
37d747c4f41f29b31664e357c5fde674025f9782c3336f29f4dcc2c85df15718
a5473075b9d58c609e45b0c226c2cf0495285a93898a2c8a478a2068c74630d1
l
```



Summary

Why? When compared to VMs:

- Low hardware footprint, better resource utilization (CPU, Memory, Disk – resources managed using namespaces and cgroups),
- Faster start-up times, no OS install or restart
- Quick deployment
- Better efficiency, elasticity, and reusability of the hosted applications
- Better portability of platform and containers
- Environment isolation, changes to host OS do not affect the container
- Multiple environment deployment
- Reusability
- Easier tooling/scripting

Docker value-add:

- User Experience
- Image layers
- Easily share images - DockerHub

Summary

- Docker is a tool to manage containers
  - Key concepts: Containers, Engine, Images, Registry

- Docker value-add:
  - An excellent User Experience
  - Image Layers
  - Easily shared images - DockerHub

- Why? When compared to VMs:
  - Better resource utilization - CPU, Memory, Disk
  - Faster start-up times
  - Easier tooling/scripting

- Discussion / Questions?

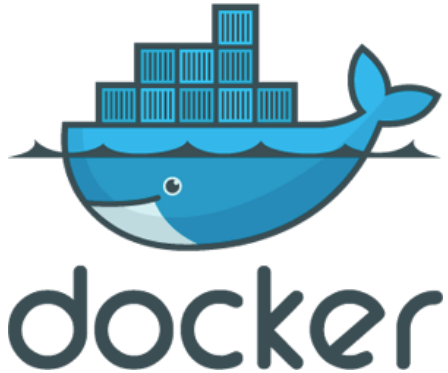
In a Traditional Deployment…

Are you testing these on ever commit?

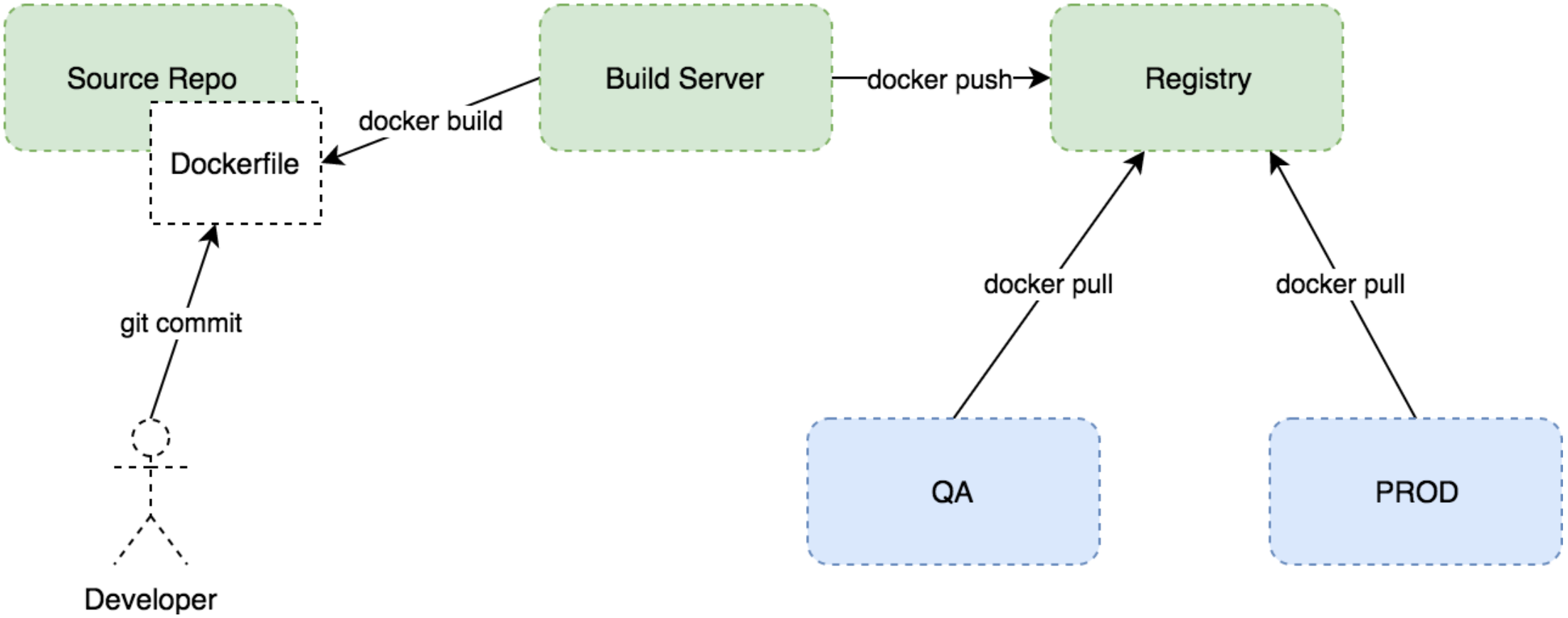
- Code (packages archive) ✓
- App server ✗
- Runtime versions ✗
- System libraries and versions ✗

Container = Code + Dependencies

- Code (packages archive)
- App server
- Runtime versions
- System libraries and versions



Container Life-Cycle



Lab Time