Intro to Docker Containers

John Zaccone

john.zaccone@ibm.com

IBM **Developer**

Agenda

- What are containers?
- Running our first container (hands-on)
- What are Docker images?
- Building a custom Docker Image (Lab)

Need technology layers diagram

What are containers?

(and how are they different then Docker?)

Introducing containers and Docker

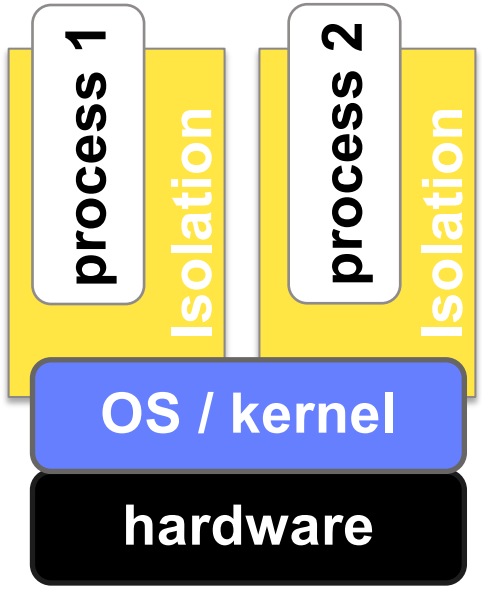
Containers – not a new idea

- chroot ('80s) process spawned in isolated file space
- FreeBSD jails
- OS-level virtualization (user-mode-linux, virtuozzo)
- Solaris Containers
- LinuX Containers (LXC)
- Cloud Foundry (Warden, Garden)

More efficient than VMs but less mindshare...

Docker – ecosystem approach transformed perception

- application container image
- Docker Hub registry (public) for sharing images
- engine widely available



What are Containers?

Similar to VMs but managed at the **process level**

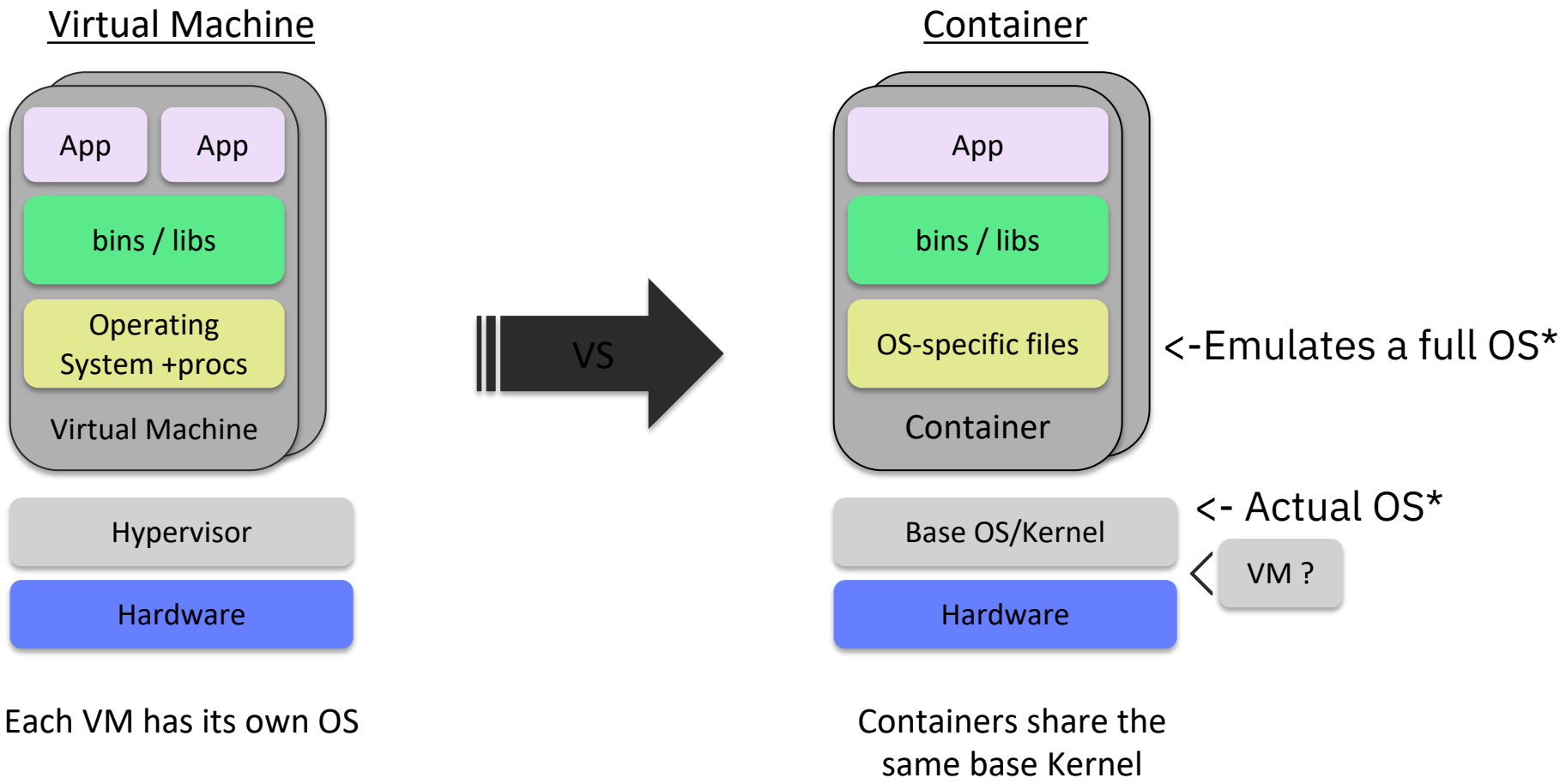
"VM-like" isolated achieved by set of "**namespaces**" (isolated view)

- 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



What is Docker?

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

Without Docker, containers would be **unusable** (for most people)

Docker **simplified** container technology to enable it for the masses

Added value: Lifecycle support, setup file system, etc

For extra confusion: Docker is also a company, which is different then Docker the technology...

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 - fake it...

```
$ 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

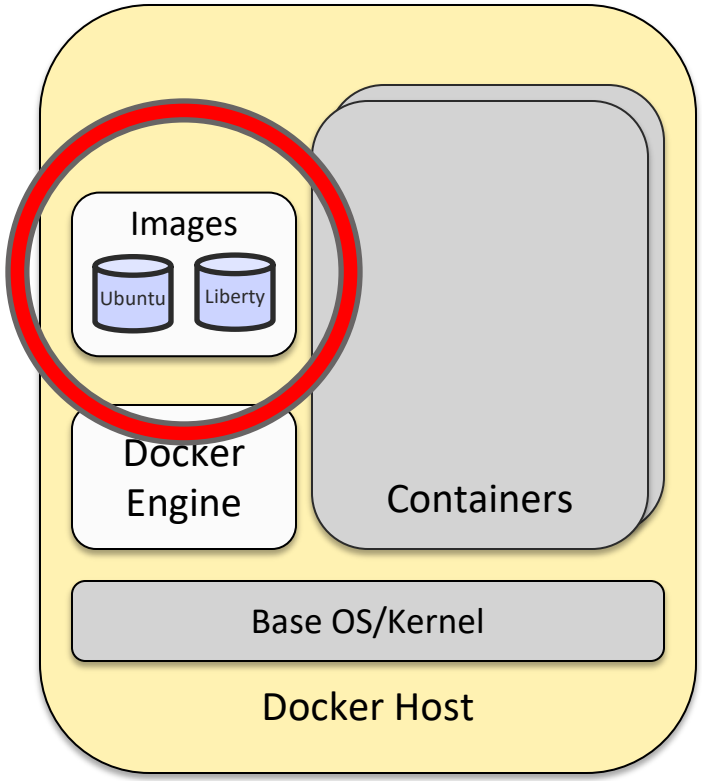
- Each container only sees its own process(es)
- Each container only sees its own filesystem
- Running as "root"
- Running as PID 1

Docker Images

Tar file containing a container's filesystem + metadata

For sharing and redistribution

– Global/public registry for sharing: DockerHub



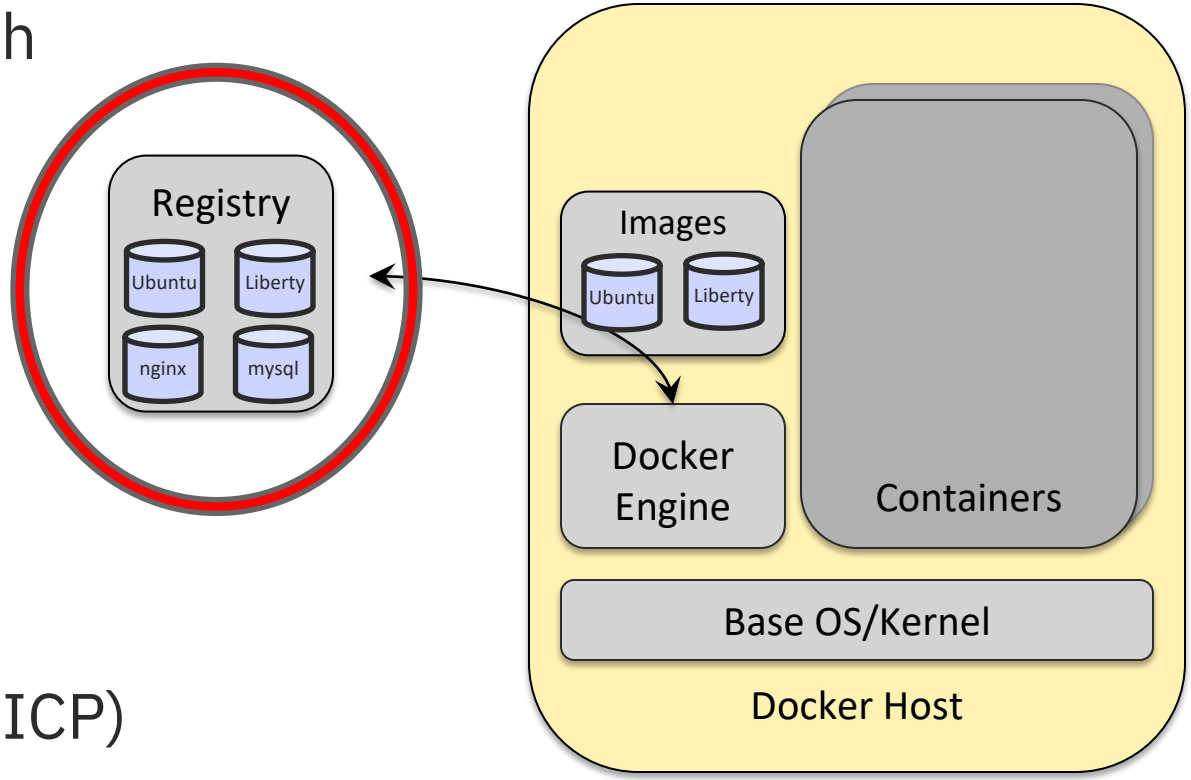
Docker Registry

The central place to share images with friends! (or coworkers)

DockerHub - http://hub.docker.com

- Public registry of Docker Images
- Also useful to find prebuilt images for web servers, databases, etc

Enterprises will want to find a private registry to use (such as one built into ICP)



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

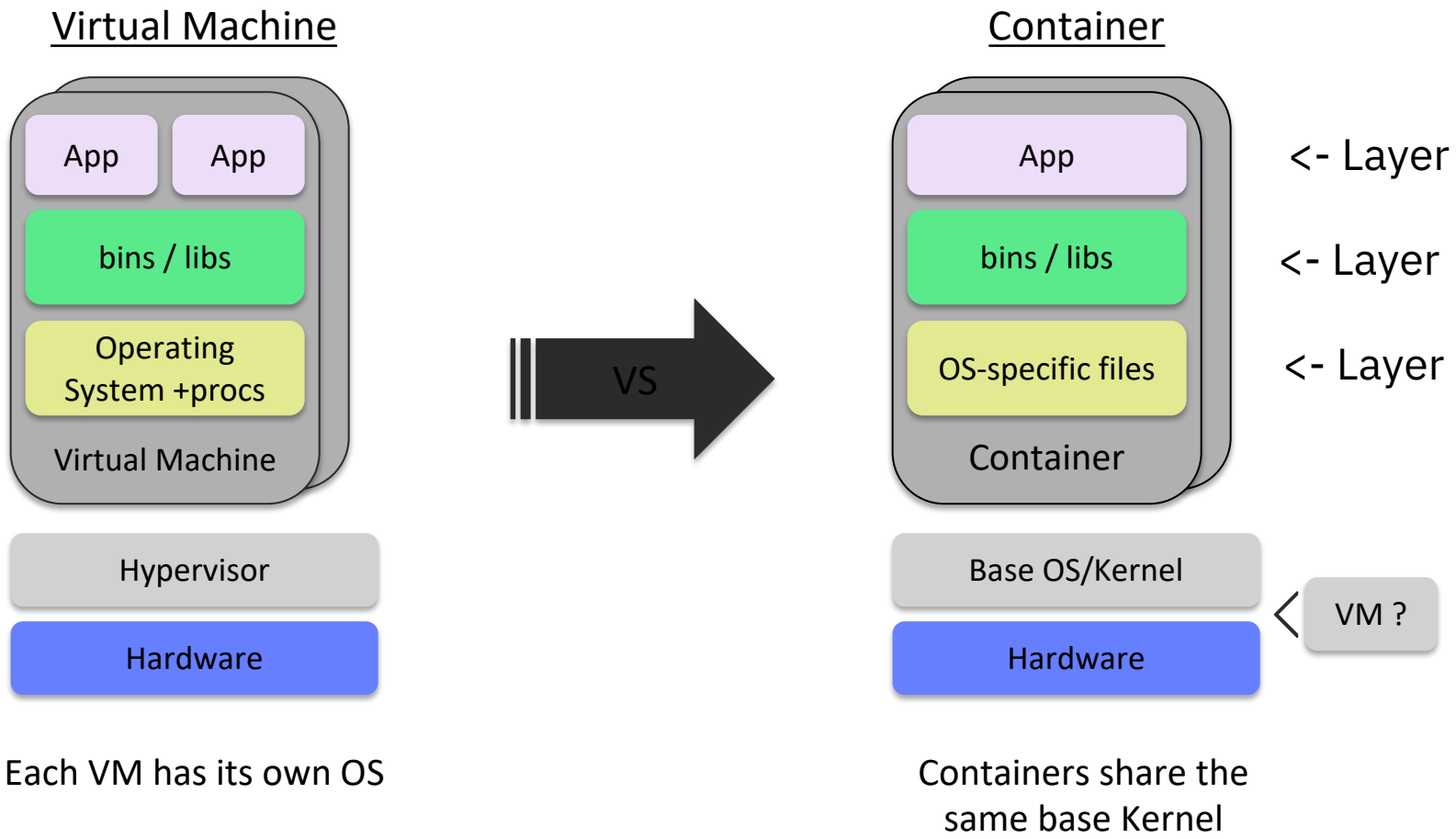
Step 3) **docker push** to push to registry

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

Docker special sauce: Layers

But first, 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 uses a copy-on-write (union) filesystem

New files(& edits) are only visible to current/above layers

Layers allow for reuse

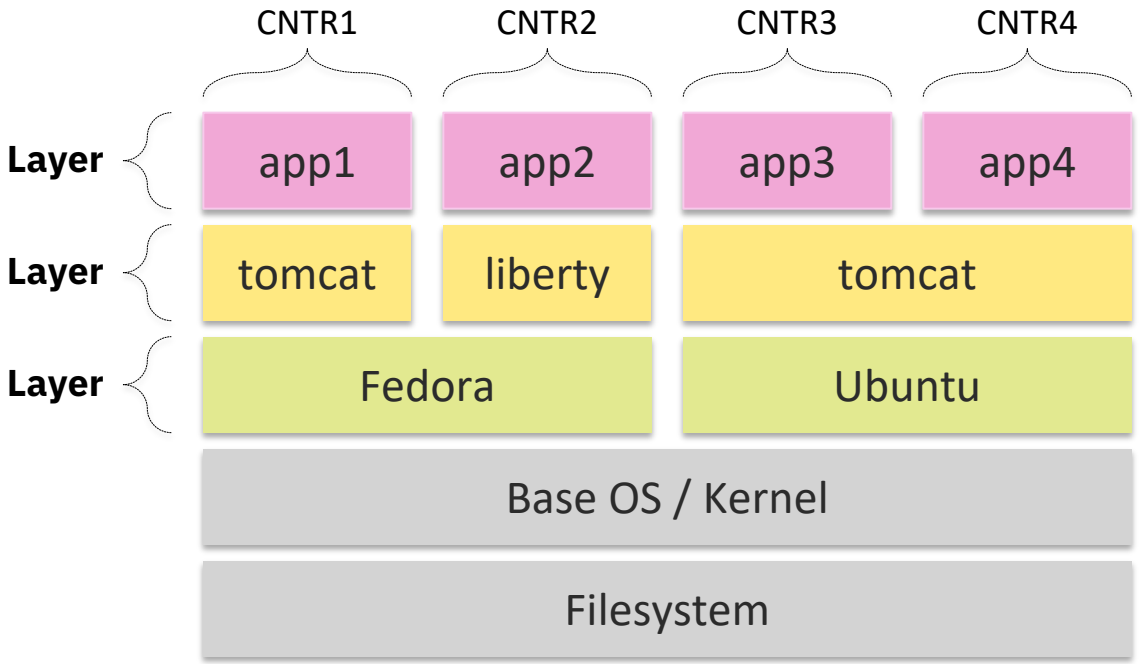
– More containers per host

– Faster start-up/download time

Images

– Tarball of layers

Think: Transparencies on projector



Summary

Docker is just 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?

Quiz!

What's the difference between a container and an image?

Answer:

- An image is a tar of a filesystem
- A container is a filesystem + a set of processing running in isolation

Lab– build your own image

ibm.biz/BdjJXT

Questions?

John Zaccone

john.zaccone@ibm.com