IBM

Open by design™

Docker——一种全新的工作方式

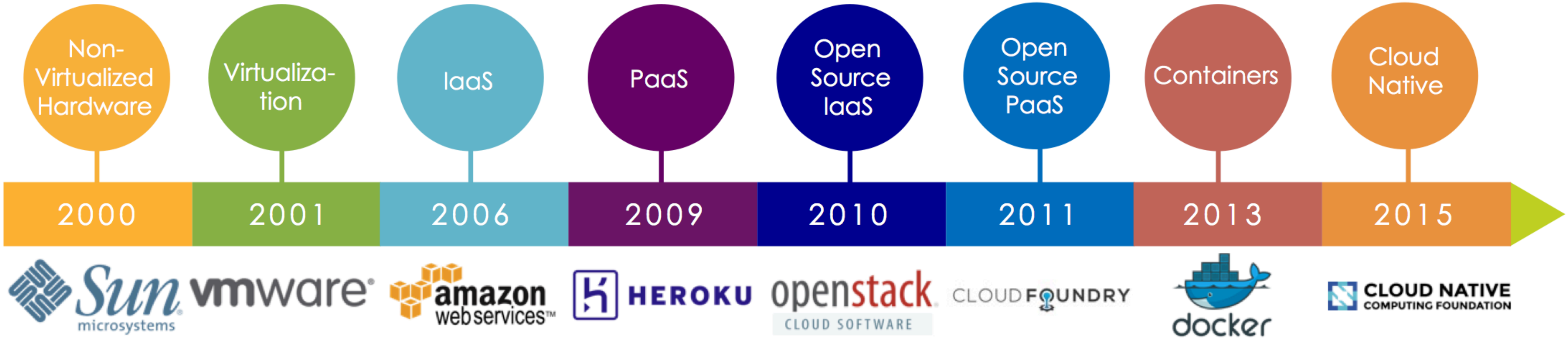
“容器技术和微服务”系列公开课

- 每周四晚8点档
  - Docker——一种全新的工作方式
  - 容器编排工具Docker Swarm
  - 数据中心操作系统的内核——Apache Mesos
  - 大数据、Web服务、CI/CD：一个都不能少——深入理解Mesos的资源调度及使用案例
  - Kubernetes实践
  - 各取所长——Kubernetes on Mesos
  - 微服务平台端到端业务解决方案
  - 事件驱动无服务器平台OpenWhisk

Agenda

- Introduction to Containers & Docker
- DevOps: Configuration Management
- DevOps: Building Value
- Container Orchestration
- Final Thoughts and Beyond Docker
- Summary

A short history of Cloud



What is Docker?

- At its core, Docker is tooling to manage containers
  - Docker is not a technology, it’s a tool or platform
  - Simplified existing technology to enable it for the masses
- But, let's first discuss containers...

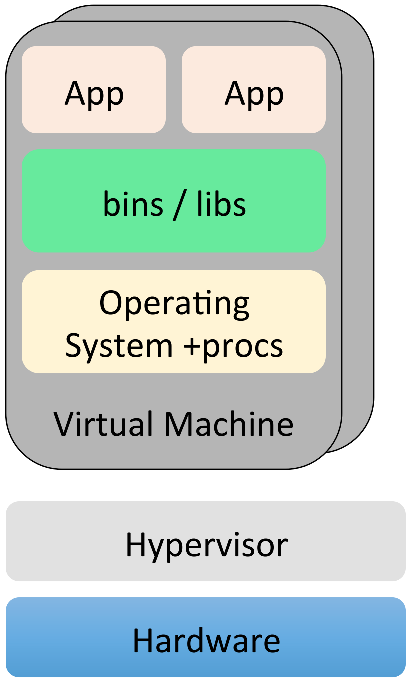
What are Containers?

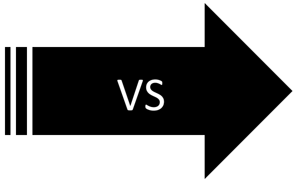
- A group of processes run in isolation
  - Similar to VMs but managed at the process level
  - All processes MUST be able to run on the shared kernel
- Each container has its own set of "namespaces" (isolated view)
  - **PID** - process IDs
  - **USER** - user and group IDs
  - **UTS** - hostname and domain name
  - **NS** - mount points
  - **NET** - Network devices, stacks, ports
  - **IPC** - inter-process communications, message queues
  - **cgroups** - controls limits and monitoring of resources
- Docker gives it its own root filesystem

IBM Open by design™

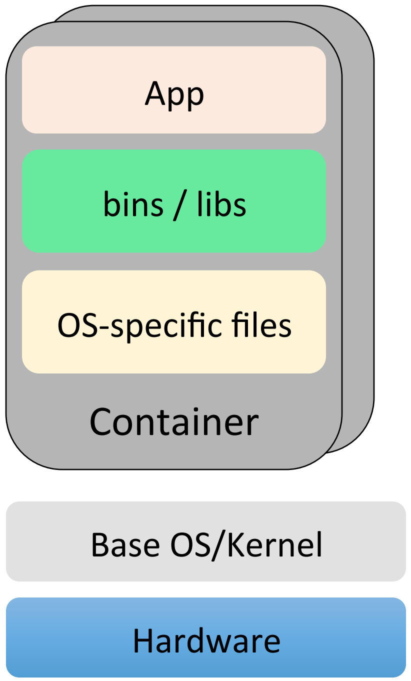
VM vs Container

Virtual Machine





Container



App, bins/libs/OS must all be runnable on the shared kernel

If OS files aren't needed they can be excluded.

Each VM has its own OS

Containers share the same base Kernel

Why Containers?

- Fast startup time - only takes milliseconds to:
  - Create a new directory
  - Lay-down the container's filesystem
  - Setup the networks, mounts, ...
  - Start the process

- Better resource utilization
  - Can fit far more containers than VMs into a host

What is Docker again?

- **Tooling** to manage containers
  - Containers are not new
  - Docker just made them easy to use
- Docker creates and manages the lifecycle of containers
  - Setup filesystem
  - CRUD container
    - Setup networks
    - Setup volumes / mounts
    - Create: start new process telling OS to run it in isolation

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
/
```

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

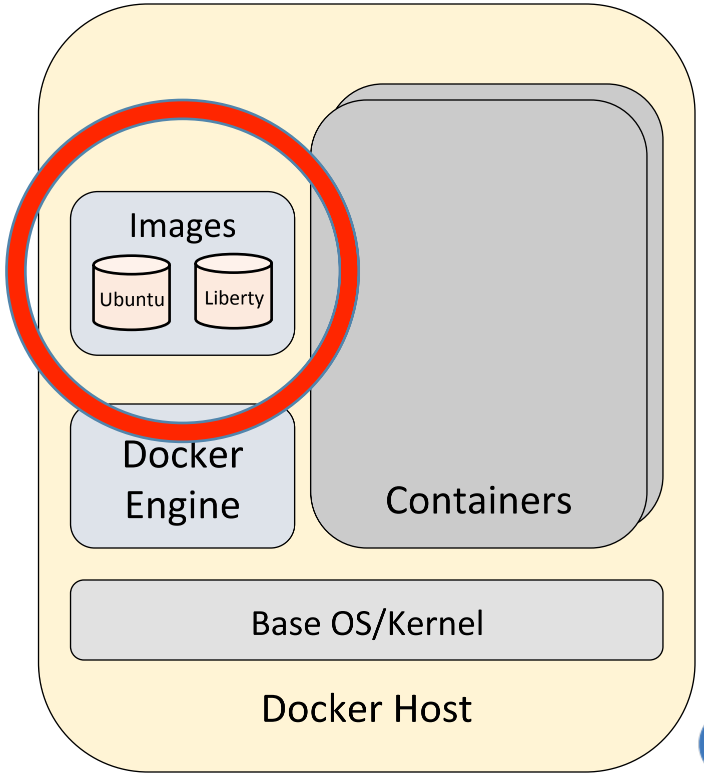
A look under the covers

```
$ 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
- Similar, in concept, to a VM image

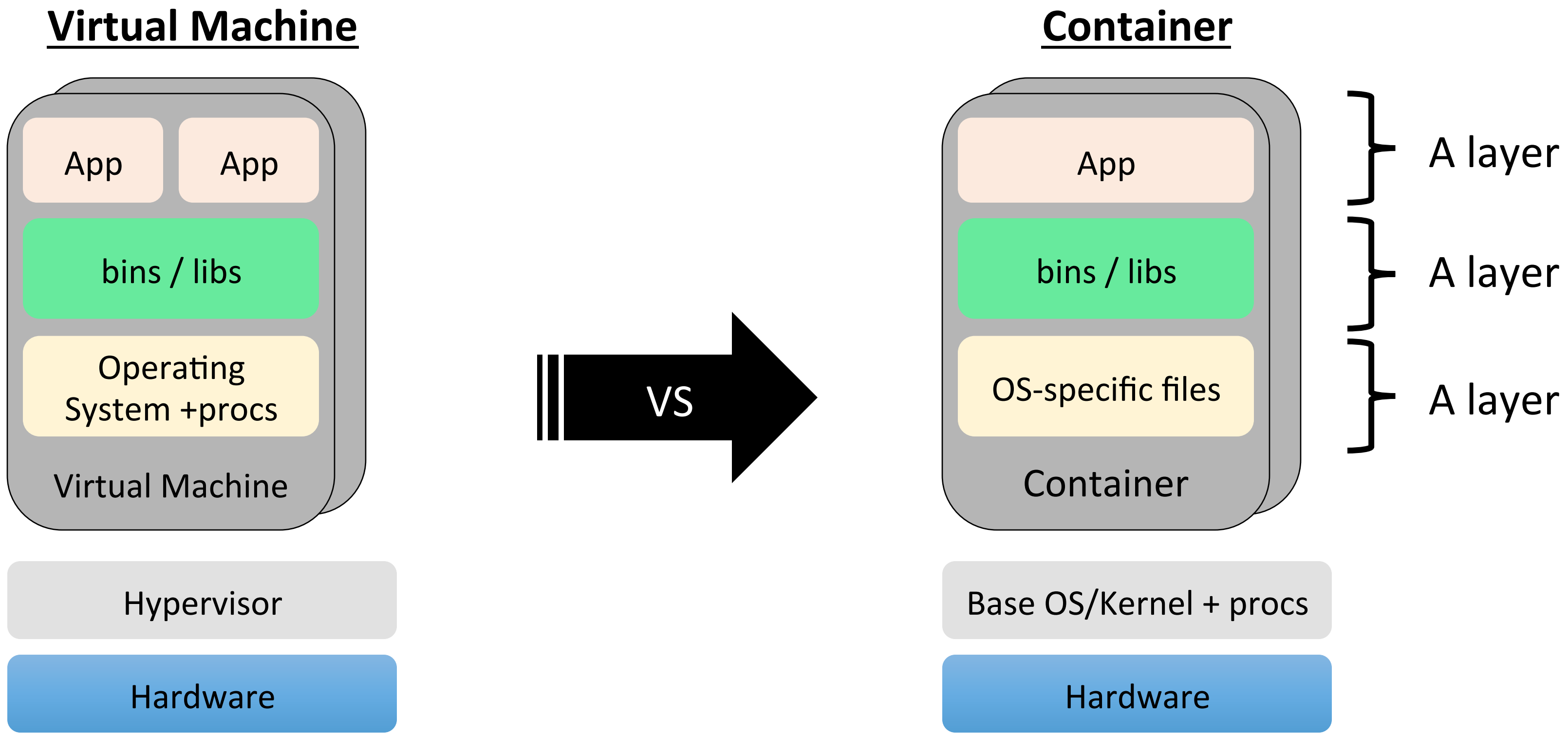


Docker special sauce: Layers

- But first, let's compare VMs and Containers one more time...

IBM Open by design™

VM vs Container: Notice the layers!

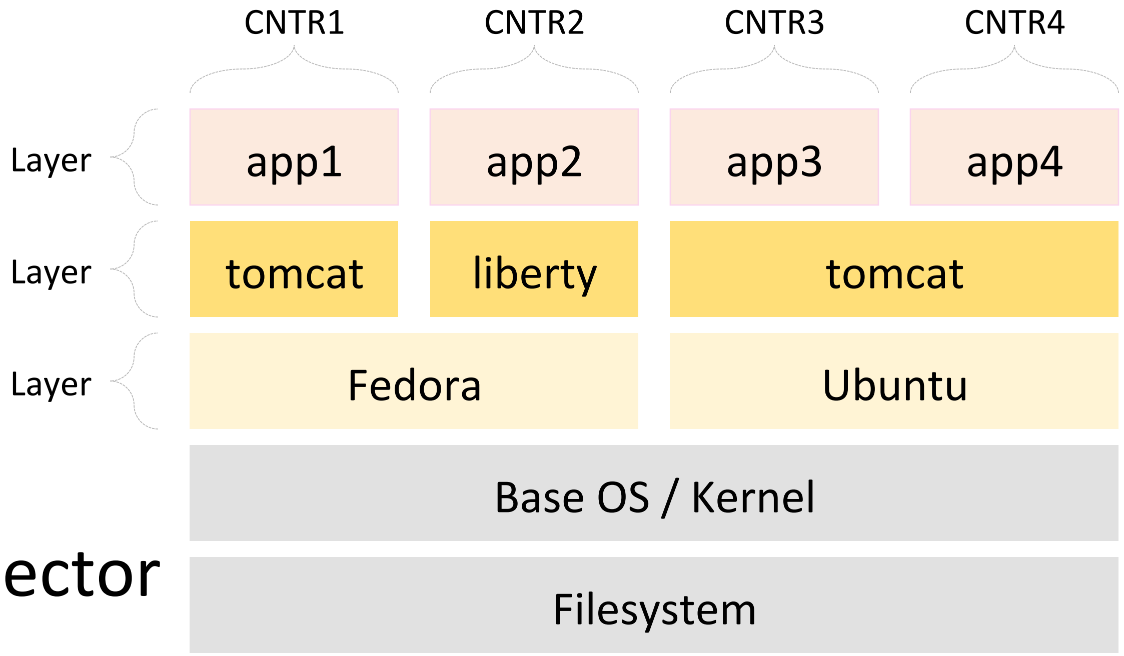


Each VM has its own OS

Containers share the same base Kernel

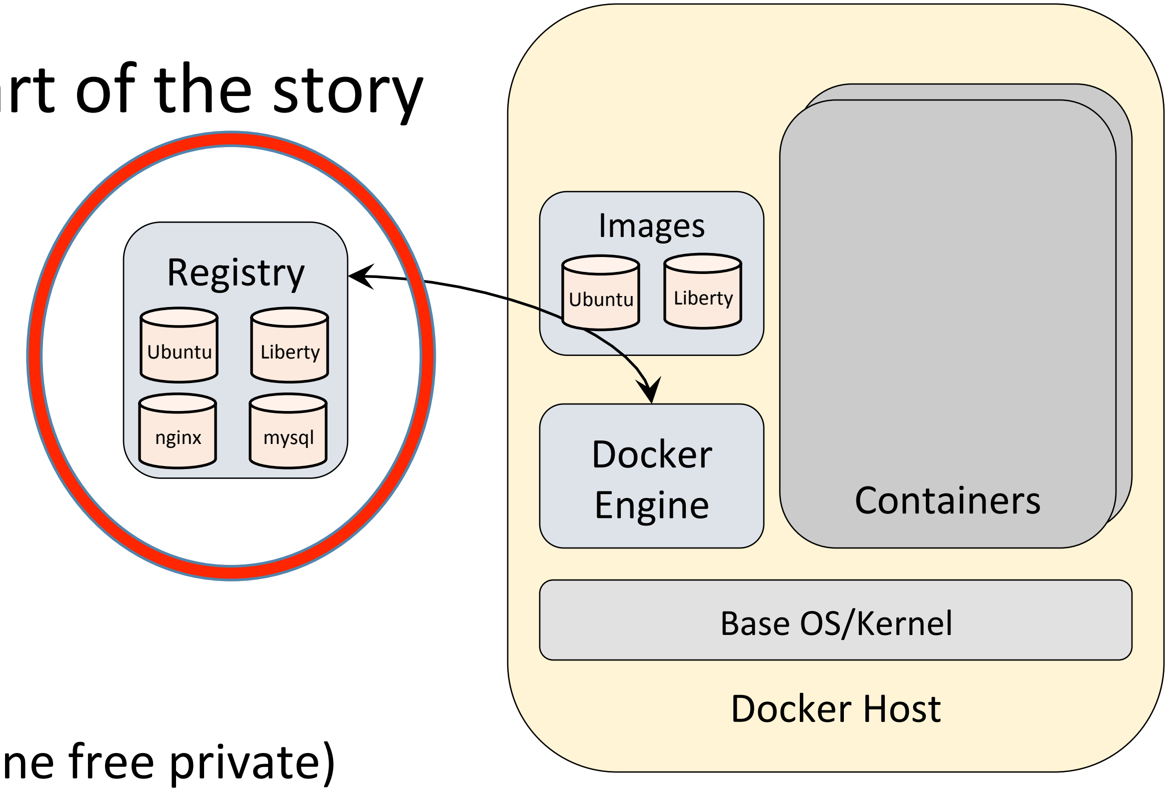
Shared / Layered / Union Filesystems

- 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



Docker Registry

- Creating and using images is only part of the story
- Sharing them is the other



- DockerHub - http://hub.docker.com
  - Public registry of Docker Images
  - Hosted by Docker Inc.
  - Free for public images, pay for private ones (one free private)
  - By default docker engines will look in DockerHub for images
  - Web interface for searching, descriptions of images

Multi-Architecture Support

- Before: **Docker runs everywhere!** (as long as its x86/Linux)

- Now: Docker daemon has multi-architecture support
  - Docker builds for Power, Z, ARM - Linux
  - Windows CLI built in community, Windows daemon built by Microsoft, not GA yet

- Registry Multi-architecture support is available
  - Engine and OSS Registry code and DockerHub supports it
  - Docker CLI doesn't provide a nice UX yet, but there are tools available

- Engine when pulling down an image:
  - Sends host's arch & OS along with the image tag
  - Registry will find image+arch+OS

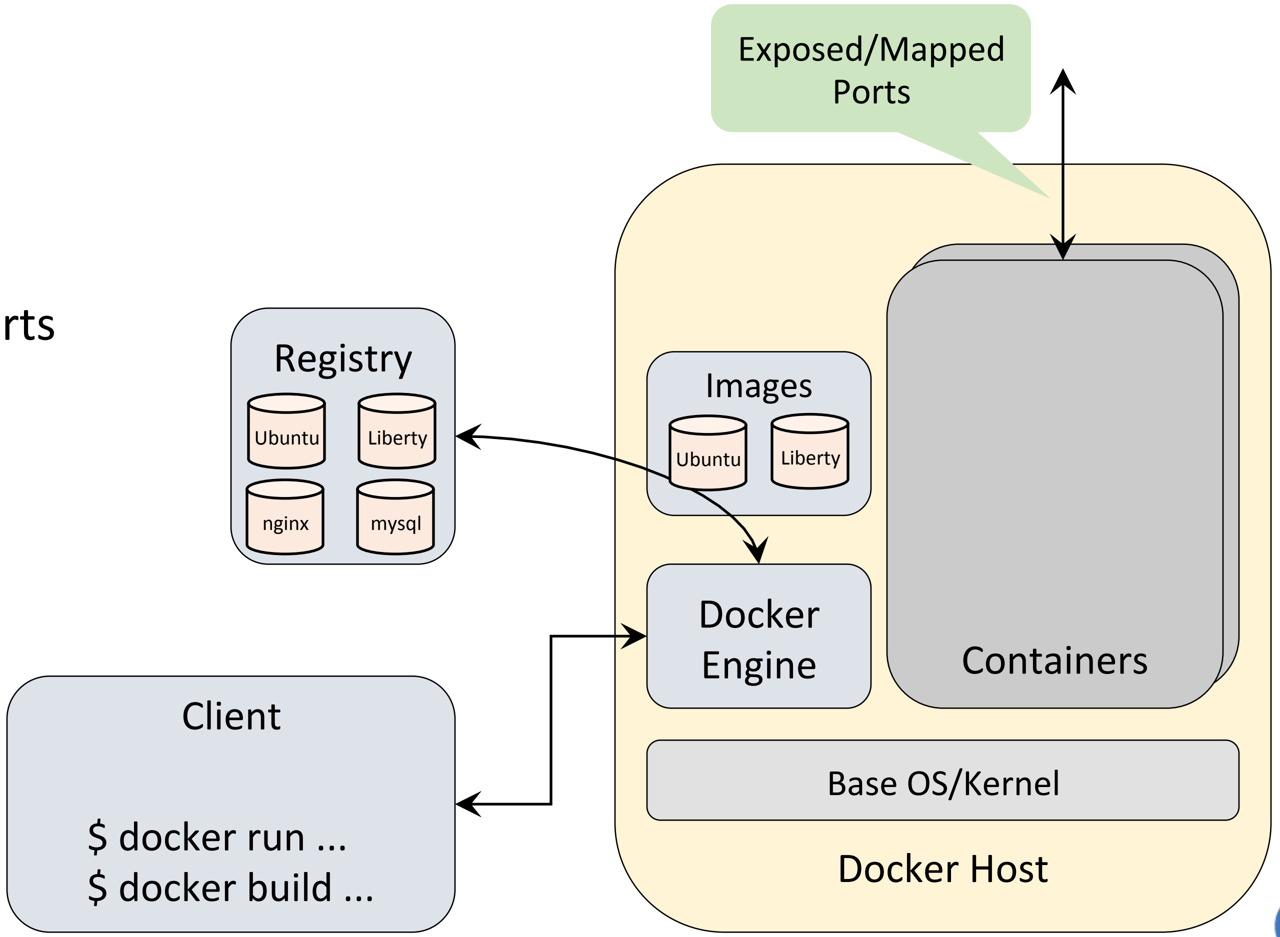
Docker Component Overview

- Docker Engine
  - Manages containers on a host
  - Accepts requests from clients
    - REST API
  - Maps container ports to host ports
    - E.g. 80 → 3582

- Images

- Docker Client
  - Drives daemon
  - Drives "builder" of Images

- Docker Registry
  - Image DB



Topics——Configuration Management

- Configuration Management

- Ensuring developers and stages of the CI pipeline have the correct environment can be a challenge
  - Install variants based on machines
  - Wrong version of products installed

Scenario

- A new developer has joined the team
- They already have:
  - Ubuntu VM with Docker installed
  - "git clone" of the source code for the project:

    `~/myapp` in the provided VM
- We need to get them up and running as quickly as possible
  - Without installing anything else!

The setup

```
$ cd myapp

$ cat Makefile
myapp: myapp.go
   go build -tags netgo -installsuffix netgo -o myapp myapp.go
```

- There's nothing here about Docker, just a normal compile step

Verify we're missing our dev environment

```
$ make
go build -tags netgo -installsuffix netgo -o myapp myapp.go
make: go: Command not found
Makefile:2: recipe for target 'myapp' failed
make: *** [myapp] Error 127
```

Solution

- Our IT department has provided a Docker image called "golang"
- This image has the go compiler installed
- Let's use this image to do our build

Using the "golang" image

- Abstractly:
  - Create a new container using the "golang" image
  - Make our source code available inside of the container
  - Build our application in the container
  - Make the executable available outside of the container
    - Otherwise the results will be lost when the container is deleted

IBM Open by design™

Using the "golang" image

- Technically the IT department would setup the Makefile like this:

```
docker run golang -v $(PWD):/src  -w /src
  go build -tags netgo -installsuffix netgo-o myapp myapp.go
```

- Summary:
  - **`docker run golang`** # Creates a container based on "golang"
  - **`-v $(PWD):/src`** # Mounts current directory into container at /src
  - **`-w /src`** # Docker will "cd" to /src before starting process
  - Notice that we didn't modify the normal developer's process, we just wrapped it with Docker

Test the Build

```
$ ./myapp 8080
Will show:
<pre><b>v1.0 Host: docker  Date: 2016-09-04
05:27:42.582058185 -0700 PDT</b>
127.0.0.1
192.168.59.147
172.17.0.1
172.19.0.1
172.18.0.1
172.20.0.1

Listening on: 0.0.0.0:8080


$ curl localhost:8080     # Test it from another window
ctrl-c                    # To stop it
```

Topics——Building Value

- Becoming a creator, and exporter of content, via Docker Images
- Adding value to existing Images
- Sharing this content via Docker Registries
- Becoming part of the value-add chain

Scenario

- Sharing the result of a build with the rest of the CI/CD pipeline
- We have the output of a product build ("myapp" executable)
- We need to build a Docker Image and share it

Creating a Docker Image - Manually

- Create a Docker Image by "snapshotting" a container

- First we need to create a container with our application

```
$ docker create ubuntu
5ed983843bbaef1062096e456e6fd931e6f24e9399d7c801adc7f
```

- Now let's copy our executable into it:

```
$ docker cp myapp 5ed98:/myapp
```

- Finally, snapshot the container as a Docker Image - called "myapp"

```
$ docker commit -c "entrypoint /myapp" 5ed98 myapp
sha256:7c640789dae5607c868a56883189d6c72478eff1080a67
```

Test the Image

```
$ docker run -ti myapp
Will show:
<pre><b>v1.0 Host: 165dcbc3e6f8  Date: 2016-09-05 02:47:50.2...</b>
127.0.0.1
172.17.0.2

Listening on: 0.0.0.0:80
```

- In another window:

```
$ curl 172.17.0.2
<pre><b>v1.0 Host: b8d73b85cc04  Date: 2016-09-05 02:53:49.803922...</b>
127.0.0.1
172.17.0.2
```

- Stop the app by pressing: **`ctrl-c`** in first window

Discussion

- Can we expose this container at the host level so others can access it?

- Yes, by mapping port 80 in the container to a unique port on the host

```
$ docker run -d -ti -p 9999:80 myapp
4b08d035deb6135eff60babd1368ab47c0c1f1d09a8ddf3f9417e7e4c4

$ curl localhost:9999
<pre><b>v1.0 Host: 4b08d035deb6  Date: 2016-09-05
03:14:31.713...</b>
127.0.0.1
172.17.0.2

$ docker rm 4b08
Failed to remove container ...

$ docker rm -f 4b08
4b08
```

Creating a Docker Image - With Docker Build

- Docker provides a "build" feature
- Uses a "Dockerfile"
  - Like a "Makefile", a list of instructions for how to construct the container

```
$ cat Dockerfile
FROM ubuntu
ADD myapp /
EXPOSE 80
ENTRYPOINT /myapp
```

Creating a Docker Image - With Docker Build

```
$ docker build -t myapp .
Sending build context to Docker daemon 5.767 MB
Step 1/4 : FROM ubuntu
 ---> ff6011336327
Step 2/4 : ADD myapp /
 ---> b867e19a859b
Removing intermediate container ea699ecc51a0
Step 3/4 : EXPOSE 80
 ---> Running in 85c240f03ae9
 ---> 5d8e53bbf9e4
Removing intermediate container 85c240f03ae9
Step 4/4 : ENTRYPOINT /myapp
 ---> Running in f318d82c2c38
 ---> 684c6c2572ff
Removing intermediate container f318d82c2c38
Successfully built 684c6c2572ff
```

Test the image - With Auto-Port Allocation

```
$ docker run -tidP myapp
469221295fae1b57615286ec7268272e3d3583c12ea66e14b2

$ docker ps
CONTAINER ID   IMAGE   COMMAND                    CREATED          STATUS
                PORTS                          NAMES
469221295fae   myapp   "/bin/sh -c /myapp"   4 seconds ago    Up 4
seconds         0.0.0.0:32768->80/tcp    clever_ardinghelli

$ curl localhost:32768
<pre><b>v1.0 Host: 469221295fae  Date: 2016-09-05 03:34:49....</b>
127.0.0.1
172.17.0.3

$ docker rm -f 469
469
```

Sharing the Image

- The "myapp" image is only in our local image cache
- To distribute it we need to upload it to a shared registry

Naming Images

- Before uploading an image, its name must include the registry

- General syntax of image names:
  - `[[registry/][namespace/]]name[:tag]`
  - `E.g. docker:5000/myapp:1.0          # "docker" is our hostname`

- Registry: host:port - presence of ":" disambiguates from "namespace"
- Namespace: user, owner
- Tag: typically a version string - defaults to "latest"

Preparing our Image

```
$ docker build -t docker:5000/myapp:1.0 .
Sending build context to Docker daemon 5.767 MB
Step 1/4 : FROM ubuntu
 ---> ff6011336327
Step 2/4 : ADD myapp /
 ---> Using cache
 ---> b867e19a859b
Step 3/4 : EXPOSE 80
 ---> Using cache
 ---> 5d8e53bbf9e4
Step 4/4 : ENTRYPOINT /myapp
 ---> Using cache
 ---> 684c6c2572ff
Successfully built 684c6c2572ff
```

- Alternative:

```
$ docker tag myapp docker:5000/myapp:1.0
```

Pushing the Image

```
$ docker push docker:5000/myapp:1.0
The push refers to a repository [docker:5000/myapp]
5d1c38831713: Pushed
447f88c8358f: Pushed
df9a135a6949: Pushed
dbaa8ea1faf9: Pushed
8a14f84e5837: Pushed
latest: digest: sha256:71f76c1b360e340614a52bcfef2cb78d8f0aa3604 size: 1363
```

- Image is in the registry and can be used by other part of the pipeline

```
$ docker run -ti docker:5000/myapp:1.0
$ docker pull docker:5000/myapp:1.0
```

Discussion Point

- Our Dockerfile stared with: **`FROM ubuntu`** do we really need Ubuntu ?
- No, there is nothing in our app that uses the operating system

```
$ docker images | grep myapp
myapp   latest     684c6c2572ff  7 hours ago  193.7 MB
```

- Instead our Dockerfile could use: **`FROM scratch`**
  - Let's do that so our image is smaller

```
$ docker build -f Dockerfile2 -t docker:5000/myapp .
$ docker push docker:5000/myapp    # and update our registry
```

```
$ docker images | grep myapp
myapp   latest    9b604f2e42da  7 seconds ago 7.591 MB
```

Discussion Point

- What are **some** of the other instructions can we have in a Dockerfile?
  - RUN
  - HEALTHCHECK
  - COPY/ADD
  - CMD & ENTRYPOINT
  - LABEL
  - ENV/ARG
  - VOLUME
  - USER
  - WORKDIR

Topics——Container Orchestration

- Compose
- Swarm
- Kubernetes
- Mesos

Docker Community

- Anyone can be a contributor
- Everyone follows the same Pull Request (PR) process
- Once proven, someone may nominate you to be a "maintainer"
- Maintainers:
    - Can LGTM (looks good to me) PRs
    - Can veto PRs
    - Can be part of off-line/private maintainer discussions (irc and mailing list)
    - Can attend the Docker Governance Advisory Board meetings (DGAB)
        - To allow key community leaders to influence higher-order issues/discussions
        - IBM (Jeff Borek) is the chair

- One of the more open and fair communities
    - Actively seeks newbies (e.g. hackathons)
    - Eager to help when people have questions

A Bit of Process

- Each Docker project should have a ROADMAP.md file
  - Defines the long-term goals and plans

- Tries to ship every 9 weeks
  - 6 weeks of dev, 3 weeks of testing (freeze)
  - All Docker projects are on the same release schedule

- During "freeze" period
  - Release Candidate branch is created by the "Release Owner"
    - v.Major releases are branches from master
    - v.Major.Minor releases are branches from the v.Major branch
  - New work is still merged into "master", "cherry-picked" into "RC" branch
  - Multiple RCs tags will be created during the testing

  - https://github.com/docker/docker/blob/master/project/RELEASE-PROCESS.md
  - Rest of their "process" docs: https://github.com/docker/docker/tree/master/project

Communications

- IRC
  - docker – general docker questions - newbies
  - **docker-dev – docker contributors**
  - docker-maintainers – for docker maintainers, r/o to others
  - Project specific - e.g. docker-compose
- Mailing list
  - docker-dev@googlegroups.com
  - Mainly for generic questions, not very active – use irc instead
- Internally
  - IRC: rochester.irc.ibm.com – "docker" channel
  - Slack: https://ibm-cloudplatform.slack.com/ – "#docker" channel
    - Request Access: http://ibm.biz/cloudplatform-request
  - Mail: docker@webconf.ibm.com

Github Repos

- Hosted at: https://github.com/docker

- Key repos:
  - docker: docker, swarm, machine, compose

  - OCI: https://github.com/opencontainers
    - runc, runtime-spec

Other Key Companies

#	Company	Commits
1	Docker	17999
2	Red Hat	1326
3	IBM	874
4	Huawei	828
5	MSFT	634

Looking ahead...

- Growing fast!
  - 500+ million containers downloaded
  - 1300 contributors, 20 core maintainers
  - 150,000 Dockerized projects on GitHub
  - 25,000 meetup members, 140 cities, 50 countries
  - Lots of "big" players: IBM, RedHat, Google, Microsoft, …
  - 97% of 685 enterprise CIOs intent to spend $ on Docker related technology

- Concerns:
  - Long-term status of project - Docker Inc. is becoming more "closed"
  - Lock-in
  - Governance – one company
  - Split between Docker OSS and Docker is very very blurry

Next Steps...

- To address these concerns the community is working with the Linux Foundation

- Setup two new foundations:
  - OCI – Open Container Initiative
  - CNCF – Cloud Native Computing Foundation

OCI

- Open Container Initiative (OCI)
  - New project under Linux Foundation
    - https://www.opencontainers.org/
  - Docker contributed “libcontainer”, renamed to "runc"
  - Deliverables
    - Core container runtime specification
    - Reference implementation - used by Docker engine - "runc"
    - Specification of command line syntax
    - Definition of what an Image looks like - for sharing between implementations

CNCF

- Cloud Native Computing Foundation (CNCF)
  - New project under Linux Foundation
    - https://cncf.io/
  - Orchestration, discovery, distribution and lifecycle management of clusters of containers across a data center (DCOS)
  - Deliverables: specifications & reference implementations
- Status
    - Kubernetes (Google),and Prometheus have been contributed

Summary——Why Containers are Appealing to Users

Lightweight & Fast
Faster startup/showdown. Gives services near instant scaling capabilities.

Faster Time to Market
Apps & dependencies are bundled into a single image. Host, OS, distro and deployment are independent allowing for workload portability.

Version Tracking
User easily rolls between versions

Simplified Isolation
Each container has its own network stack with controls over ports and permissions.

Enhanced Security
Containers allow for finer-grained control over data and software installed. Reduces the attack surface area/vulnerabilities of the apps.

Easier to Manage
Enables frequent patch of applications while reducing the effort of validating compatibility between apps/environment.

Simpler to Maintain
Install, run, maintain and upgrade applications and their envs quickly, consistently and more efficiently than VMs.

Resource Friendly
Can host more containers then corresponding VMs.

Summary——Why Containers instead of VMs?

- Why Containers?
  - Better resource utilization - orders of magnitude more containers per host than VMs
  - Ease of automation
  - Can be used in places that would be a challenge for VMs
    - E.g. our "git" example where we run an app in a container instead of installing it

- Encourages a Micro-services architecture
  - Encourages the split from monolithic apps into smaller pieces
  - Because you're focused on single processes, not entire systems/VMs

- When do I use containers?
  - Start with them and only switch to VMs when you hit a brick wall

Summary——Rewards of using Docker

- Consistent Reproducible Environments
  - Developer
    - Pre-built dev and/or test env - reset test env (db?) on each test
    - Spend time coding – not setting up environments
  - CI Pipeline and Software Delivery
    - Guaranteed to be the same in each phase of pipeline - same OS, libraries, patches…
    - No more lengthy, complicated, inconsistent install processes
    - Can version control everything in the container - roll backward/forwards thru images
- In fairness – similar concepts to VM images, just faster, easier, better...

Summary——Cloud Programming Model

- With containers we are encouraged to do the things we were told to do with VMs:
  - Easier to sell the "good practices" we're been pitching for years
  - Treat them as ephemeral workloads - cattle
    - Assume they will crash
    - However, nothing about them (or Docker) requires them to be short-lived
  - Automate their creation so you have reproducible environments
    - Docker's tooling is so UX-friendly it hard not to want to automate things
  - Micro-services architecture
    - Could do it with VMs, but its less likely to include the kitchen sink with containers

Summary——Container Migration Considerations

- Micro-Services: You don't have to switch to micro-services to use containers
  - There is nothing stopping you from moving the monolith into a container
  - Build up your CI/CD infrastructure & automation around it
  - Then look for opportunities to split the app into smaller components

- Containerizing applications
  - Ideally your application containers should be read-only / ephemeral
    - Customize app's runtime via env vars, flags, mounted config files, ...
  - For data, use persistent volumes

Summary——Risks with Containers and Docker

- Containers:
  - In general, change is scary so there's a comfort factor to overcome
    - But most the (initial) change is outside of the application code itself
  - On-Prem - no new risks, no real difference than VMs
  - Public Cloud - suspected security issues have gone by the way-side
- Docker
  - Lack of true open governance
  - Docker is becoming more closed and proprietary

- Thanks!