Docker Registry

A FRAMEWORK FOR DATA INTENSIVE COMPUTING

Agenda

Intro / Prep Environments

Day 1: Docker Deep Dive

Images, Containers, and Storage Drivers

- In the lesson we'll learn how Docker builds and stores images
- Then, we'll learn how these images are used by containers
- We'll also get a short introduction in the technologies that enable both images and container operations
- Finally, we will look briefly at creating a Docker Registry and what it is

Docker Images

- Image tags
- Points to a specific layer
- Usually the last most layer gets changed
- Can have multiple tags each pointing to diff layers; same base
- Don’t indicate `latest`, if you can help it

Docker Images

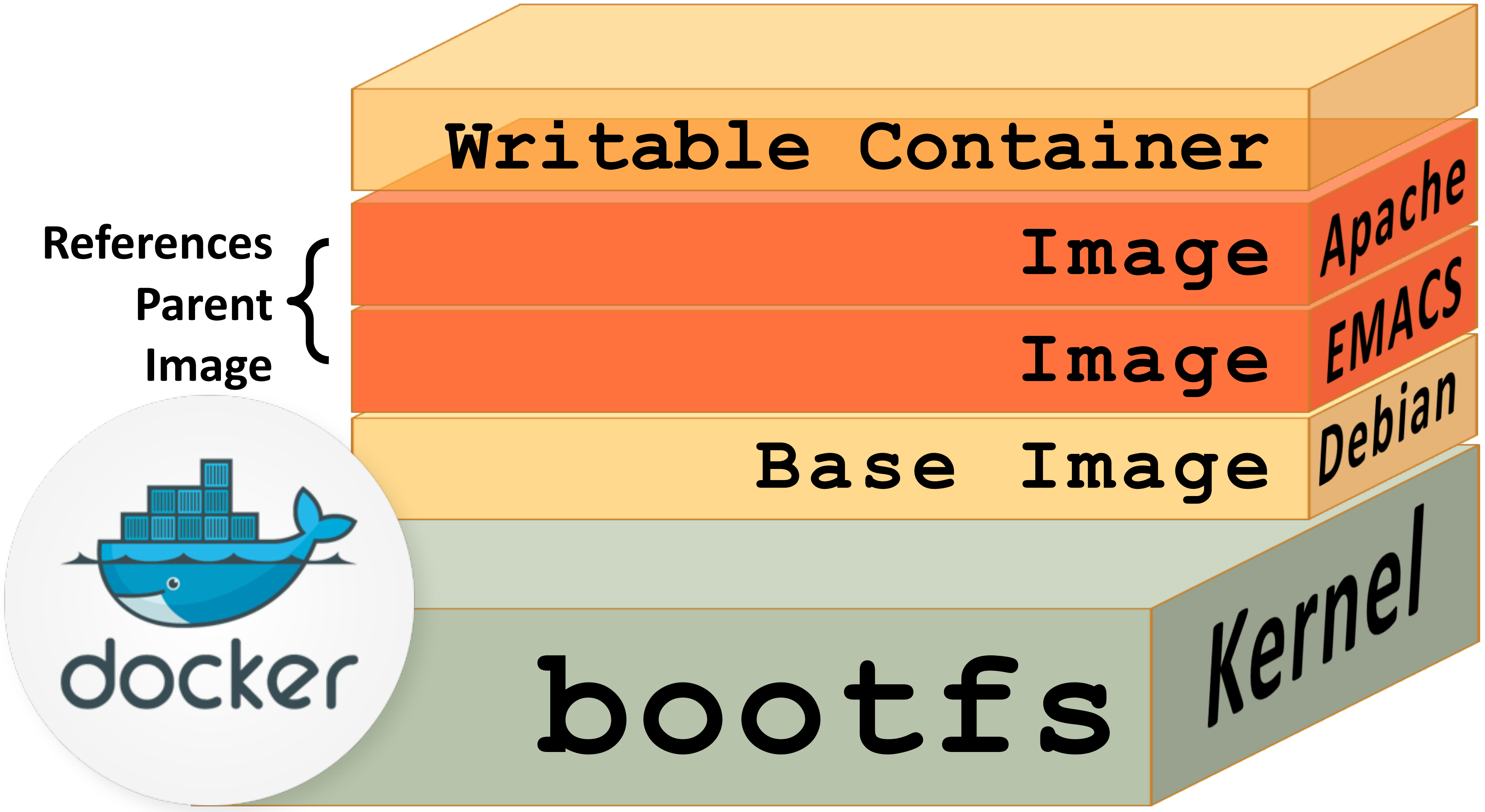
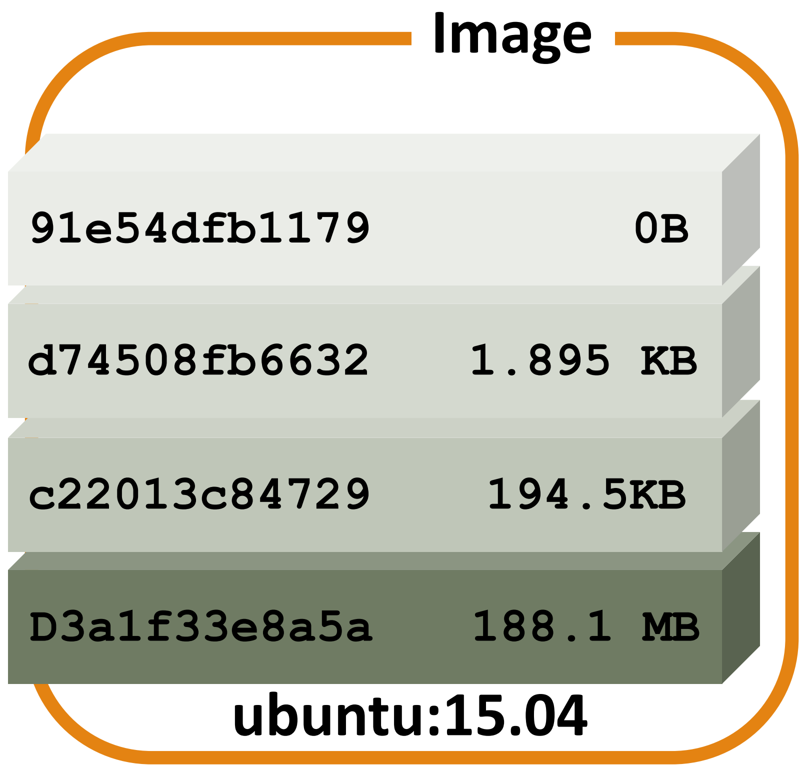


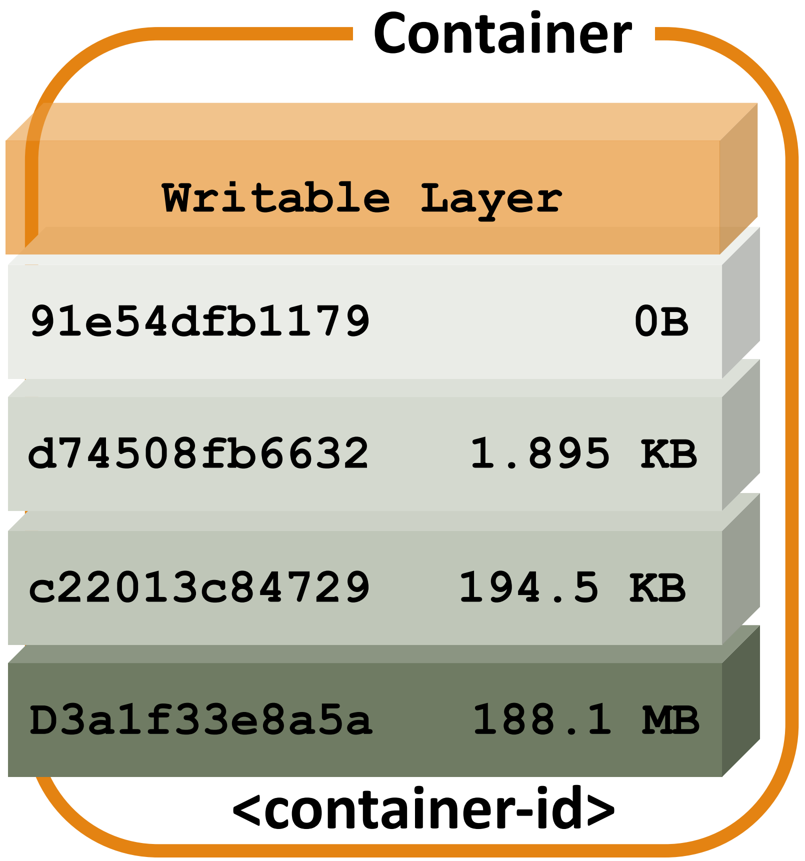
Image Layers

- Each Docker **image** references a list of read-only layers that represent filesystem differences.
- Layers are stacked on top of each other to form a base for a container's root filesystem.
- The diagram shows Ubuntu 15.04 image's stacked layers.
- NOTE: Docker storage driver is responsible for stacking these layers and providing a single unified view.



Container Layers

- When you **create** a new container, you add a new, thin, writable layer on top of the underlying stack.
- This **layer** is often called the "container layer".
- All changes made to the running container are written to this thin **writable** container layer.
- The diagram here shows a **container** based on the Ubuntu 15.04 image.



Differences Summary

- ❑ The major difference between a container and an image is the top writable layer.
- ❑ All writes to the container that add new or modify existing data are stored in this writable layer.
- ❑ When the container is deleted the writable layer is also deleted and the underlying image remains unchanged.
- ❑ Because each container has its own thin writable container layer, and all changes are stored in this container layer, this means that multiple containers can share access to the same underlying image and yet have their own data state.

Docker Images

- Now, let’s run a JVM based application, like Apache Tomcat:

```
docker run –it --rm centos:7 bash
```

- Now in another window, we could list the Docker containers running:

```
docker ps
```

- We can ssh into the VM and see where the images/containers are stored

```
docker-machine ssh default
```

  - NOTE: Docker Machine is used here.

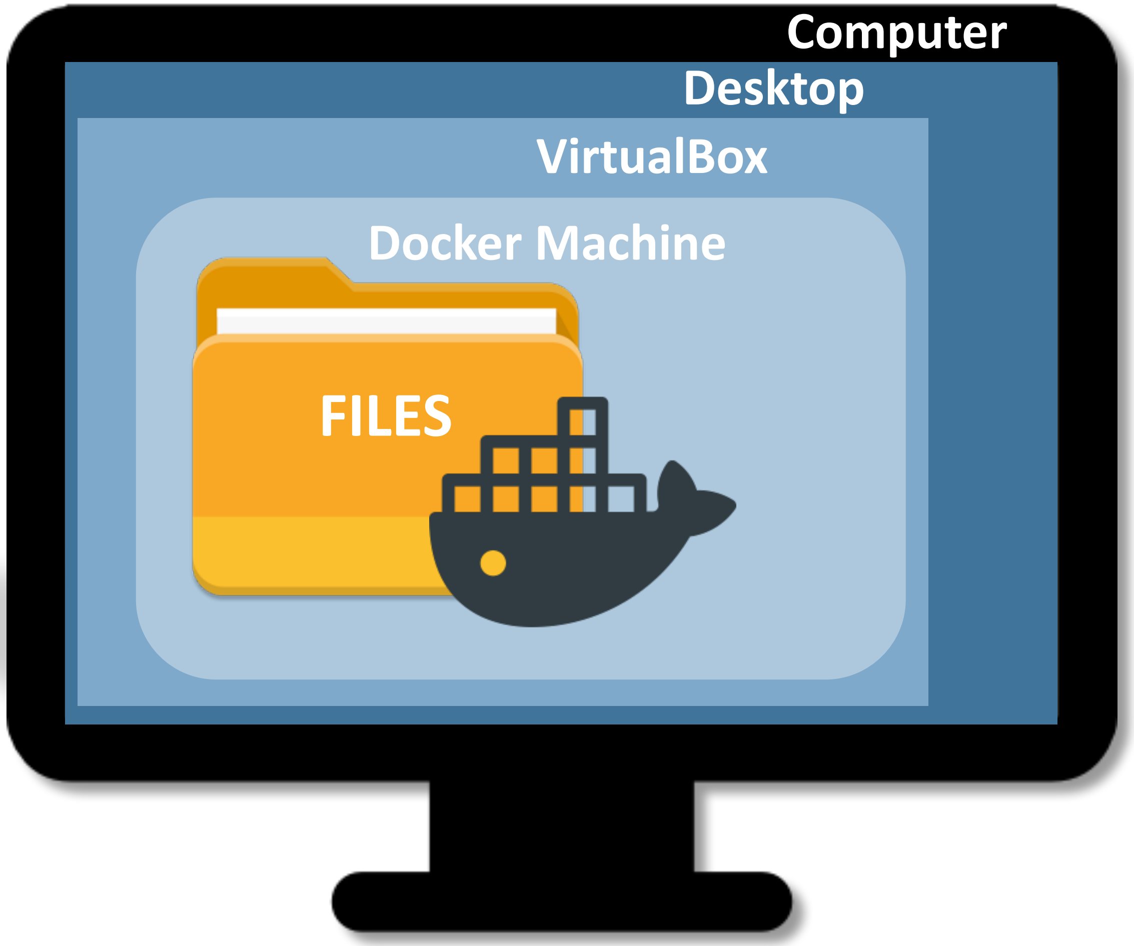
Where is Docker

- So now, we are inside a Docker Machine
- First, we'd elevate our user privileges:

```
sudo su -
```

- Navigate to the appropriate file:

```
cd /var/lib/docker
```

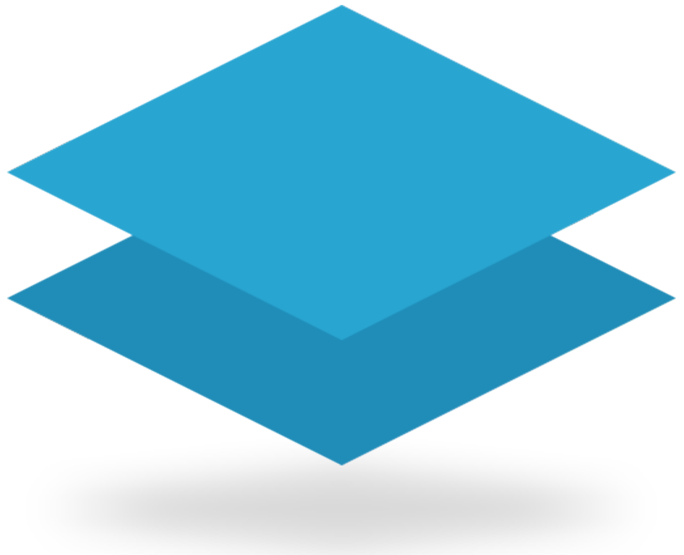


Where is Docker

- Here's how we might use the `find`:

```
find ./aufs/mnt -name <container id>*
```

- This is the location where your container lives.
- If you inspect that folder, you'll see the running container's files right there.
- Each directory in that location is a layer in the image.



Docker Backend

- **Device Mapper** is a kernel-based framework that underpins many advanced volume management technologies on Linux.
- Docker’s **devicemapper** storage driver leverages the thin provisioning and snapshotting capabilities of this framework for image and container management.
- **NOTE**: The Device Mapper storage driver is referred to as devicemapper, and the kernel framework as Device Mapper.

AUFS Storage Driver

- ❑ AUFS was the first storage driver in use with Docker.
- ❑ AUFS has several features that make it a good choice for Docker.
- ❑ These features enable:
  - ❑ Fast container startup times.
  - ❑ Efficient use of storage.
  - ❑ Efficient use of memory.
- ❑ Despite its capabilities and long history with Docker, some Linux distributions do not support AUFS.
- ❑ This is usually because AUFS is not included in the mainline (upstream) Linux kernel.

BTRFS Storage Driver

- ❑ BTRFS is a next generation copy-on-write filesystem that supports many advanced storage technologies that make it a good fit for Docker.
- ❑ BTRFS is included in the mainline Linux kernel and its on-disk-format is now considered stable.
- ❑ However, many of its features are still under heavy development and users should consider it a fast-moving target.
- ❑ Docker’s BTRFS storage driver stores every image layer and container in its own BTRFS subvolume or snapshot.
- ❑ The base layer of an image is stored as a subvolume whereas child image layers and containers are stored as snapshots.

Mac & Windows Storage Drivers

- The average Mac and Windows user won’t need to change our storage drivers.
- Docker for Mac and Docker for Windows only support `overlay2` `aufs`, `overlay`, or `vfs`.
- The last two are **not** recommended.
- Currently, `aufs` is the default in stable releases and `overlay2` is the default in Edge releases.
- To see what driver you are currently utilizing, display `info`, like this:

```
docker info
```

What is Docker Register

- So, now that we've learned more about images and containers, what do we do with them?
- Let's look ahead at how a Docker Registry can assist that process.
- A registry is a great place to park, or store an image.
- Docker Registry not only stores them, but instructions on how they go together.
- The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images.

Why Use Docker Register

- You should use the Registry if you want to:
  - Tightly control where your images are being stored.
  - Fully own your images distribution pipeline.
  - Integrate image storage and distribution tightly into your in-house development workflow

Registry Quick Look

- Start the registry:

```
docker run -d -p 5000:5000 --name registry registry:2
```

- Pull (or build) some image from the hub:

```
docker pull ubuntu
```

- Tag the image so that it points to your registry:

```
docker tag ubuntu localhost:5000/myfirstimage
```

Registry Quick Look

- Push it:

```
docker push localhost:5000/myfirstimage
```

- Pull it back:

```
docker pull localhost:5000/myfirstimage
```

- Now, stop your registry and remove all data:

```
docker stop registry && docker rm -v registry
```

Docker Register Alternatives

- ❑ Users looking for a zero maintenance, ready-to-go solution are encouraged to head-over to the **Docker Hub**.
- ❑ Docker Hub provides a **free-to-use**, hosted Registry, plus additional features (organization accounts, automated builds, and more).
- ❑ Users looking for a commercially supported version of the Registry should look into Docker **Trusted** Registry.
- ❑ Docker Registry is the **core** technology behind the Docker Hub.

Public/Private Docker Registry

- Docker hub: `http://docker.io`
- Can host public images
- Can also host private repos, like Github and Bitbucket.
- Other registries:
  - JFrog
  - Quay.io
  - Google Container Registry

Enterprise Docker Registry

- Be careful with images on Docker Hub, because of security vulnerabilities.
- Use official registries, which are sponsored by legitimate vendors.
- Trusted, official images on Docker Hub feature an “Official Repository” heading.
- Red Hat Docker registry
  - registry.access.redhat.com:5000
- Try pulling Red Hat’s registry, like this:

```
docker pull registry.access.redhat.com/rhel7:latest
```

Creating Dockerfile Images

- Let’s try to build Docker images from a Dockerfile, like this:

```
FROM fabric8/java-agent-bond

ENV CLASSPATH /maven/*:/maven

RUN mkdir /maven

EXPOSE 8778 9779

ADD run.sh /fabric8/run.sh

CMD [ "/fabric8/run.sh" ]
```

Dockerfile Basic Commands

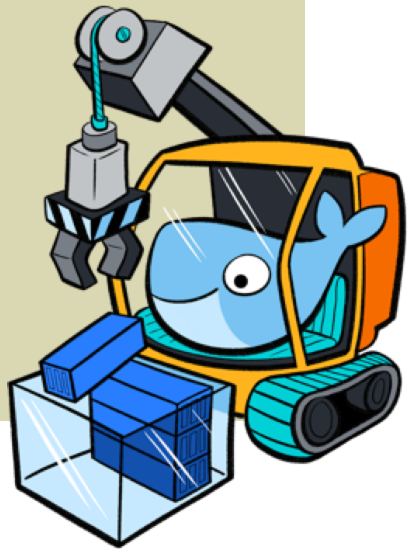
- `FROM`
- `ADD`
- `COPY`
- `USER`
- `ENV`
- `VOLUME`
- `WORKDIR`
- `CMD`
- `ENTRYPOINT`

Advanced Dockerfile

```
FROM ubuntu:14.04

MAINTAINER fabric8.io (http://fabric8.io/)

ENV GERRIT_HOME /home/gerrit
ENV GERRIT_TMP_DIR /home/tmp
ENV GERRIT_USER gerrit
ENV GERRIT_VERSION 2.11

RUN \
 sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \
 apt-get update && \
 DEBIAN_FRONTEND=noninteractive apt-get -y upgrade && \
 DEBIAN_FRONTEND=noninteractive apt-get install -y sudo vim-tiny git && \
 DEBIAN_FRONTEND=noninteractive apt-get install -y openjdk-7-jre-headless && \
 DEBIAN_FRONTEND=noninteractive apt-get install -y curl
```

Advanced Dockerfile

```
# Add user gerrit & group like also gerrit to sudo
# to allow the gerrit user to issue a sudo cmd
RUN groupadd $GERRIT_USER && \
   useradd -r -u 1000 -g $GERRIT_USER $GERRIT_USER
RUN mkdir ${GERRIT_HOME}
# Download Gerrit
ADD http://gerrit-releases.storage.googleapis.com/gerrit-${GERRIT_VERSION}.war && \
    ${GERRIT_HOME}/${GERRIT_WAR}

# Copy the files to bin, config & job folders
ADD ./configs ${GERRIT_HOME}/configs
# Copy the plugins
ADD ./plugins ${GERRIT_HOME}/plugins
...
```



Advanced Dockerfile

```
...
WORKDIR ${GERRIT_HOME}


EXPOSE 8080 29418
CMD ["/home/gerrit/bin/conf-and-run-gerrit.sh"]
```

Difference Between CMD and ENTRYPOINT

- CMD can be overridden at run time, like this:

```
docker run -it centos:7 <command_to_run>
```

- ENTRYPOINT is a fixed command, but we can pass things in as parameters:

```
docker run -it centos:7 <params_to_add>
```

  - NOTE: ENTRYPOINT cannot be overridden.

Demo Creating Docker Images

- Clone the following repository:

```
git clone git@github.com:fabric8io/base-images.git
```

- Cd into `./base-images/java/images/centos/openjdk8/jdk`

```
docker build -t local.io/docker-java:latest .
```

- Run the following command from the directory that has the Dockerfile

  - NOTE: Don’t forget the (.) character!

Demo Creating Docker Images

- Now, list the Docker images:

```
docker images
```

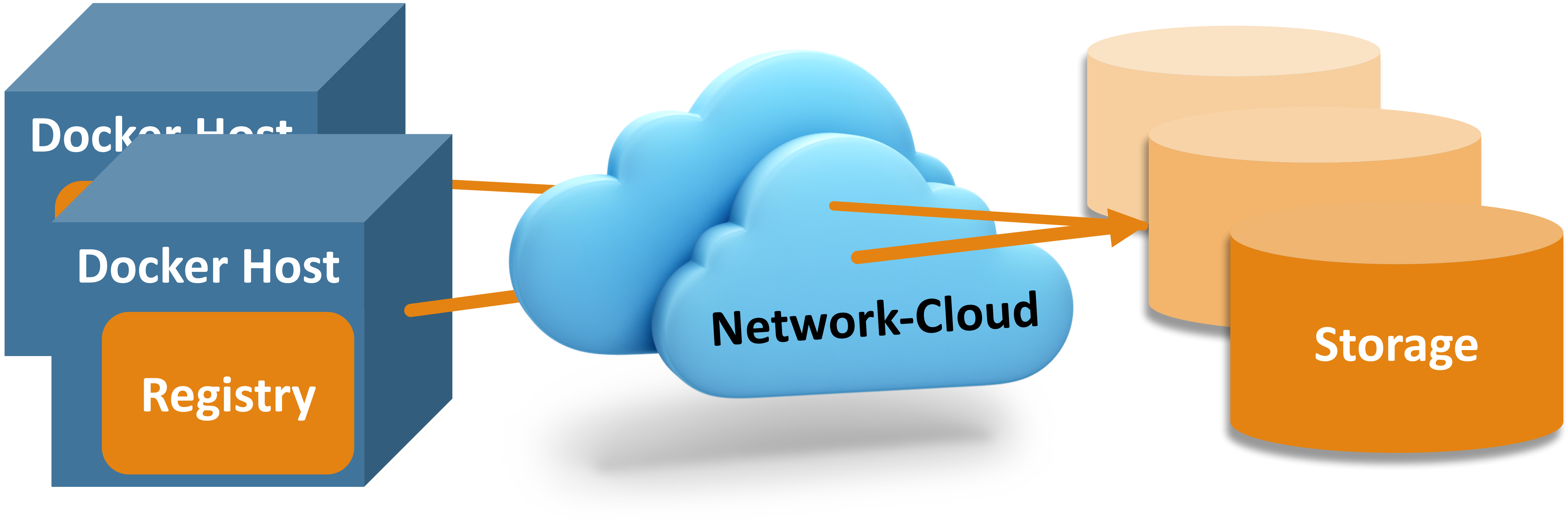
- Output:

```
REPOSITORY                         TAG         IMAGE ID        CREATED                     VIRTUAL SIZE
 local.io/docker-java               latest      8d11c57aafa6    Less than a second ago     431 MB
 tomcat                             8.0         1e41e2ebc383    2 days ago                 347.7 MB
 centos                             7           e9fa5d3a0d0e    3 days ago                 172.3 MB
 registry.access.redhat.com/rhel7   latest      82ad5fa11820    5 weeks ago                158.3 MB
 registry.access.redhat.com/rhel    latest      82ad5fa11820    5 weeks ago                158.3 MB
```

Running a Local Docker Registry

- The old "python" based Docker registry (before Docker 1.6 has been deprecated
- New Docker registry exists in "Docker Distribution" tools
  - `https://github.com/docker/distribution`
- Can run local / scaled out Docker registries
- Backed by storage
- Getting started: `https://docs.docker.com/registry/`

Registry Architecture



Registry Architecture

- **Implemented** with a Storage API that can be extended
  - More info at: `https://docs.docker.com/registry/storagedrivers/`
- **In-memory**
  - Local, in-memory; only expected for testing/reference
- **Filesystem**
  - Local-storage driver

Plugin Registry Architecture

- **Amazon Web Services**
  - S3 Buckets
- **Microsoft Azure**
  - Blob Storage
- **Ceph Rados**
  - Object Storage
- **Swift**
  - OpenStack Object Storage
- **Overhead Storage Solutions**
  - Aliyun OSS

Local Registry

- Let’s deploy a local registry and try it out:

```
docker run -d -p 5000:5000 --name registry registry:2
```

- Output:

```
Unable to find image 'registry:2' locally

 2: Pulling from library/registryf9a9f253f610: Pull complete

 eeb7cb91b09d: Pull complete

 b3b2a507517e: Pull complete

 34e7db8ae1dc: Pull complete

 2eafecf5086b: Pull complete

 Digest:
sha256:802127562bcb59ac617a1296d70023258f22fc3e401fa86c866447a8c36e4278

 Status: Downloaded newer image for registry:2

 d89a9c4719089af289e38bcc436dff0db37aa1e82ebbe5e19ce508d87dd9cd0a
```

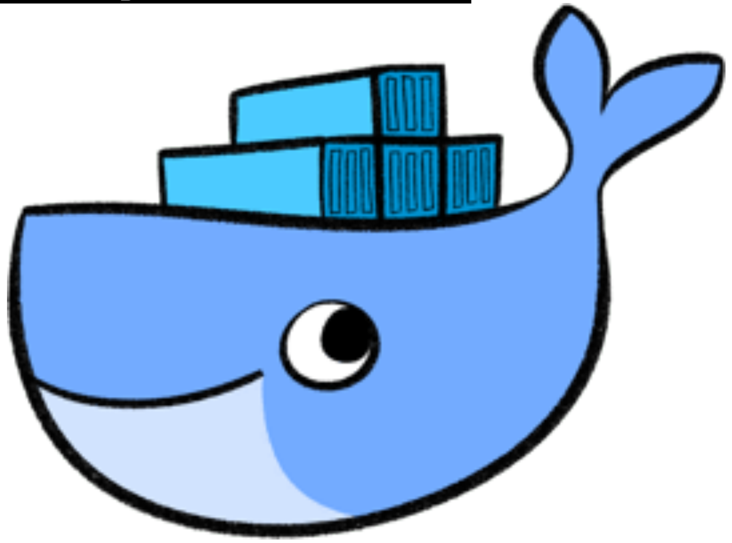
Push to Registry

- Let’s tag and push our previously created Docker image to our new registry:

```
docker tag local.io/docker-java localhost:5000/local.io/docker-java
```

- Then push it:

```
docker push localhost:5000/local.io/docker-java
```



Push to Registry

- Output:

```
The push refers to a repository [localhost:5000/local.io/docker-java] (len:
1)

8d11c57aafa6: Image successfully pushed

9dbce2cf69a6: Image successfully pushed

e9fa5d3a0d0e: Image already exists

c9853740aa05: Image already exists

e9407f1d4b65: Image already exists

0cd86ce0a197: Image successfully pushed

fa5be2806d4c: Image already exists

latest: digest:
sha256:0cebcc42cbc25848524eff2cf4aa9d5a47e5d360c5ebfb931e6d33cfd8a38b97 size:
29837
```

Lab

End of Chapter