Introduction to Docker

The open platform to build, ship and run any applications anywhere

Agenda

- Introduction to containers
- Docker concepts and terms
- Introduction to images
- Running and managing containers
- Building images

When there is time left:

- Managing and distributing images

Introduction to Containers

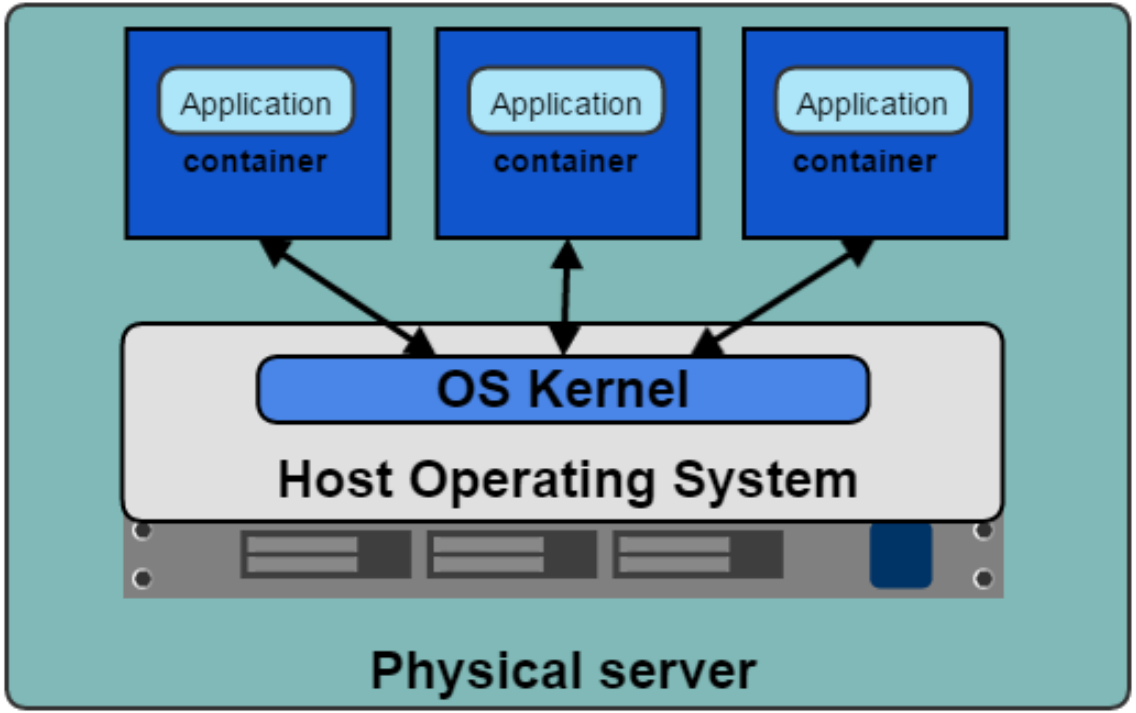
docker

Introducing Containers

Containerization uses the kernel on the host operating system to run multiple root file systems

- Each root file system is called a **container**
- Each container also has its own
  - Processes
  - Memory
  - Devices
  - Network stack

Containers


Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

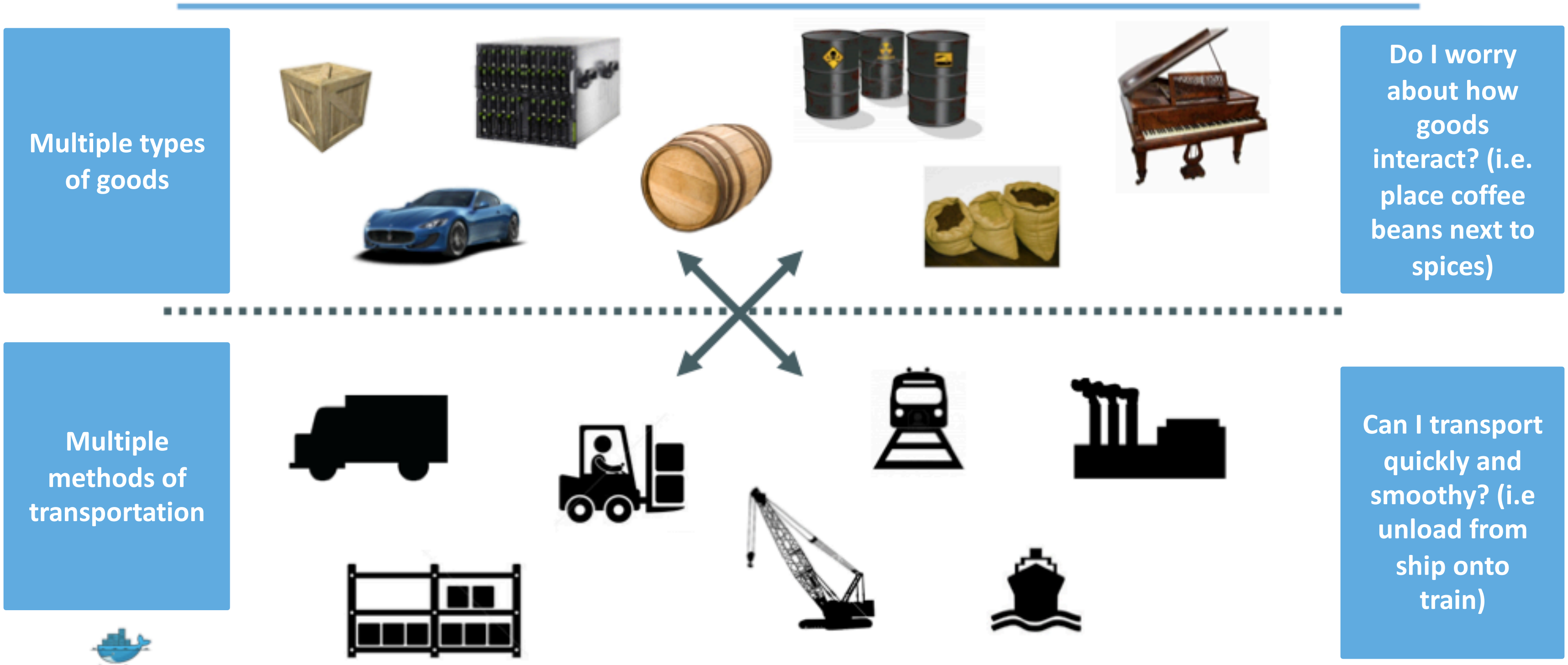
Containers vs VM’s

- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability

Why use Docker?

- Applications are no longer one big monolithic stack
- Service oriented architecture means there are multiple application stacks that need to be deployed
- Services are decoupled, built iteratively and scaled out
- Deployment can be a complex exercise

A shipping analogy



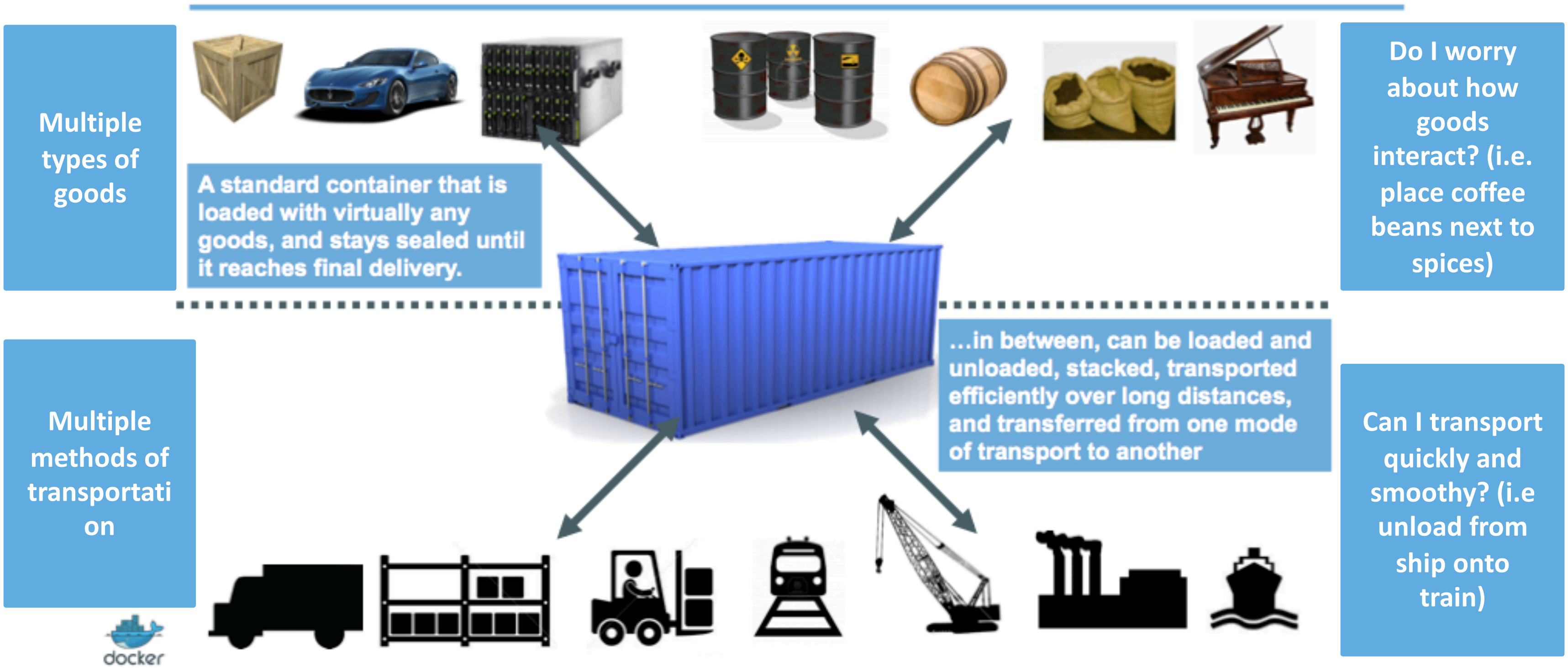
Multiple types of goods

Do I worry about how goods interact? (i.e. place coffee beans next to spices)

Multiple methods of transportation

Can I transport quickly and smoothy? (i.e unload from ship onto train)

The shipping container



Multiple types of goods

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

Do I worry about how goods interact? (i.e. place coffee beans next to spices)

Multiple methods of transportation

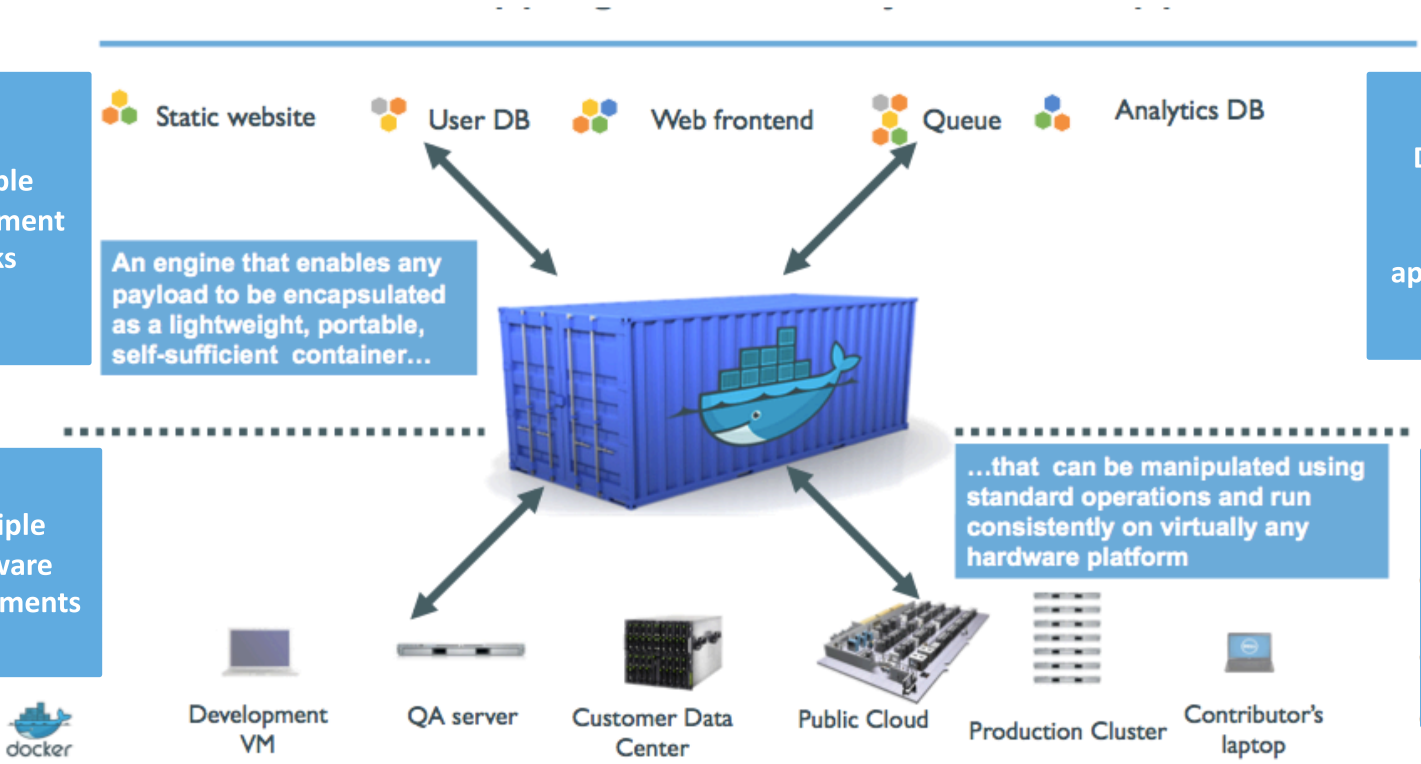
…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Can I transport quickly and smoothy? (i.e unload from ship onto train)

Docker containers

Multiple development stacks

Multiple hardware environments



An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

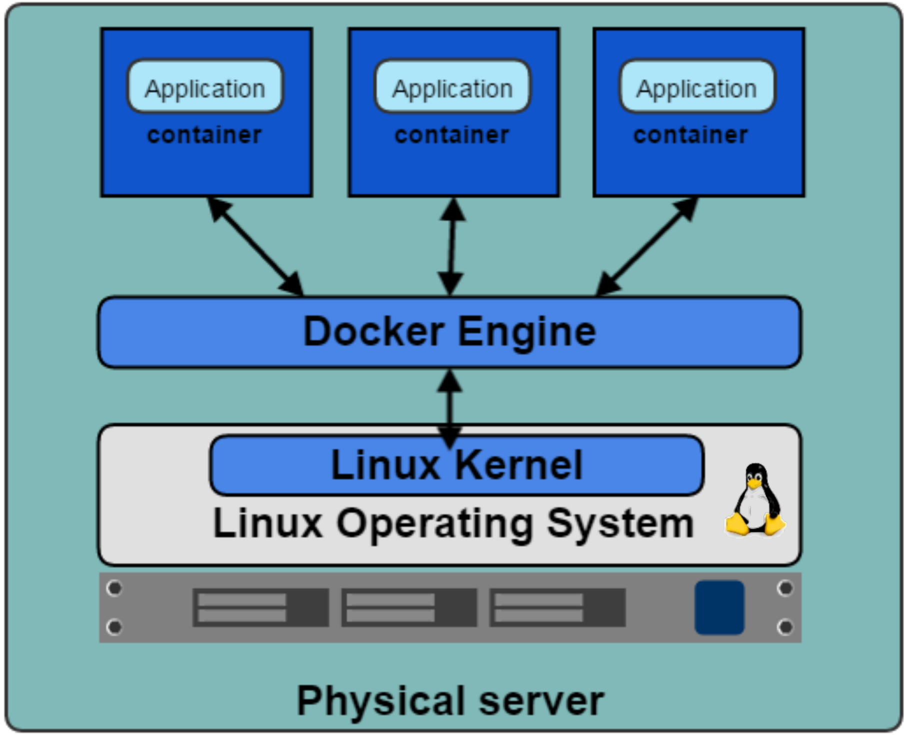
Do services and apps interact appropriately?

Can I migrate quickly and smoothly?

Docker Concepts and Terms

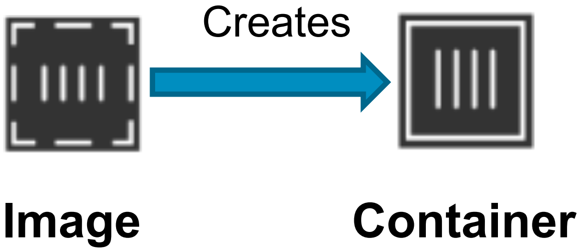
Docker and the Linux Kernel

- **Docker Engine** is the program that enables containers to be distributed and run
- Docker Engine uses Linux Kernel namespaces and control groups
- Namespaces give us the isolated workspace



Docker Containers and Images

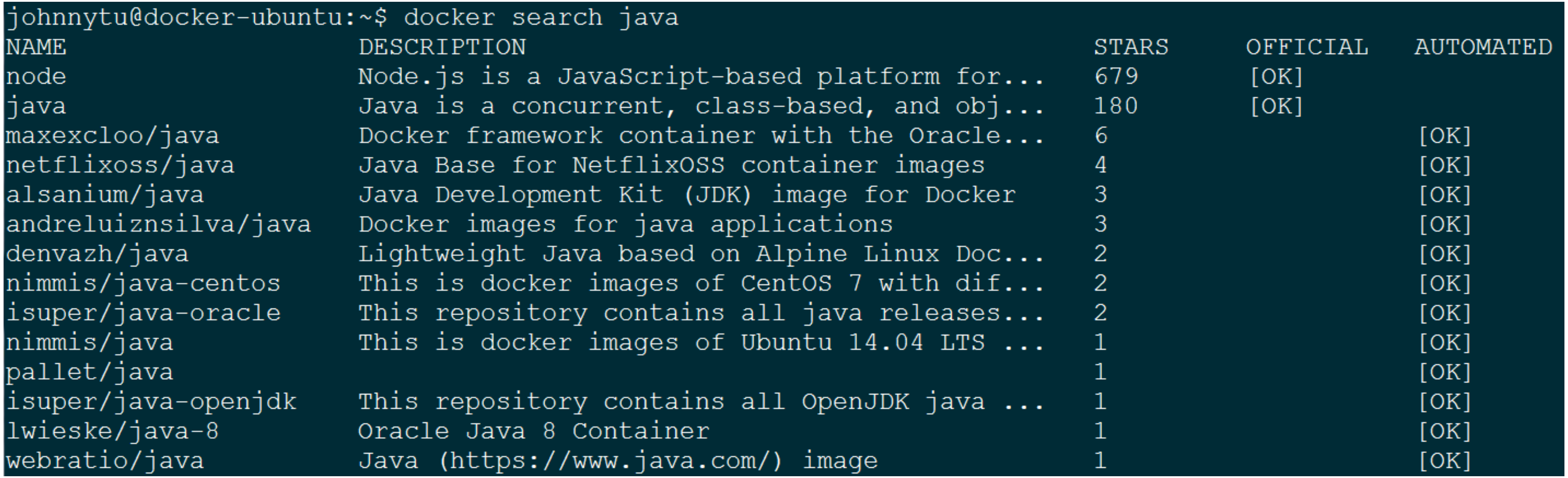
- Images
  - Read only template used to create containers
  - Built by you or other Docker users
  - Stored in Docker Hub, Docker Trusted Registry or your own Registry
- Containers
  - Isolated application platform
  - Contains everything needed to run your application
  - Based on one or more images



Introduction to Images

Search for images using Docker client

- Run the `docker search` command
- Results displayed in table form



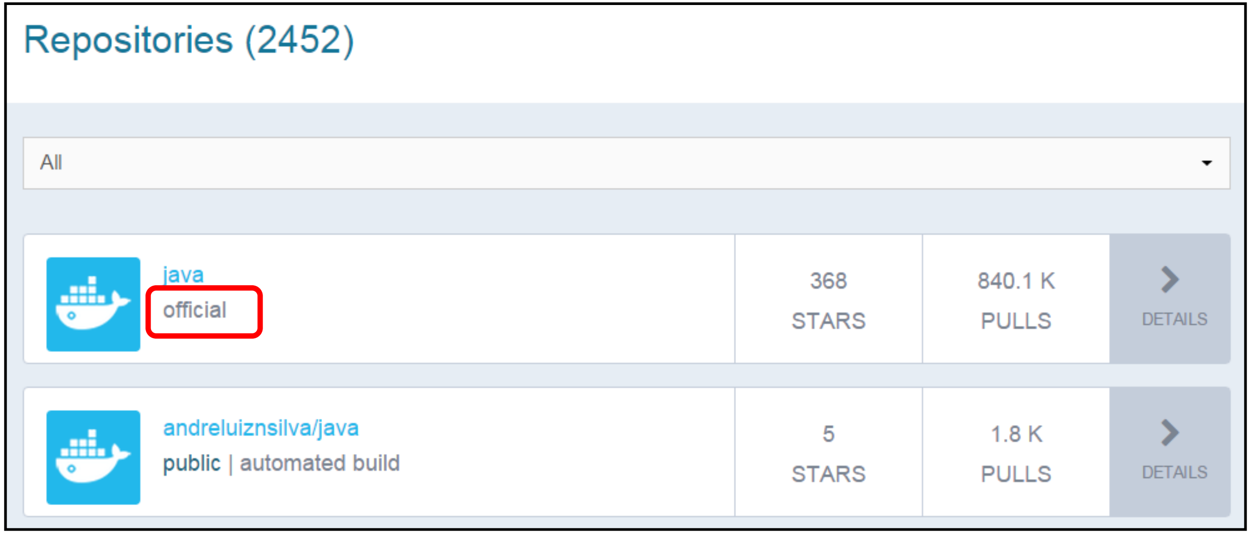
```
johnnytu@docker-ubuntu:~$ docker search java
NAME                   DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
node                   Node.js is a JavaScript-based platform for...   679       [OK]
java                   Java is a concurrent, class-based, and obj...   180       [OK]
maxexcloo/java         Docker framework container with the Oracle...   6                    [OK]
netflixoss/java        Java Base for NetflixOSS container images       4                    [OK]
alsanium/java          Java Development Kit (JDK) image for Docker     3                    [OK]
andreluiznsilva/java   Docker images for java applications             3                    [OK]
denvazh/java           Lightweight Java based on Alpine Linux Doc...   2                    [OK]
nimmis/java-centos     This is docker images of CentOS 7 with dif...   2                    [OK]
isuper/java-oracle     This repository contains all java releases...   2                    [OK]
nimmis/java            This is docker images of Ubuntu 14.04 LTS ...   1                    [OK]
pallet/java                                                            1                    [OK]
isuper/java-openjdk    This repository contains all OpenJDK java ...   1                    [OK]
lwieske/java-8         Oracle Java 8 Container                         1                    [OK]
webratio/java          Java (https://www.java.com/) image              1                    [OK]
```

Official repositories

- Official repositories are a certified and curated set of Docker repositories that are promoted on Docker Hub
- Repositories come from vendors such as NGINX, Ubuntu, Red Hat, Redis, etc…
- Images are **supported by their maintainers**, optimised and up to date
- Official repository images are a mixture of
  - Base images for Linux operating systems (Ubuntu, CentOS etc…)
  - Images for popular development tools, programming languages, web and application servers, data stores

Identifying an official repository

- There are a few ways to tell if a repository is official
  - Marked on the `OFFICIAL` column in the terminal output
  - Repository is labelled “official” on the Docker Hub search results
  - Can filter search results to only display official repositories



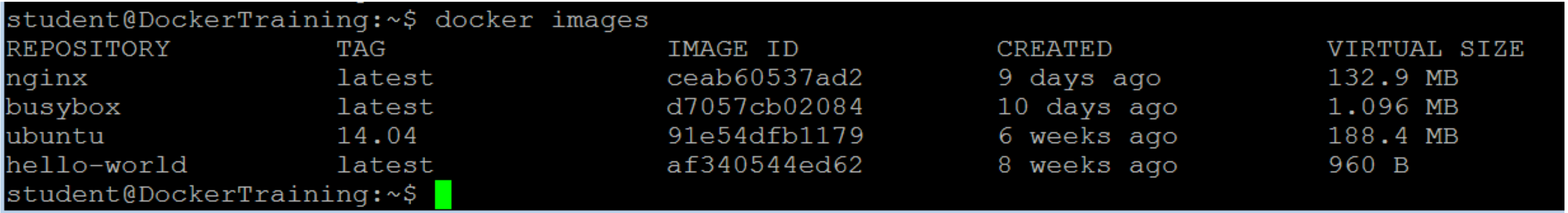
Repositories (2452)

All

java official	368 STARS	840.1 K PULLS	DETAILS
andreluiznsilva/java public | automated build	5 STARS	1.8 K PULLS	DETAILS

Display Local Images

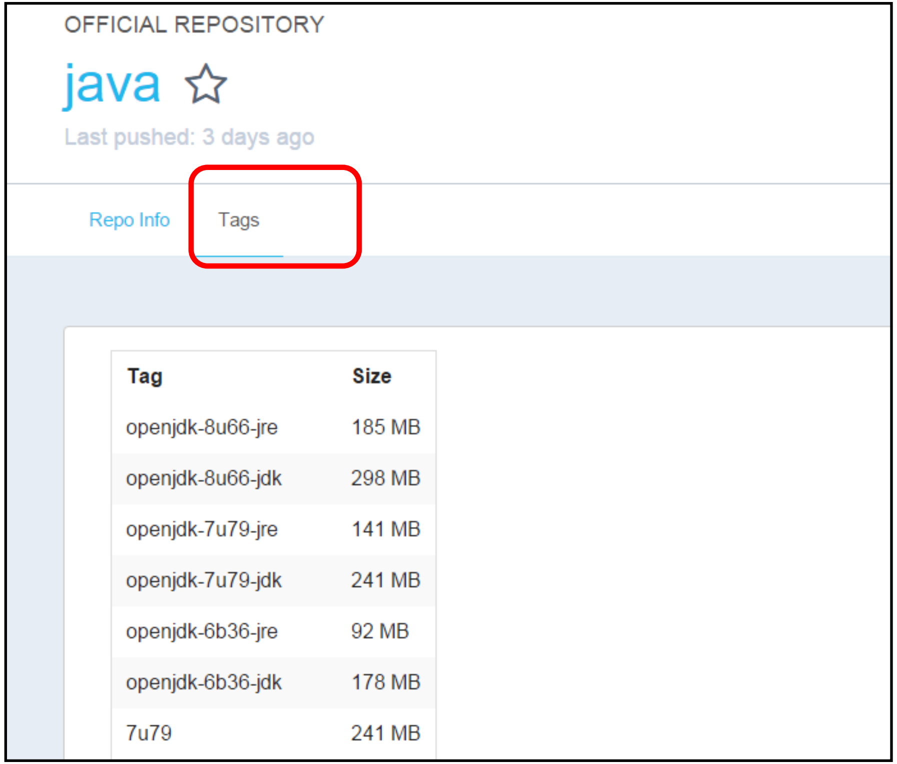
- Run `docker images`
- When creating a container Docker will attempt to use a local image first
- If no local image is found, the Docker daemon will look in Docker Hub unless another registry is specified



```
student@DockerTraining:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
nginx               latest              ceab60537ad2        9 days ago          132.9 MB
busybox             latest              d7057cb02084        10 days ago         1.096 MB
ubuntu              14.04               91e54dfb1179        6 weeks ago         188.4 MB
hello-world         latest              af340544ed62        8 weeks ago         960 B
student@DockerTraining:~$
```

Image Tags

- Images are specified by **repository:tag**
- The same image may have multiple tags
- The default tag is `latest`
- Look up the repository on Docker Hub to see what tags are available



OFFICIAL REPOSITORY

java ☆

Last pushed: 3 days ago

Tag	Size
openjdk-8u66-jre	185 MB
openjdk-8u66-jdk	298 MB
openjdk-7u79-jre	141 MB
openjdk-7u79-jdk	241 MB
openjdk-6b36-jre	92 MB
openjdk-6b36-jdk	178 MB
7u79	241 MB

Pulling images

- To download an image from Docker Hub or any registry, use `docker pull` command
- When running a container with the `docker run` command, images are automatically pulled if no local copy is found

Pull the latest image from the Ubuntu repository in Docker Hub

```
docker pull ubuntu
```

Pull the image with tag 12.04 from Ubuntu repository in Docker Hub

```
docker pull ubuntu:12.04
```

Running and Managing Containers

Creating and running a Container

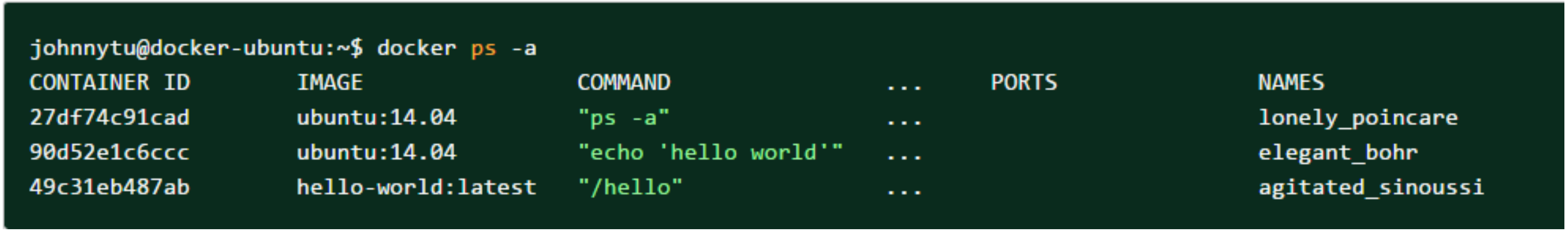
- Use `docker run` command
- The `docker run` command actually does two things
  - Creates the container using the image we specify
  - Runs the container
- Syntax
  `docker run [options] [image] [command] [args]`
- Image is specified with `repository:tag`

Examples

```
docker run ubuntu:14.04 echo “Hello World”
docker run ubuntu ps ax
```

Find your Containers

- Use **`docker ps`** to list running containers
- The `-a` flag to list all containers (includes containers that are stopped)



```
johnnytu@docker-ubuntu:~$ docker ps -a
CONTAINER ID        IMAGE                COMMAND                ...     PORTS               NAMES
27df74c91cad        ubuntu:14.04         "ps -a"                ...                         lonely_poincare
90d52e1c6ccc        ubuntu:14.04         "echo 'hello world'"   ...                         elegant_bohr
49c31eb487ab        hello-world:latest   "/hello"               ...                         agitated_sinoussi
```

Container with Terminal

- Use `-i` and `-t` flags with docker run
- The `-i` flag tells docker to connect to STDIN on the container
- The `-t` flag specifies to get a pseudo-terminal
- **Note:** You need to run a terminal process as your command (e.g. `bash`)

Example

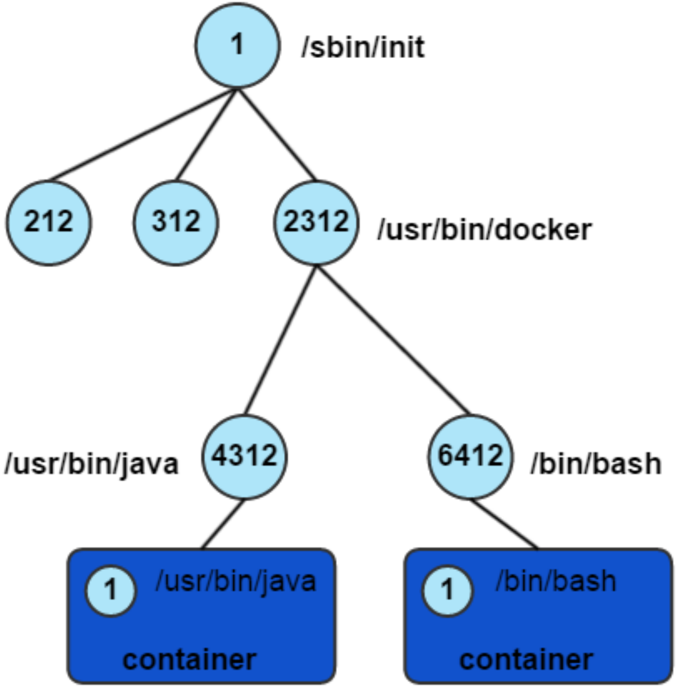
```
docker run -i -t ubuntu:latest bash
```

Exit the Terminal

- Type `exit` to quit the terminal and return to your host terminal
- Exiting the terminal will shutdown the container
- To exit the terminal without a shutdown, hit **`CTRL + P + Q`** together

Container Processes

- A container only runs as long as the process from your specified `docker run` command is running
- Your command’s process is always PID 1 inside the container



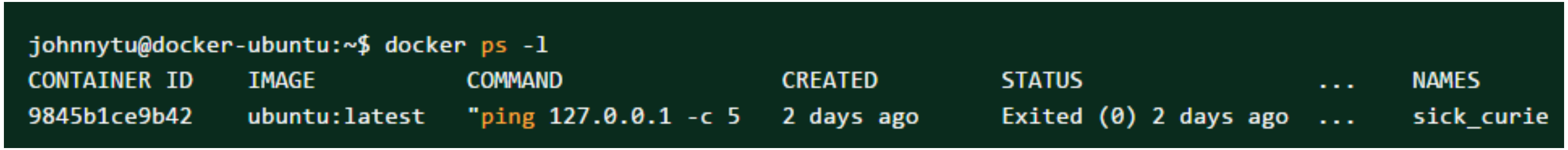
Container ID

- Containers can be specified using their ID or name
- Long ID and short ID
- Short ID and name can be obtained using `docker ps` command to list containers
- Long ID obtained by inspecting a container or by using the `--no-trunc` flag on the `docker ps` command.

  ```
  docker ps -a --no-trunc
  ```

docker ps command

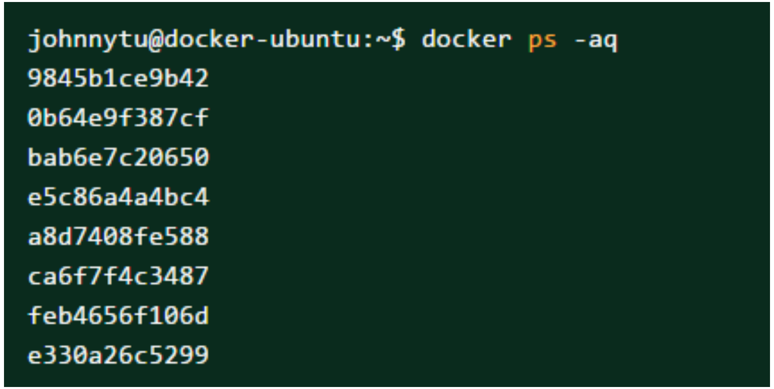
- To view only the container ID’s (displays short ID)
  `docker ps -q`
- The view the last container that was started
  `docker ps -l`



```
johnnytu@docker-ubuntu:~$ docker ps -l
CONTAINER ID    IMAGE           COMMAND                 CREATED         STATUS                      ...     NAMES
9845b1ce9b42    ubuntu:latest   "ping 127.0.0.1 -c 5    2 days ago      Exited (0) 2 days ago       ...     sick_curie
```

docker ps command

- Combining flags to list all containers with only their short ID
  `docker ps -aq`
- Combining flags to list the short ID of the last container started
  `docker ps -lq`



```
johnnytu@docker-ubuntu:~$ docker ps -aq
9845b1ce9b42
0b64e9f387cf
bab6e7c20650
e5c86a4a4bc4
a8d7408fe588
ca6f7f4c3487
feb4656f106d
e330a26c5299
```

Running in Detached Mode

- Also known as running in the background or as a daemon
- Use `-d` flag
- To observe output use **`docker logs [container id]`**

Create a centos container and run the ping command to ping the container itself 50 times

```
docker run -d centos:7 ping 127.0.0.1 -c 50
```

A More Practical Container

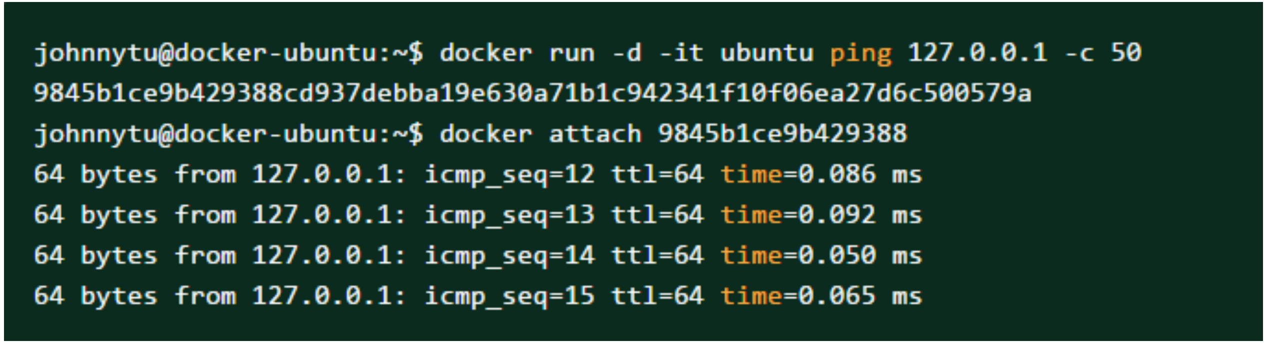
- Run a web application inside a container
- The `-P` flag to map container ports to host ports

Create a container using the nginx image, run in detached mode and map the nginx ports to the host port

```
docker run -d -P nginx
```

Attaching to a container

- Attaching a client to a container will bring a container which is running in the background into the foreground
- The containers PID 1 process output will be displayed on your terminal
- Use `docker attach` command and specify the container ID or name
- **Warning:** Attaching to containers is error prone because if you hit `CTRL + C` by accident, you will stop the process and therefore stop the container



```
johnnytu@docker-ubuntu:~$ docker run -d -it ubuntu ping 127.0.0.1 -c 50
9845b1ce9b429388cd937debba19e630a71b1c942341f10f06ea27d6c500579a
johnnytu@docker-ubuntu:~$ docker attach 9845b1ce9b429388
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.086 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.092 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.065 ms
```

Detaching from a container

- Hit CTRL + P + Q together on your terminal
- Only works if the following two conditions are met
  - The container standard input is connected
  - The container has been started with a terminal
  - For example: `docker run -i -t ubuntu`
- Hitting CTRL + C will terminate the process, thus shutting down the container

Docker exec command

- `docker exec` command allows us to execute additional processes inside a container
- Typically used to gain command line access
- `docker exec -i -t [container ID] bash`
- Exiting from the terminal will not terminate the container

Inspecting container logs

- Container PID 1 process output can be viewed with **`docker logs`** command
- Will show whatever PID 1 writes to stdout and stderr
- Displays the entire log output from the time the container was created

View the output of the containers PID 1 process

```
docker logs <container name>
```

Tailing container logs

- We can specify to only show the last “x” number of lines from the logs
- Use `--tail` option and specify the number of lines
- Use the `--follow` option or `-f` to get a streaming output from the log

Show the last 5 lines from the container log

```
docker logs --tail 5 <container ID>
```

Show the last 5 lines and follow the log

```
docker logs --tail 5 -f <container ID>
```

Stopping a container

- Two commands we can use
  - `docker stop`
  - `docker kill`
- `docker stop` sends a SIGTERM to the main container process
  - Process then receives a SIGKILL after a grace period
  - Grace period can be specified with -t flag (default is 10 seconds)
- `docker kill` sends a SIGKILL immediately to the main container process

Restarting a container

- Use `docker start` to restart a container that has been stopped
- Container will start using the same options and command specified previously
- Can attach to the container with `-a` flag

Start a stopped container and attach to the process that it is running

```
docker start -a <container ID>
```

Deleting containers

- Can only delete containers that have been stopped
- Use `docker rm` command
- Specify the container ID or name
- To delete a container that is still running, use `-f` option

```
docker rm -f <container ID>
```

Delete all containers

- Use `docker ps -aq` to list the id’s of all containers
- Feed the output into `docker rm` command
- Output will print an error message for containers that are still running

Delete all containers that are stopped

```
docker rm $(docker ps -aq)
```

Delete all containers (including the ones still running)

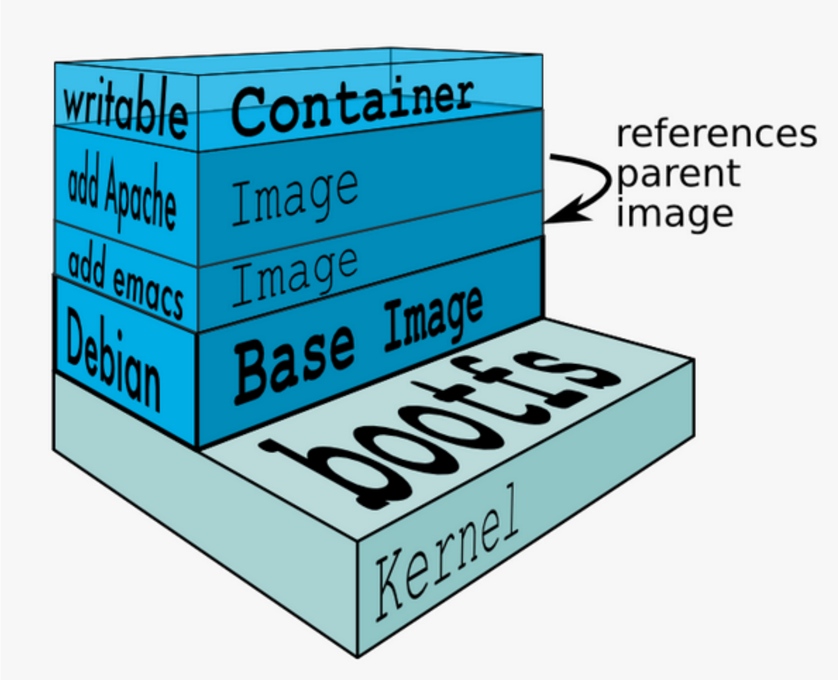
```
docker rm -f $(docker ps -aq)
```

Building Images

docker

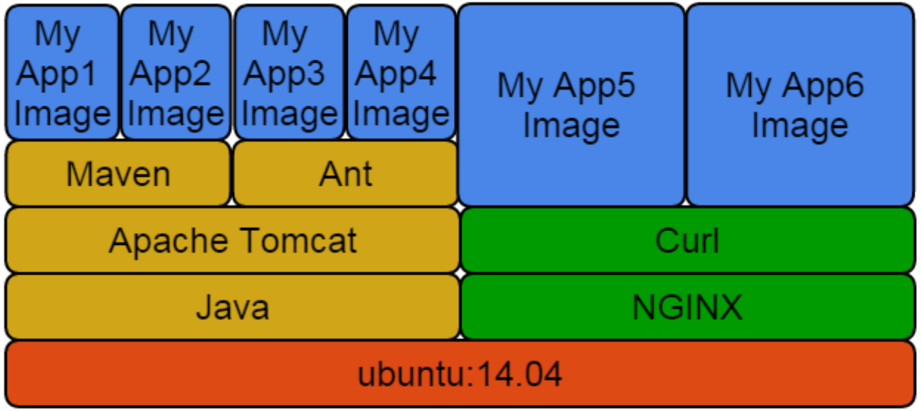
Understanding image layers

- An image is a collection of files and some meta data
- Images are comprised of multiple layers
- A layer is also just another image
- Each image contains software you want to run
- Every image contains a base layer
- Docker uses a copy on write system
- Layers are read only



Sharing layers

- Images can share layers in order to speed up transfer times and optimize disk and memory usage
- Parent images that already exists on the host do not have to be downloaded



My App1 Image

My App2 Image

My App3 Image

My App4 Image

My App5 Image

My App6 Image

Maven

Ant

Apache Tomcat

Curl

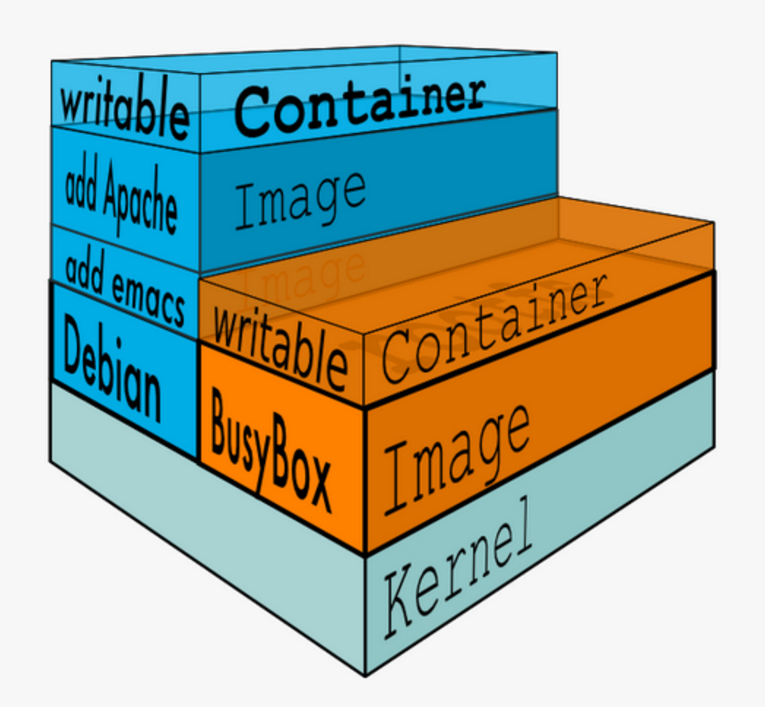
Java

NGINX

ubuntu:14.04

The container writable layer

- Docker creates a top writable layer for containers
- Parent images are read only
- All changes are made at the writeable layer
- When changing a file from a read only layer, the copy on write system will copy the file into the writable layer



Methods of building images

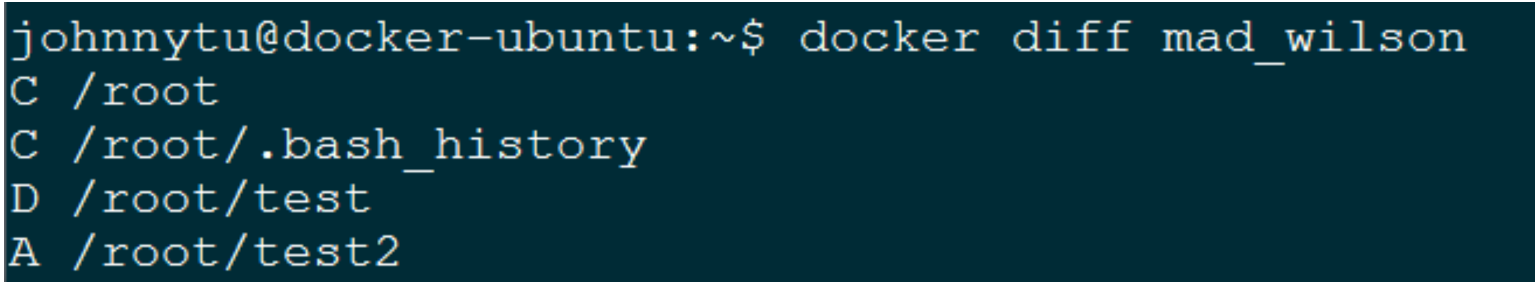
- Three ways
  - Commit changes from a container as a new image
  - Build from a Dockerfile
  - Import a tarball into Docker as a standalone base layer

Committing changes in a container

- Allows us to build images interactively
- Get terminal access inside a container and install the necessary programs and your application
- Then save the container as a new image using the `docker commit` command

Comparing container changes

- Use the `docker diff` command to compare a container with its parent image
  - Recall that images are read only and changes occur in a new layer
  - The parent image (the original) is being compared with the new layer
- Copy on write system ensures that starting a container from a large image does not result in a large copy operation
- Lists the files and directories that have changed



```
johnnytu@docker-ubuntu:~$ docker diff mad_wilson
C /root
C /root/.bash_history
D /root/test
A /root/test2
```

Docker Commit

- `docker commit` command saves changes in a container as a new image
- Syntax
  ```
  docker commit [options] [container ID] [repository:tag]
  ```
- Repository name should be based on username/application
- Can reference the container with container name instead of ID

Save the container with ID of 984d25f537c5 as a new image in the repository johnnytu/myapplication. Tag the image as 1.0

```
docker commit 984d25f537c5 johnnytu/myapplication:1.0
```

Intro to Dockerfile

*A **Dockerfile** is a configuration file that contains instructions for building a Docker image*

- Provides a more effective way to build images compared to using `docker commit`
- Easily fits into your development workflow and your continuous integration and deployment process

Process for building images from Dockerfile

1. Create a Dockerfile in a new folder or in your existing application folder
2. Write the instructions for building the image
   - What programs to install
   - What base image to use
   - What command to run
3. Run `docker build` command to build an image from the Dockerfile

Dockerfile Instructions

- Instructions specify what to do when building the image
- **FROM** instruction specifies what the base image should be
- **RUN** instruction specifies a command to execute
- Comments start with “#”

```
#Example of a comment
FROM ubuntu:14.04
RUN apt-get install vim
RUN apt-get install curl
```

FROM instruction

- Must be the first instruction specified in the Dockerfile (not including comments)
- Can be specified multiple times to build multiple images
  - Each `FROM` marks the beginning of a new image
- Can use any image including, images from official repositories, user images and images in self hosted registries.

Examples

```
FROM ubuntu
FROM ubuntu:14.04
FROM johnnytu/myapplication:1.0
FROM company.registry:5000/myapplication:1.0
```

More about RUN

- `RUN` will do the following:
  - Execute a command.
  - Record changes made to the filesystem.
  - Works great to install libraries, packages, and various files.
- `RUN` will NOT do the following:
  - Record state of *processes*.
  - Automatically start daemons.

Docker Build

- Syntax

  ```
  docker build [options] [path]
  ```

- Common option to tag the build

  ```
  docker build -t [repository:tag] [path]
  ```

Build an image using the current folder as the context path. Put the image in the johnnytu/myimage repository and tag it as 1.0

```
docker build -t johnnytu/myimage:1.0 .
```

As above but use the myproject folder as the context path

```
docker build -t johnnytu/myimage:1.0 myproject
```

Multiple commands in a single RUN Instruction

- Use the shell syntax “&&” to combine multiple commands in a single RUN instruction
- Commands will all be run in the same container and committed as a new image at the end
- Reduces the number of image layers that are produced

```
RUN apt-get update && apt-get install -y \
      curl \
      vim \
      openjdk-7-jdk
```

CMD Instruction

- `CMD` defines a default command to execute when a container is created
- Shell format and EXEC format
- Can only be specified once in a Dockerfile
  - If specified multiple times, the last CMD instruction is executed
- **Can be overridden at run time**

Shell format

```
CMD ping 127.0.0.1 –c 30
```

Exec format

```
CMD [“ping”, “127.0.0.1”, “-c”, “30”]
```

ENTRYPOINT Instruction

- Defines the command that will run when a container is executed
- Run time arguments and `CMD` instruction are passed as parameters to the `ENTRYPOINT` instruction
- Shell and EXEC form
- Container essentially runs as an executable

```
ENTRYPOINT [“ping”]
```

Using CMD with ENTRYPOINT

- If `ENTRYPOINT` is used, the `CMD` instruction can be used to specify default parameters
- Parameters specified during `docker run` will override `CMD`
- If no parameters are specified during `docker run`, the `CMD` arguments will be used for the `ENTRYPOINT` command

Shell vs exec format

- The RUN, CMD and ENTRYPOINT instructions can be specified in either shell or exec form

In shell form, the command will run inside a shell with `/bin/sh -c`

```
RUN apt-get update
```

Exec format allows execution of command in images that don’t have `/bin/sh`

```
RUN [“apt-get”, “update”]
```

Copying source files

- When building “real” images you would want to do more than just install some programs
- Examples
  - Compile your source code and run your application
  - Copy configuration files
  - Copy other content
- How do we get our content on our host into the container?
- Use the `COPY` instruction

COPY instruction

- The `COPY` instruction copies new files or directories from a specified **source** and adds them to the container filesystem at a specified **destination**
- Syntax

  ```
  COPY <src> <dest>
  ```
- The `<src>` path must be inside the build context
- If the `<src>` path is a directory, all files in the directory are copied. The directory itself is not copied
- You can specify multiple `<src>` directories

COPY examples

Copy the server.conf file in the build context into the root folder of the container

```
COPY server.conf /
```

Copy the files inside the `data/server` folder of the build context into the `/data/server` folder of the container

```
COPY data/server /data/server
```

Dockerize an application

- The Dockerfile is essential if we want to adapt our existing application to run on containers
- Take a simple Java program as an example. To build and run it, we need the following on our host
  - The Java Development Kit (JDK)
  - The Java Virtual Machine (JVM)
  - Third party libraries depending on the application itself
- You compile the code, run the application and everything looks good

Dockerize an application

- Then you distribute the application and run it on a different environment and it fails
- Reasons why the Java application fails?
  - Missing libraries in the environment
  - Missing the JDK or JVM
  - Wrong version of libraries
  - Wrong version of JDK or JVM
- So why not run your application in a Docker container?
- Install all the necessary libraries in the container
- Build and run the application inside the container and distribute the image for the container
- Will run on any environment with the Docker Engine installed

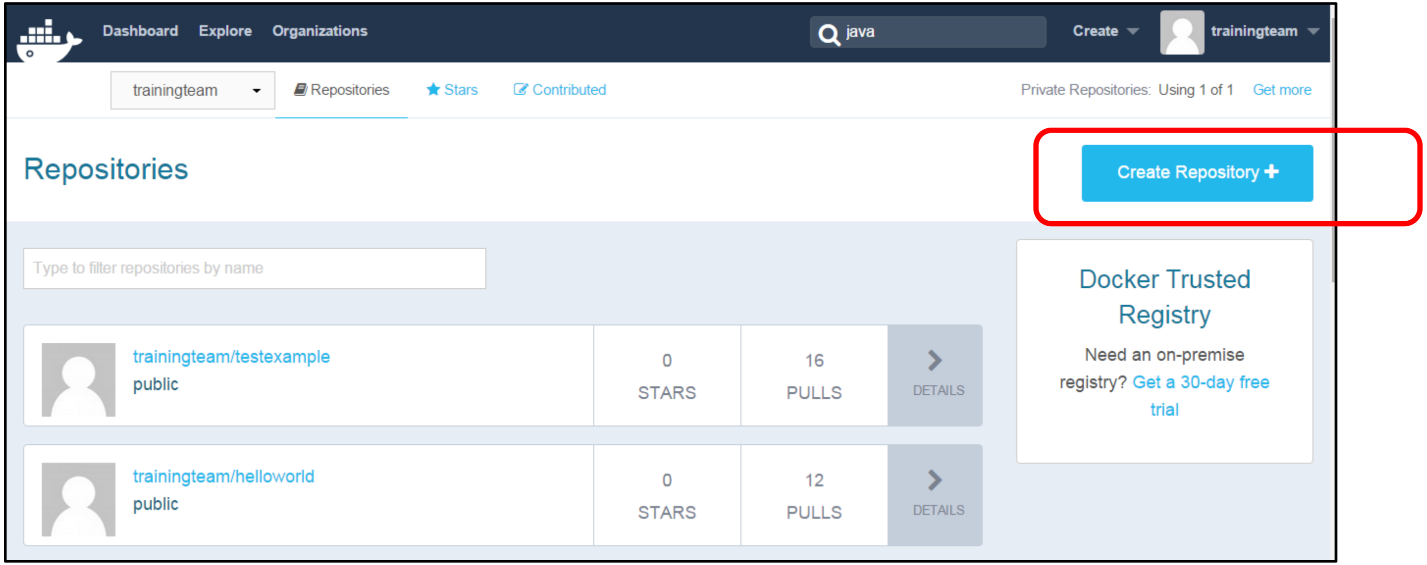
Managing and Distributing Images

Distributing your image

- To distribute your image there are two options
  - Push to Docker Hub
  - Push to your own registry server
  - `docker export` and `docker import` commands
- Images in Docker Hub can reside in public or private repositories
- Registry server can be setup to be publicly available or behind the firewall

Docker Hub Repositories

- Users can create their own repositories on Docker Hub
- Public and Private
- Push local images to a repository

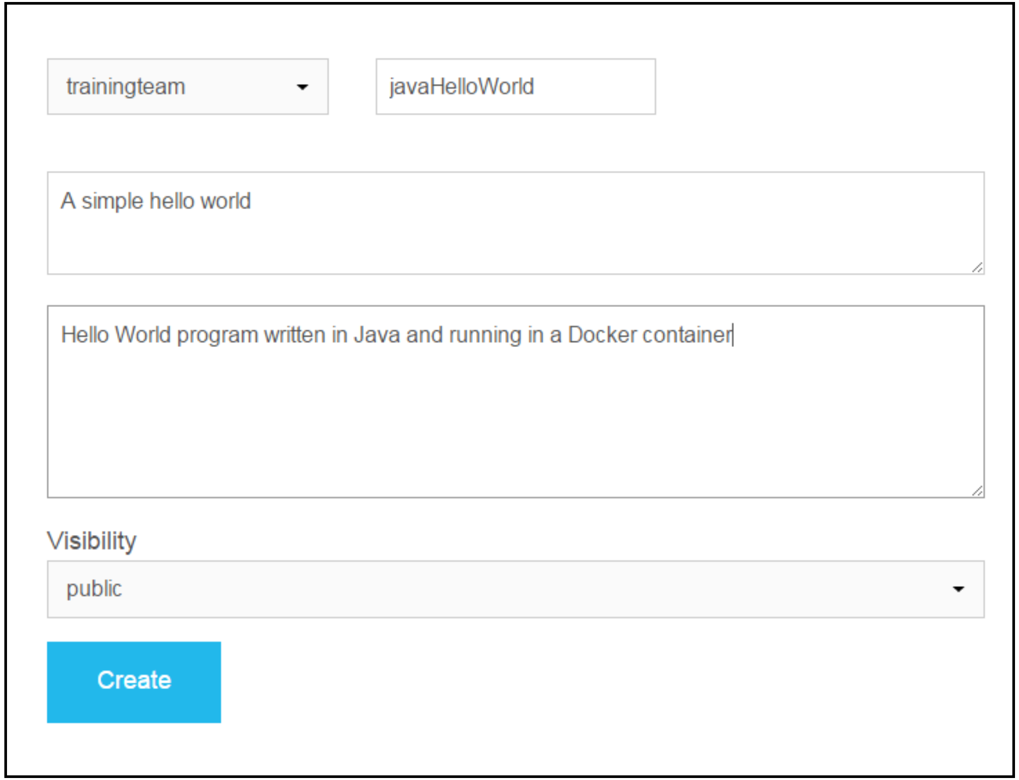


trainingteam/testexample public	0 STARS	16 PULLS	DETAILS
trainingteam/helloworld public	0 STARS	12 PULLS	DETAILS

Creating a repository

- Repository will reside in the user or organization namespace. For example:
  - trainingteam/myrepo
  - johnny/myrepo
- Public repositories are listed and searchable for public use
- Anyone can pull images from a public repository

Creating a repository



trainingteam

javaHelloWorld

A simple hello world

Hello World program written in Java and running in a Docker container

Visibility

public

Repository description

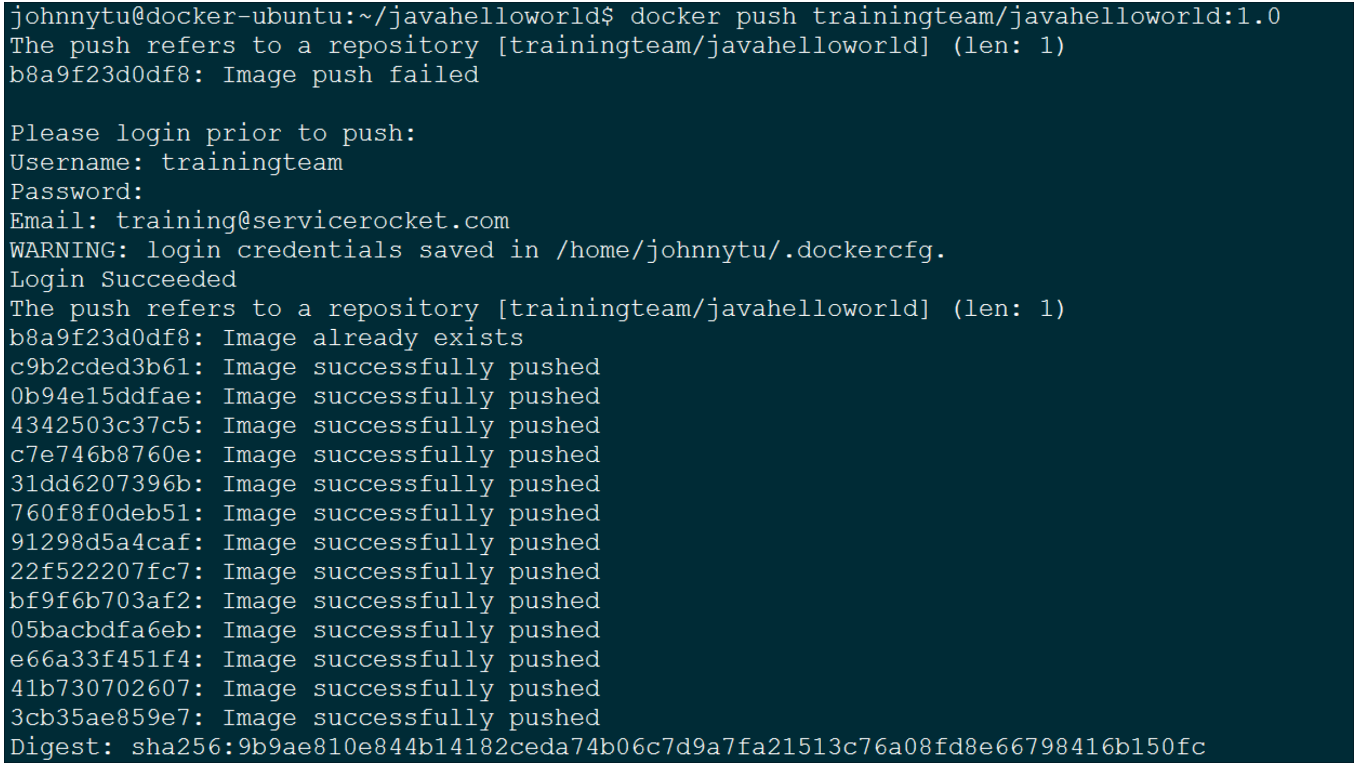
- Once a repository has been created we can write a detailed description about the images
- Good to include instructions on how to run the images
- You may want to
  - Link to the source repository of the application the image is designed to run
  - Link to the source of the Dockerfile
- Description is written in markdown http://daringfireball.net/projects/markdown/syntax

Pushing Images to Docker Hub

- Use **`docker push`** command
- Syntax

  ```
  docker push [repo:tag]
  ```
- Local repo must have same name and tag as the Docker Hub repo
- Only the image layers that have changed get pushed
- You will be prompted to login to your Docker Hub account
- **Note:** You don’t need to create the repository on Docker Hub first.
  - If you push a local repository that does not exist on Docker Hub, it will be automatically created

Pushing Images



```
johnnytu@docker-ubuntu:~/javahelloworld$ docker push trainingteam/javahelloworld:1.0
The push refers to a repository [trainingteam/javahelloworld] (len: 1)
b8a9f23d0df8: Image push failed

Please login prior to push:
Username: trainingteam
Password:
Email: training@servicerocket.com
WARNING: login credentials saved in /home/johnnytu/.dockercfg.
Login Succeeded
The push refers to a repository [trainingteam/javahelloworld] (len: 1)
b8a9f23d0df8: Image already exists
c9b2cded3b61: Image successfully pushed
0b94e15ddfae: Image successfully pushed
4342503c37c5: Image successfully pushed
c7e746b8760e: Image successfully pushed
31dd6207396b: Image successfully pushed
760f8f0deb51: Image successfully pushed
91298d5a4caf: Image successfully pushed
22f522207fc7: Image successfully pushed
bf9f6b703af2: Image successfully pushed
05bacbdfa6eb: Image successfully pushed
e66a33f451f4: Image successfully pushed
41b730702607: Image successfully pushed
3cb35ae859e7: Image successfully pushed
Digest: sha256:9b9ae810e844b14182ceda74b06c7d9a7fa21513c76a08fd8e66798416b150fc
```

Tagging Images

- Used to rename a local image repository before pushing to Docker Hub
- Syntax:
  `docker tag [image ID] [repo:tag]`
  OR
  `docker tag [local repo:tag] [Docker Hub repo:tag]`

Tag image with ID (trainingteam/testexample is the name of repository on Docker hub)

```
docker tag edfc212de17b trainingteam/testexample:1.0
```

Tag image using the local repository tag

```
docker tag johnnytu/testimage:1.5 trainingteam/testexample
```

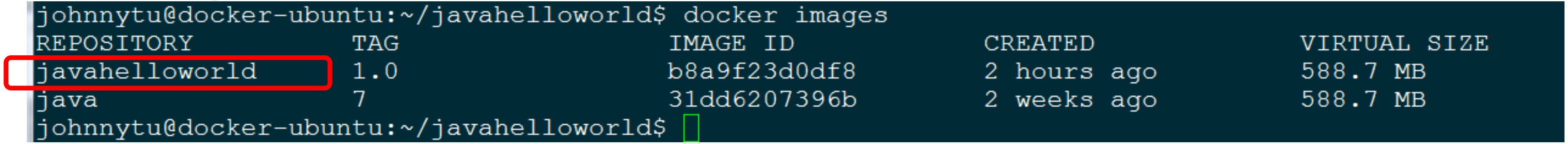
One image, many tags

- The same image can have multiple tags
- Image can be identified by it’s ID
  - The ID is generated using a hash of the image content for consistency

```
REPOSITORY                    TAG                 IMAGE ID            CREATED             VIRTUAL
SIZE
trainingteam/javahelloworld   1.1                 76b3b2455967        5 minutes ago       598.1 MB
trainingteam/testimage        1.0                 ee8800b0677b        8 minutes ago       263.8 MB
javahelloworld                1.0                 b8a9f23d0df8        3 hours ago         588.7 MB
javahelloworld                latest              b8a9f23d0df8        3 hours ago         588.7 MB
trainingteam/javahelloworld   1.0                 b8a9f23d0df8        3 hours ago         588.7 MB
java                          7                   31dd6207396b        2 weeks ago         588.7 MB
ubuntu                        14.04               07f8e8c5e660        2 weeks ago         188.3 MB
```

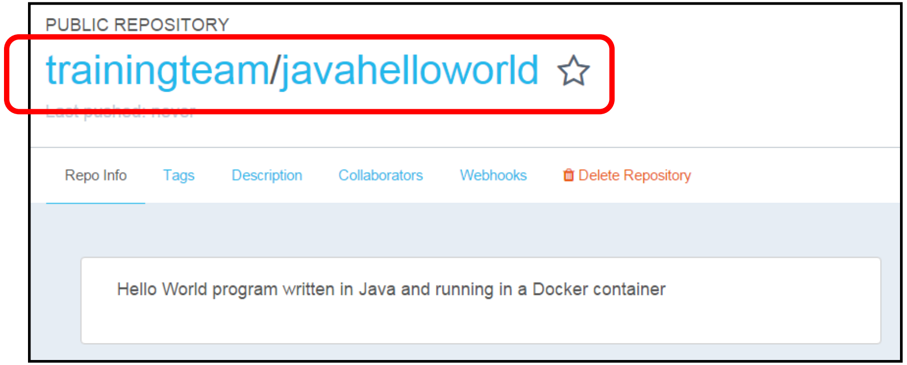
Tagging images for a push

Local repo name



```
johnnytu@docker-ubuntu:~/javahelloworld$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
javahelloworld      1.0                 b8a9f23d0df8        2 hours ago         588.7 MB
java                7                   31dd6207396b        2 weeks ago         588.7 MB
johnnytu@docker-ubuntu:~/javahelloworld$ 
```

Repo name on Docker Hub



PUBLIC REPOSITORY

trainingteam/javahelloworld ☆

Last pushed: never

Hello World program written in Java and running in a Docker container

Deleting local Images

- Use **`docker rmi`** command
- **`docker rmi [image ID]`**
  or
  `docker rmi [repo:tag]`
- If an image is tagged multiple times, remove each tag

```
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
test1               latest              cbfa5ab76a11        12 seconds ago      262.5 MB
test                latest              cbfa5ab76a11        12 seconds ago      262.5 MB

johnnytu@dockertraining:~/test$ docker rmi test
Untagged: test:latest
johnnytu@dockertraining:~/test$ docker rmi test1
Untagged: test1:latest
Deleted: cbfa5ab76a11eec84b751ae261d3f870a0be61bb899e651c857ae4cc3eed9bc9
```