Docker Recap 2019

Arijeet Majumdar

04-Jan-2020

Docker Delhi Meetup

The Evolution of Docker Platform

2013

Debut of Docker
Docker released to Open Source

2014

Docker 1.0 Release
Docker Hub replaced DockerIndex
Docker Swarm was released
First Dockercon

2015

Runc
Docker Plugins

2016

Docker Store was Announced
Docker for Mac & Windows
Docker for AWS and Azure

2017

Rise of Docker EE 1.0
Moby Project, LinuxKit. MTA

2018

Docker Desktop Enterprise
Support for Kubernetes

2019

Rise of DKS
Docker Enterprise 3.0

Docker Products Offerings

SaaS	Desktop	On Server & Cloud	From Ecosystem
DockerHub	Docker Desktop for Mac	Docker Enterprise (EE)	Docker Certified
	Docker Desktop for Windows	Universal Control Plane	Certified Containers
	Docker Desktop Enterprise	Docker Trusted Registry	Certified Plugins
		Docker Engine(Enterprise)	Certified Infrastructure

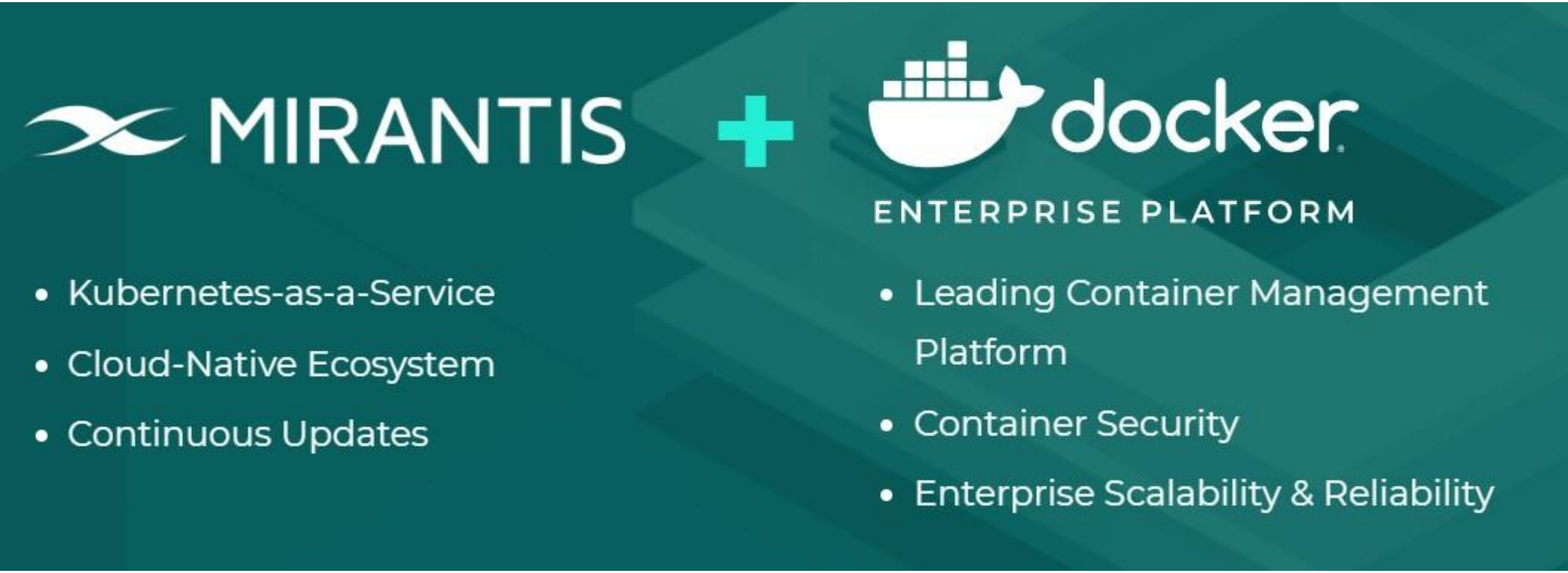
Docker Products Offerings

SaaS	Desktop	On Server & Cloud	From Ecosystem
$ DockerHub	Docker Desktop for Mac	Docker Enterprise (EE)	Docker Certified
	Docker Desktop for Windows	Uni[illegible] C[illegible]Plane	[illegible]d [illegible]ainers
	Docker Desktop Enterprise $	Docker Trusted Registry	Certified Plugins
		Docker Engine(Enterprise)	Certified Infrastructure

MIRANTIS

MIRANTIS

Mirantis Occupied Docker Enterprise in November 2019



MIRANTIS + docker

ENTERPRISE PLATFORM

- Kubernetes-as-a-Service
- Cloud-Native Ecosystem
- Continuous Updates

- Leading Container Management Platform
- Container Security
- Enterprise Scalability & Reliability

Future Road Map

- Mirantis Inheriting all Docker Enterprise customers and contracts(750+), as well as its strategic technology alliances and partner programs.

- Docker will now focus on Docker Desktop, its container developer IDE and platform, Docker Hub, a service for finding and sharing container images & Docker Compose.

Reference

https://www.docker.com/blog/docker-next-chapter-advancing-developer-workflows-for-modern-apps

What’s a Container?

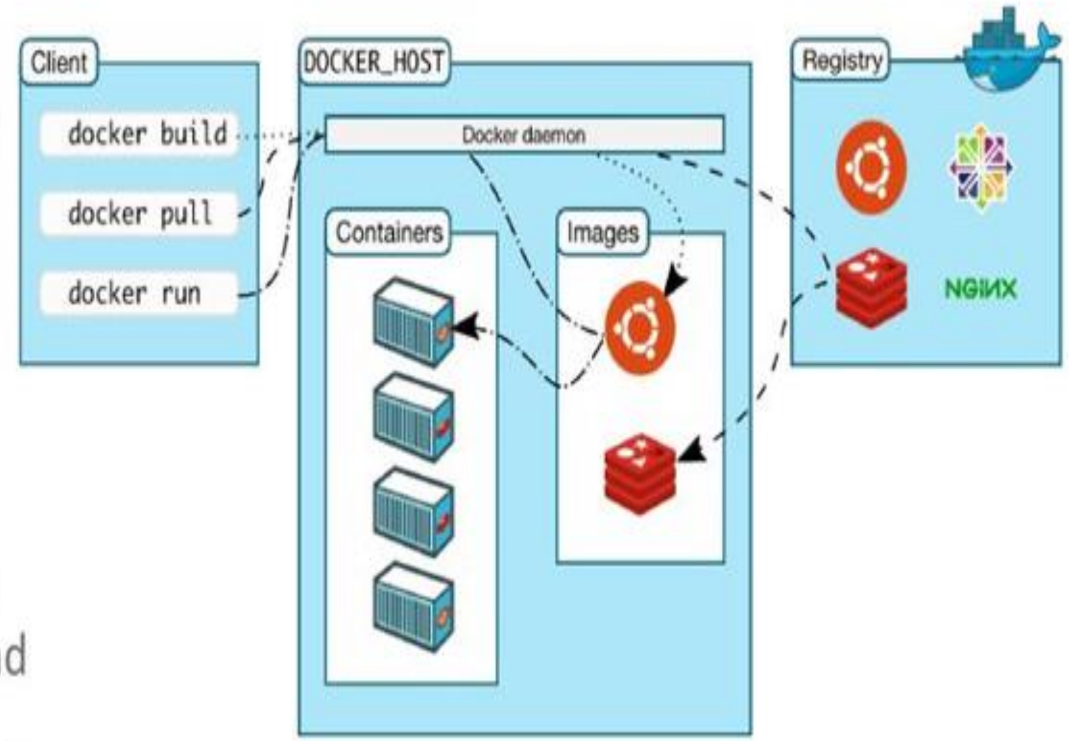
Looks like a
Walks like a
Runs like a

Virtual Machine

Containers are a Sandbox inside Linux Kernel sharing the kernel with separate Network Stack, Process Stack, IPC Stack etc.

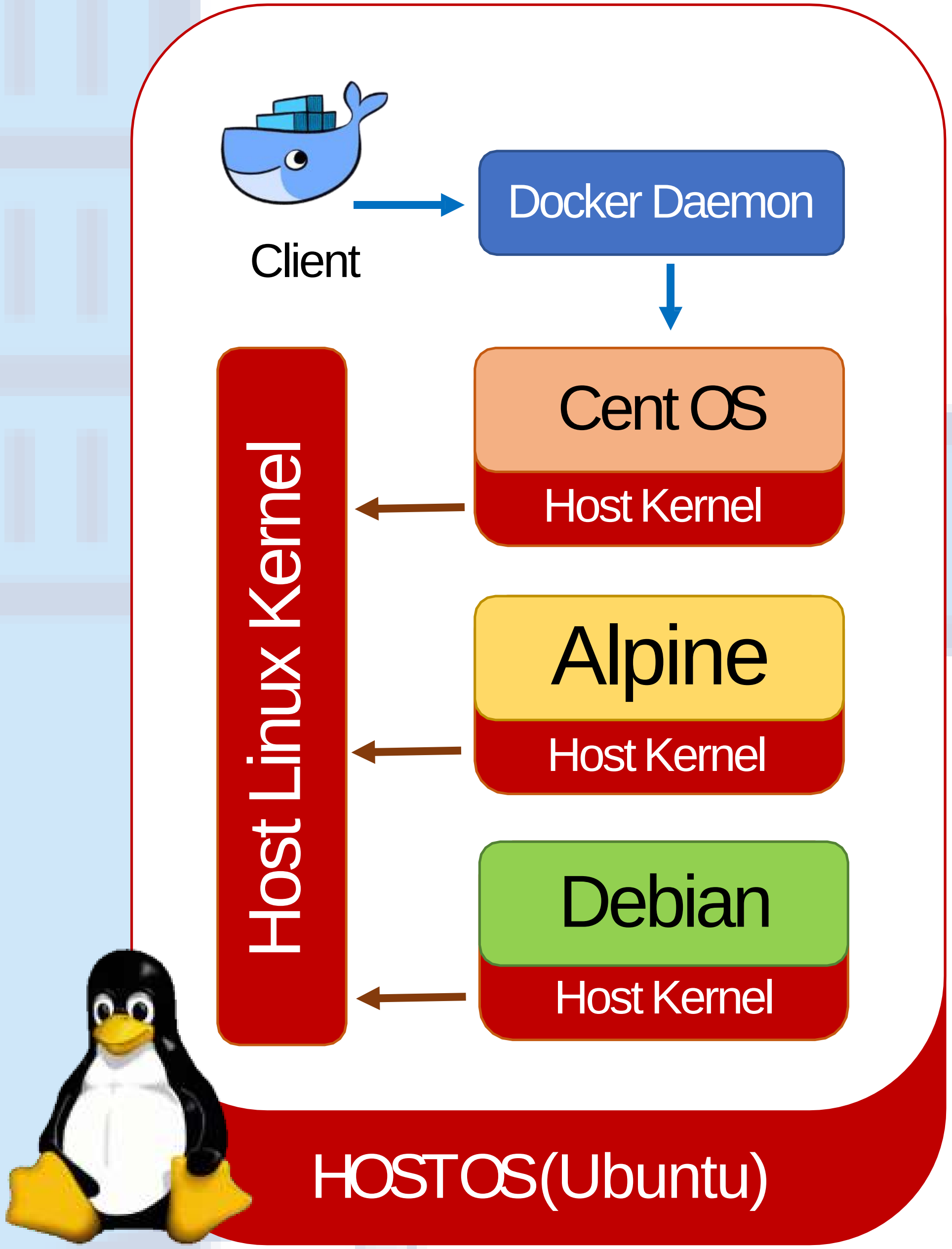
Docker Architecture

- Docker client – Command Line Interface (CLI) for interfacing with the Docker
- Dockerfile – Text file of Docker instructions used to assemble a Docker Image
- Image – Hierarchies of files built from a Dockerfile, the file used as input to the docker build command
- Container – Running instance of an Image using the docker run command
- Registry – Image repository



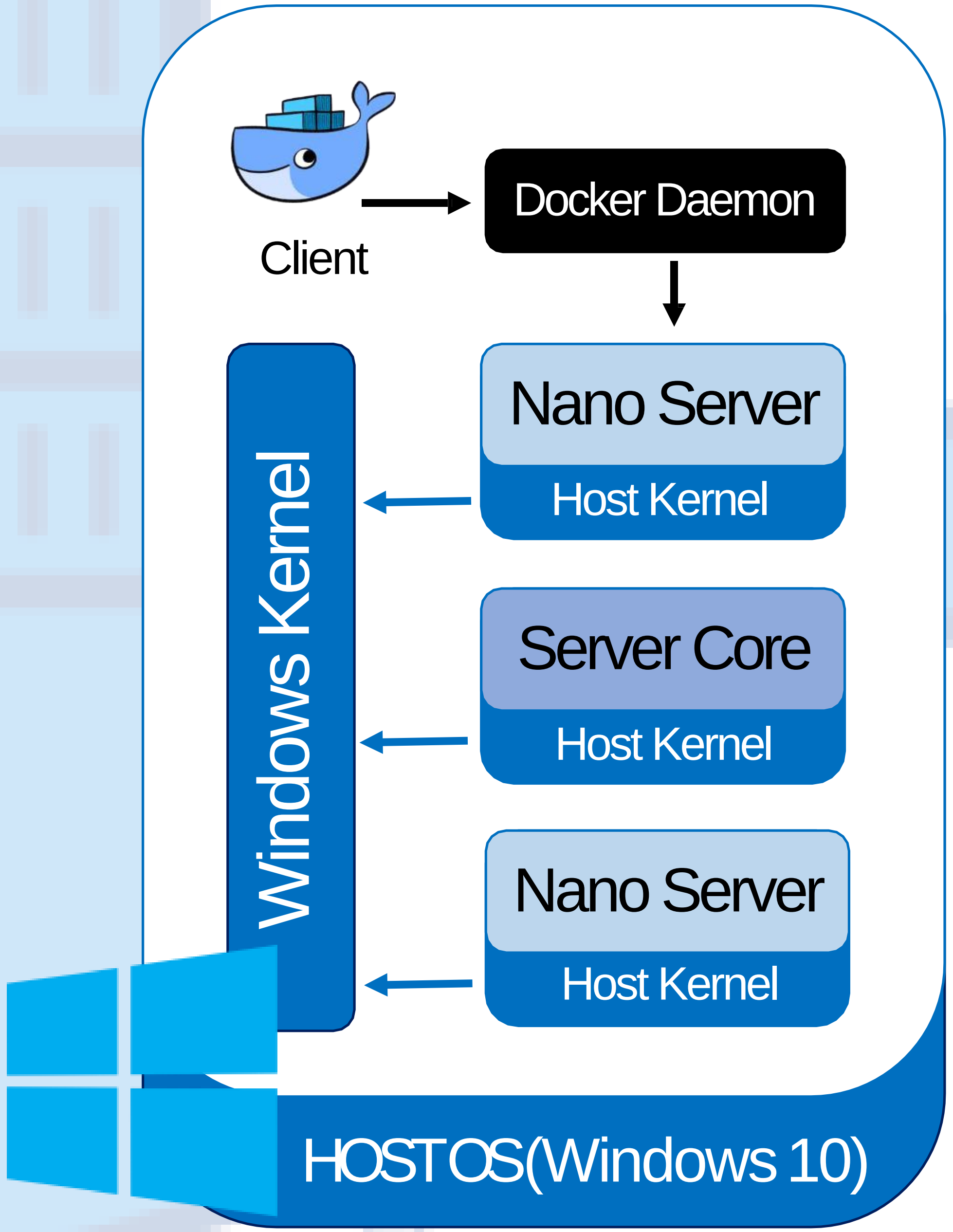
Linux Kernel

All the containers will have the same Host OSKernel
If you require a specific Kernel version then Host Kernel needs to be updated



Windows Kernel

All the containers will have the same Host OSKernel
If you require a specific Kernel version then Host Kernel needs to be updated

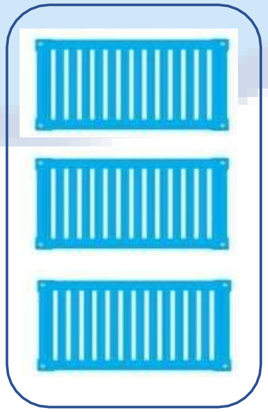


Docker Key Concepts

- Docker images
  - A Docker image is a read-only template.
  - For example, an image could contain an Ubuntu operating system with Apache and your web application installed.
  - Images are used to create Docker containers.
  - Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
  - Docker images are the build component of Docker.
- Docker containers
  - Docker containers are similar to a directory.
  - A Docker container holds everything that is needed for an application to run.
  - Each container is created from a Docker image.
  - Docker containers can be run, started, stopped, moved, and deleted.
  - Each container is an isolated and secure application platform.
  - Docker containers are the run component of Docker.
- Docker Registries
  - Docker registries hold images.
  - These are public or private stores from which you upload or download images.
  - The public Docker registry is called Docker Hub.
  - It provides a huge collection of existing images for your use.
  - These can be images you create yourself or you can use images that others have previously created.
  - Docker registries are the distribution component of Docker.

Images

Containers



What is Docker Image and What isn,t

- An image is a collection of files + some meta data.
  (Technically: those files form the root filesystem of a container.)
- Images are made of *layers*, conceptually stacked on top of each other.
- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.
- App binaries and dependencies.
- Metadata about the image data and how to run the image.
- Official definition:

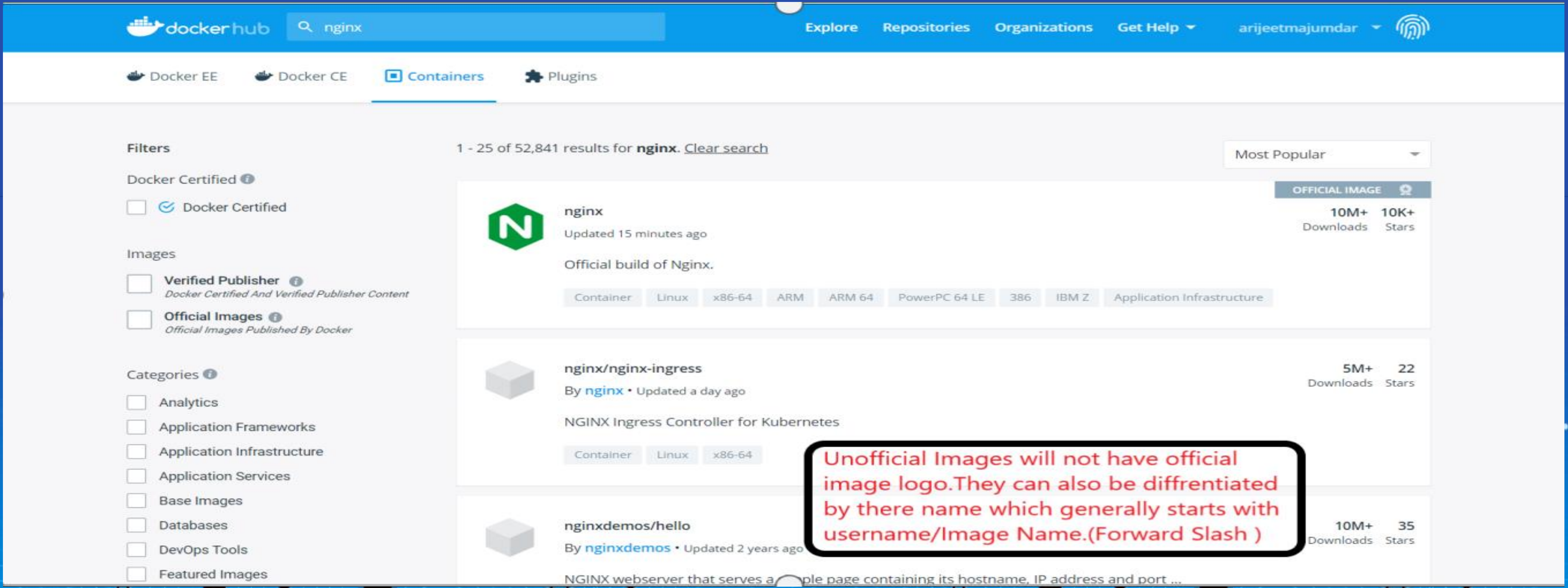
"An Image is an ordered collection of root filesystem changes and the corresponding executionparameters for use within a container runtime."

- Not a complete OS. No kernel, kernel modules (e.g. drivers).

Type of Images

- Basics of Docker Hub (hub.docker.com)
- Find Official and other good public images
- Download images and basics of image tags

Docker Hub



1 - 25 of 52,841 results for **nginx**. Clear search

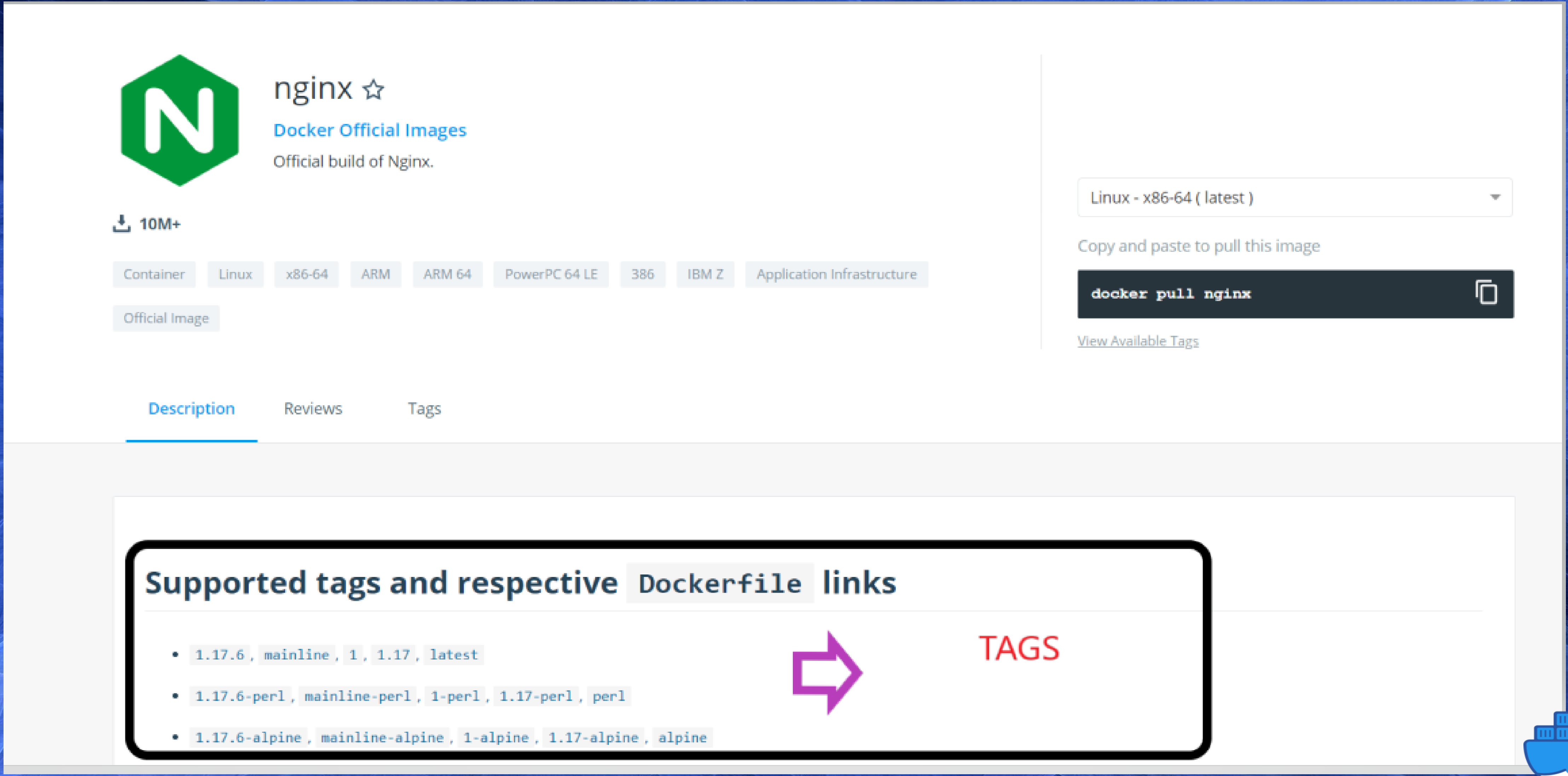
nginx
Updated 15 minutes ago
Official build of Nginx.
Container | Linux | x86-64 | ARM | ARM 64 | PowerPC 64 LE | 386 | IBM Z | Application Infrastructure
10M+ Downloads | 10K+ Stars

nginx/nginx-ingress
By nginx • Updated a day ago
NGINX Ingress Controller for Kubernetes
Container | Linux | x86-64
5M+ Downloads | 22 Stars

nginxdemos/hello
By nginxdemos • Updated 2 years ago
NGINX webserver that serves a ple page containing its hostname, IP address and port ...
10M+ Downloads | 35 Stars

Unofficial Images will not have official image logo.They can also be diffrentiated by there name which generally starts with username/Image Name.(Forward Slash)

TAGS



nginx ☆

Docker Official Images

Official build of Nginx.

10M+

Container | Linux | x86-64 | ARM | ARM 64 | PowerPC 64 LE | 386 | IBM Z | Application Infrastructure | Official Image

Linux - x86-64 (latest)

Copy and paste to pull this image

```
docker pull nginx
```

View Available Tags

Description | Reviews | Tags

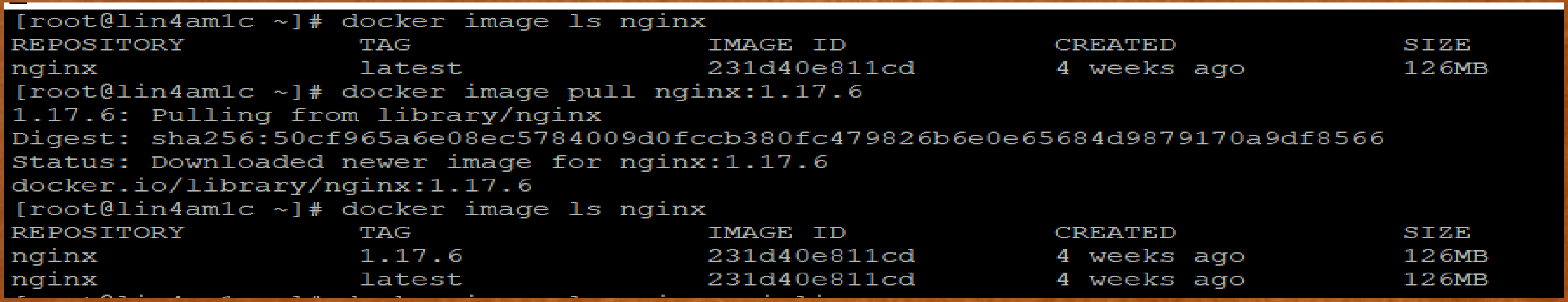
Supported tags and respective `Dockerfile` links

- `1.17.6`, `mainline`, `1`, `1.17`, `latest`
- `1.17.6-perl`, `mainline-perl`, `1-perl`, `1.17-perl`, `perl`
- `1.17.6-alpine`, `mainline-alpine`, `1-alpine`, `1.17-alpine`, `alpine`

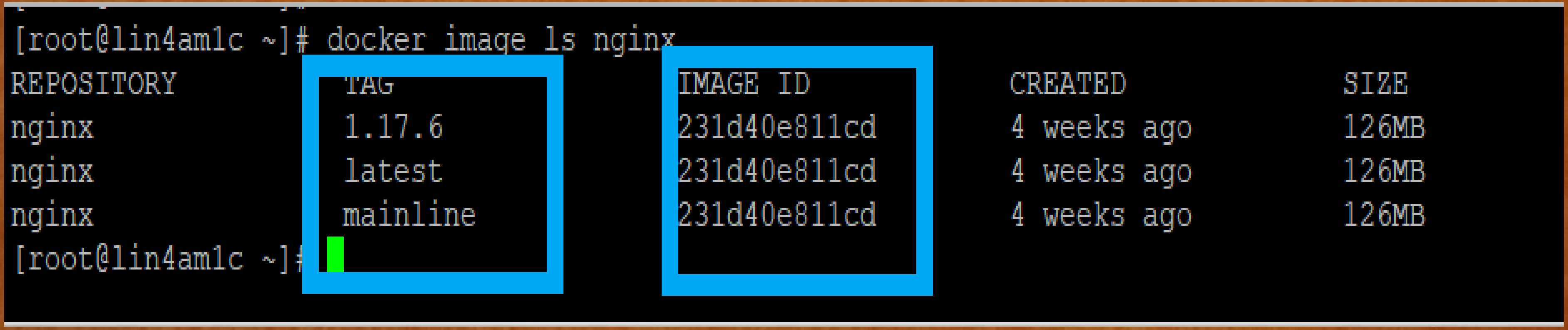
TAGS

Images can have Multiple Tags

Pulling Images of specific version.
Latest is the default tag given to an docker image. Latest is a rolling tag.



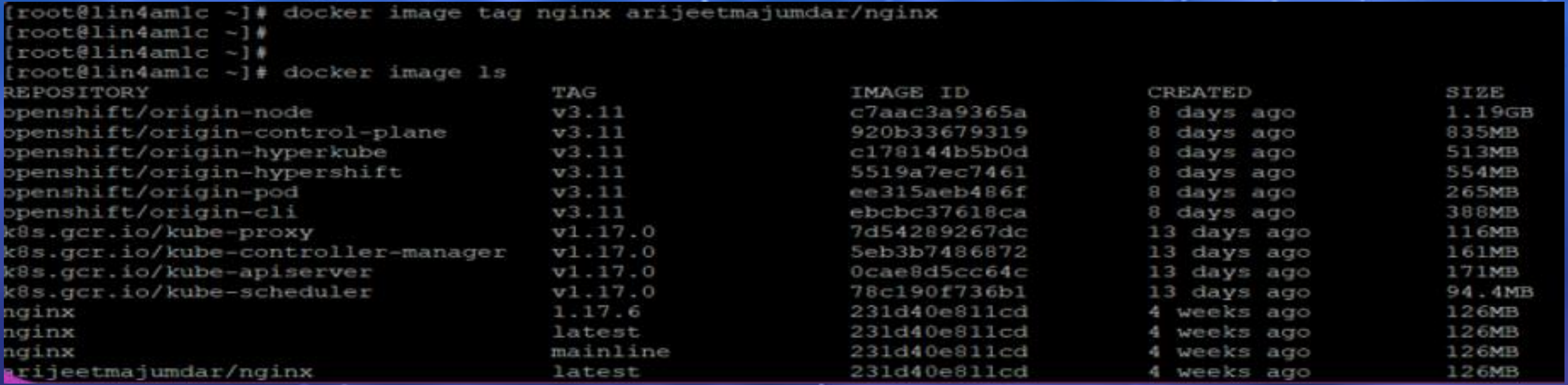
```
[root@lin4am1c ~]# docker image ls nginx
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              231d40e811cd        4 weeks ago         126MB
[root@lin4am1c ~]# docker image pull nginx:1.17.6
1.17.6: Pulling from library/nginx
Digest: sha256:50cf965a6e08ec5784009d0fccb380fc479826b6e0e65684d9879170a9df8566
Status: Downloaded newer image for nginx:1.17.6
docker.io/library/nginx:1.17.6
[root@lin4am1c ~]# docker image ls nginx
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               1.17.6              231d40e811cd        4 weeks ago         126MB
nginx               latest              231d40e811cd        4 weeks ago         126MB
```



```
[root@lin4am1c ~]# docker image ls nginx
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               1.17.6              231d40e811cd        4 weeks ago         126MB
nginx               latest              231d40e811cd        4 weeks ago         126MB
nginx               mainline            231d40e811cd        4 weeks ago         126MB
[root@lin4am1c ~]#
```

Changing Image Tags

```
$     docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

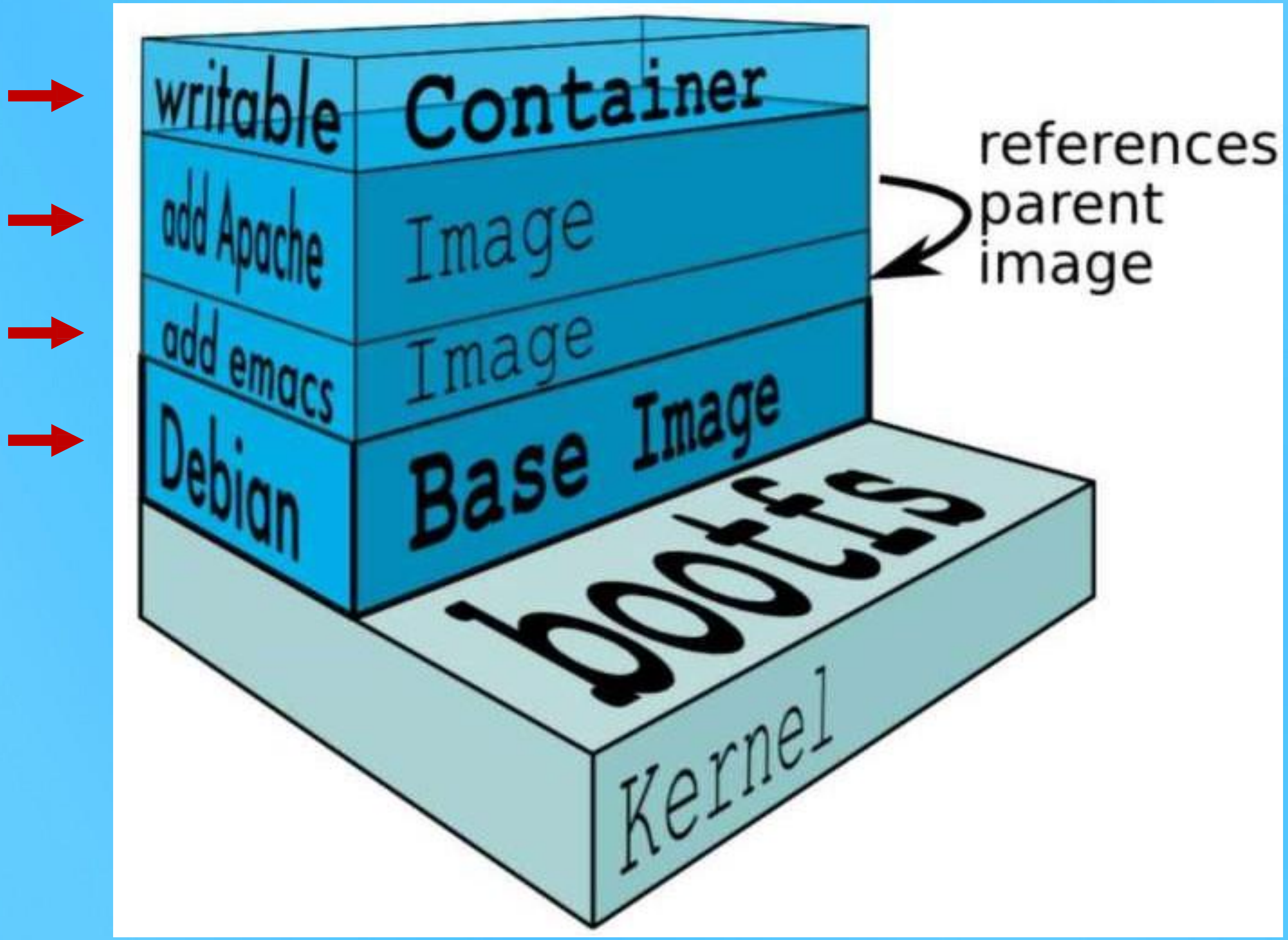


```
[root@lin4am1c ~]# docker image tag nginx arijeetmajumdar/nginx
[root@lin4am1c ~]#
[root@lin4am1c ~]#
[root@lin4am1c ~]# docker image ls
REPOSITORY                           TAG                 IMAGE ID            CREATED             SIZE
openshift/origin-node                v3.11               c7aac3a9365a        8 days ago          1.19GB
openshift/origin-control-plane       v3.11               920b33679319        8 days ago          835MB
openshift/origin-hyperkube           v3.11               c178144b5b0d        8 days ago          513MB
openshift/origin-hypershift          v3.11               5519a7ec7461        8 days ago          554MB
openshift/origin-pod                 v3.11               ee315aeb486f        8 days ago          265MB
openshift/origin-cli                 v3.11               ebcbc37618ca        8 days ago          388MB
k8s.gcr.io/kube-proxy                v1.17.0             7d54289267dc        13 days ago         116MB
k8s.gcr.io/kube-controller-manager   v1.17.0             5eb3b7486872        13 days ago         161MB
k8s.gcr.io/kube-apiserver            v1.17.0             0cae8d5cc64c        13 days ago         171MB
k8s.gcr.io/kube-scheduler            v1.17.0             78c190f736b1        13 days ago         94.4MB
nginx                                1.17.6              231d40e811cd        4 weeks ago         126MB
nginx                                latest              231d40e811cd        4 weeks ago         126MB
nginx                                mainline            231d40e811cd        4 weeks ago         126MB
arijeetmajumdar/nginx                latest              231d40e811cd        4 weeks ago         126MB
```

Agenda

- Image layers
- Union file system
- history and inspect commands
- Copy on write
- Pushing image to Docker Hub

Docker Image structure



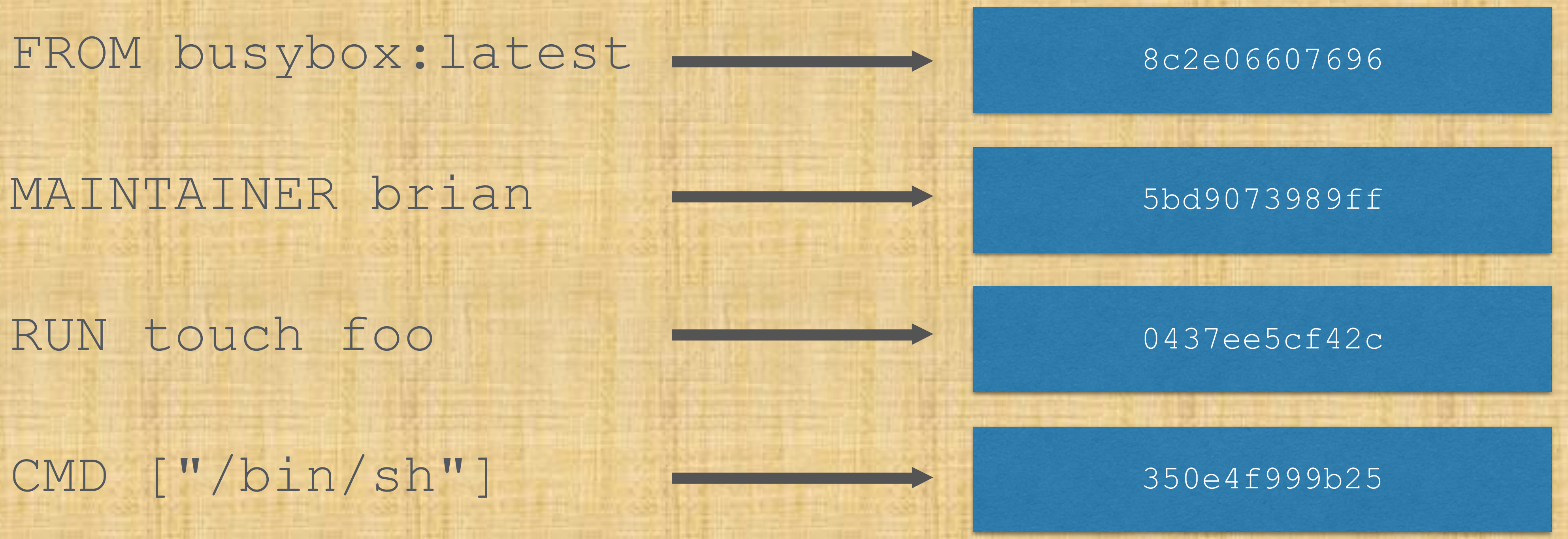
- Images are read-only.
- Multiple layers of image gives the final Container.
- Layers can be sharable.
- Layers are portable.

Image and Their Layers: Review

- Images are made up of file system changes and metadata
- Each layer is uniquely identified and only stored once on a host
- This saves storage space on host and transfer time on push/pull
- A container is just a single read/write layer on top of image
- docker image history and inspect commands can teachus

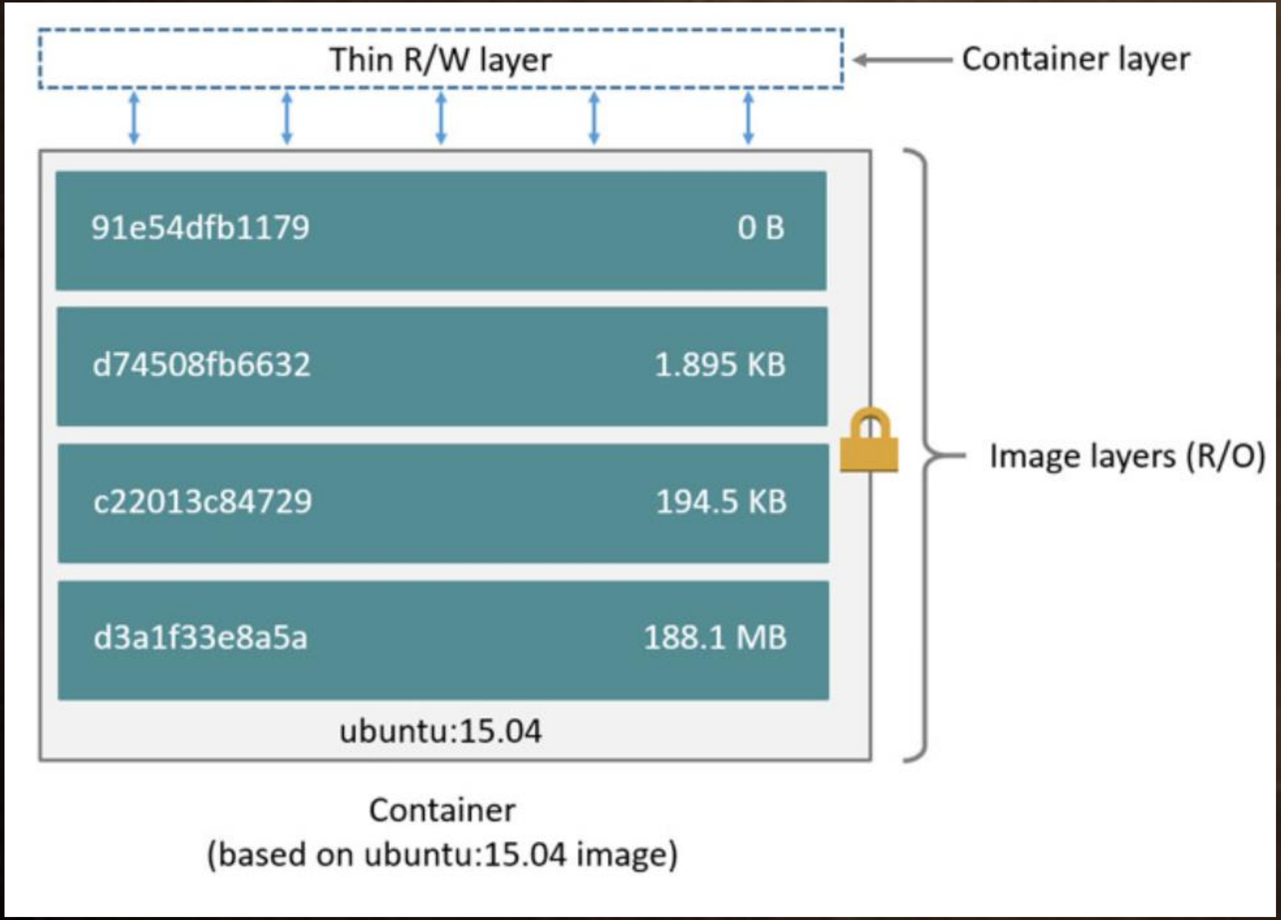
Image Layers

- Each Dockerfile instruction generates a new layer



```
FROM busybox:latest
MAINTAINER brian
RUN touch foo
CMD ["/bin/sh"]
```

Union File system



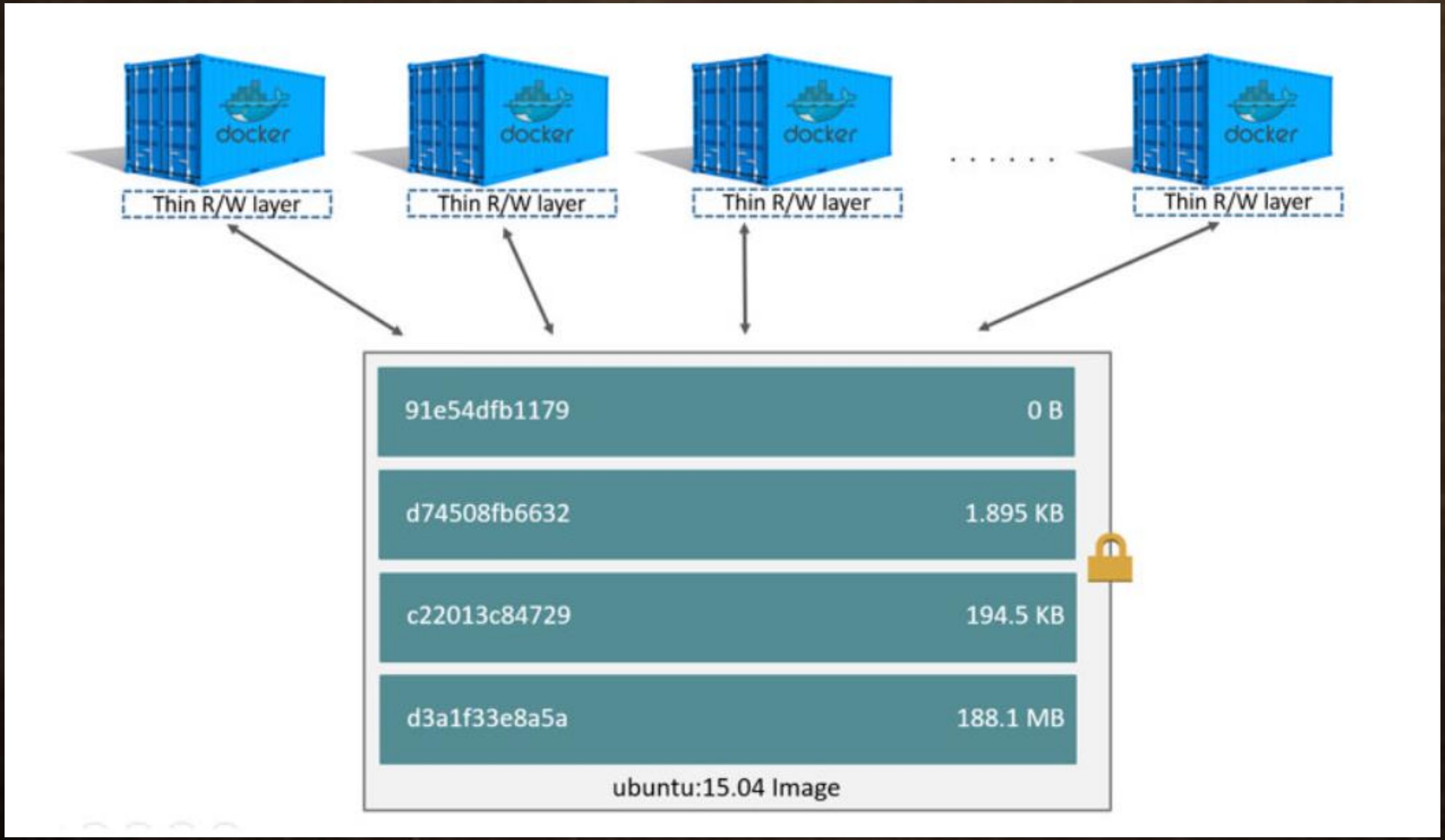
1.Docker Images are actually just multiple Union File Systems stacked on top of each other.
2. **Logical merge** of multiple layers.

3.Read-only **lower layers**, writable **upper layer**.

4. Start reading from the **upper layer** than defaults to **lower layers**.

5. Copy on Write (CoW) mechanism is enabled.

Multiple Container Deployed out of same image



What are container layers?

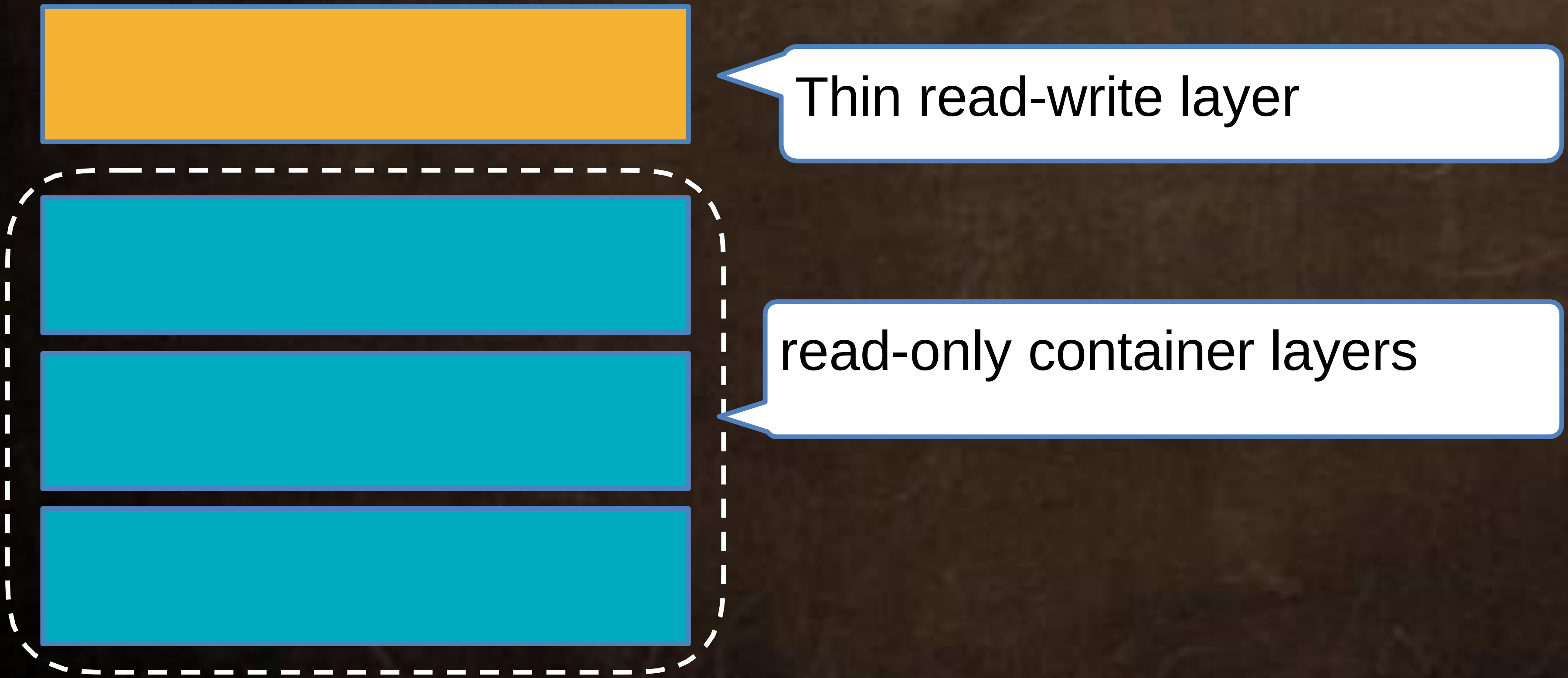
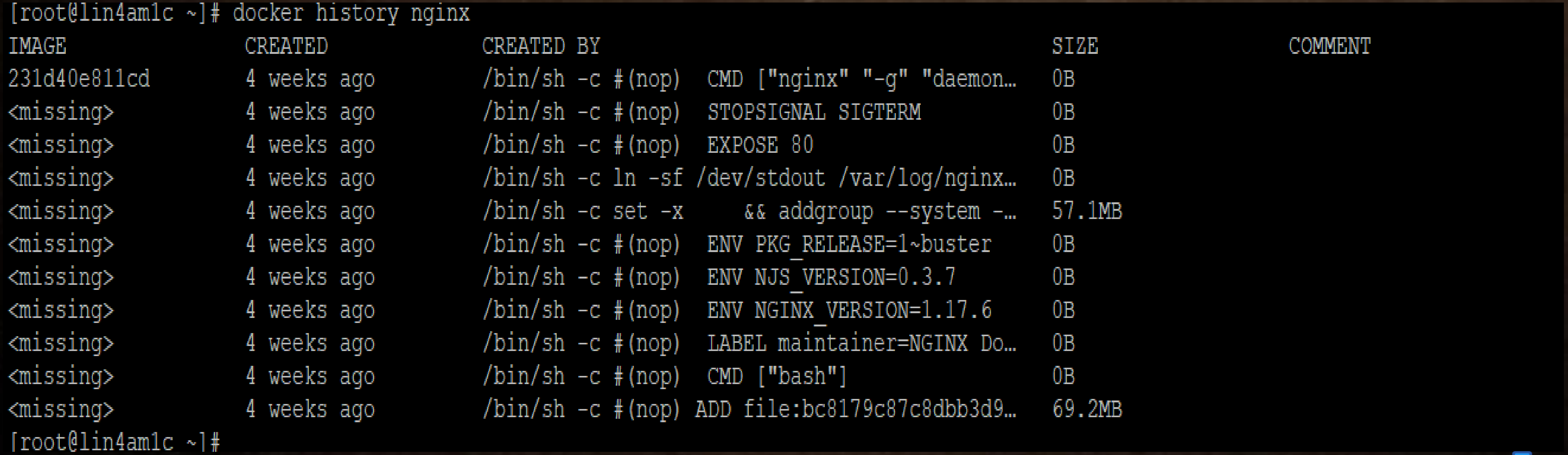


Image Layers

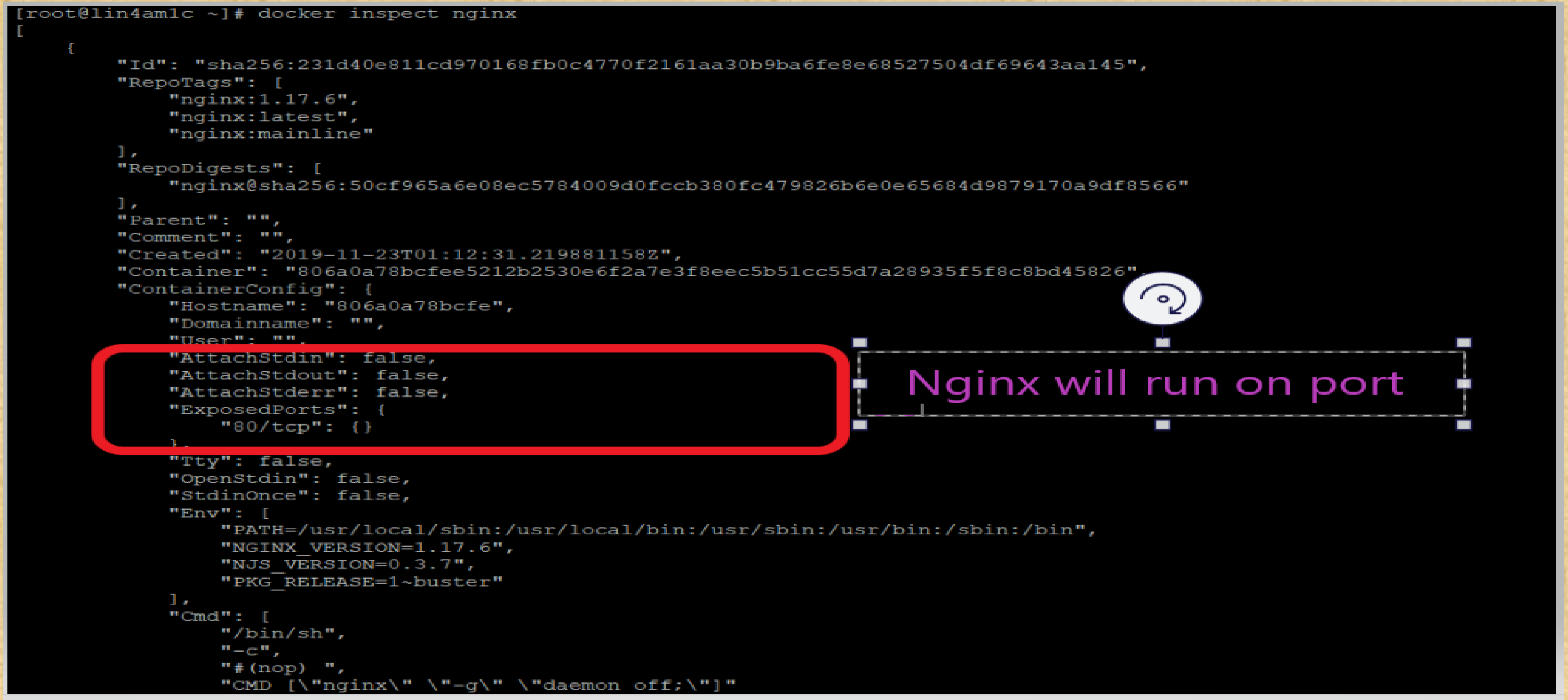
- To see the various layers of Docker Image use the below command.



```
[root@lin4am1c ~]# docker history nginx
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
231d40e811cd        4 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           4 weeks ago         /bin/sh -c ln -sf /dev/stdout /var/log/nginx…   0B
<missing>           4 weeks ago         /bin/sh -c set -x     && addgroup --system -…   57.1MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.3.7        0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.17.6     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:bc8179c87c8dbb3d9…   69.2MB
[root@lin4am1c ~]#
```

Inspecting Images

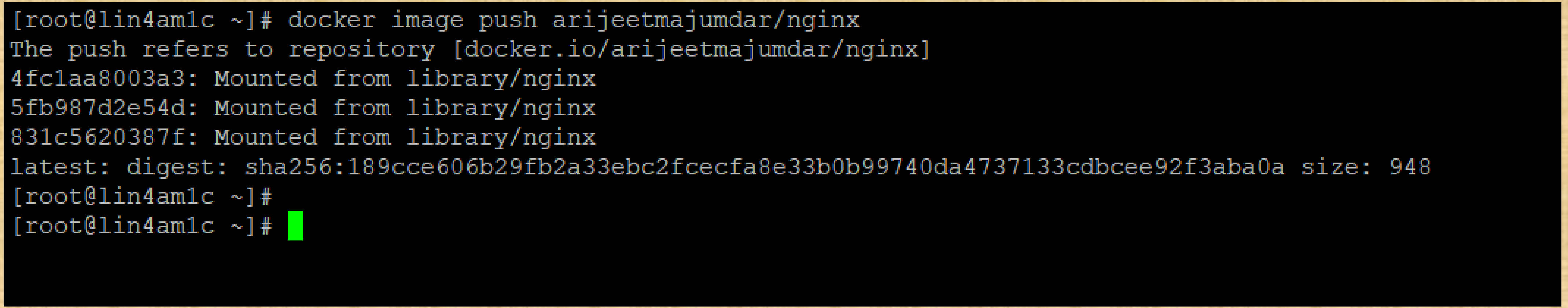
- To inspect the metadata of the images use the following command.



```
[root@lin4am1c ~]# docker inspect nginx
[
    {
        "Id": "sha256:231d40e811cd970168fb0c4770f2161aa30b9ba6fe8e68527504df69643aa145",
        "RepoTags": [
            "nginx:1.17.6",
            "nginx:latest",
            "nginx:mainline"
        ],
        "RepoDigests": [
            "nginx@sha256:50cf965a6e08ec5784009d0fccb380fc479826b6e0e65684d9879170a9df8566"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2019-11-23T01:12:31.219881158Z",
        "Container": "806a0a78bcfee5212b2530e6f2a7e3f8eec5b51cc55d7a28935f5f8c8bd45826",
        "ContainerConfig": {
            "Hostname": "806a0a78bcfe",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.17.6",
                "NJS_VERSION=0.3.7",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"nginx\" \"-g\" \"daemon off;\"]"
```

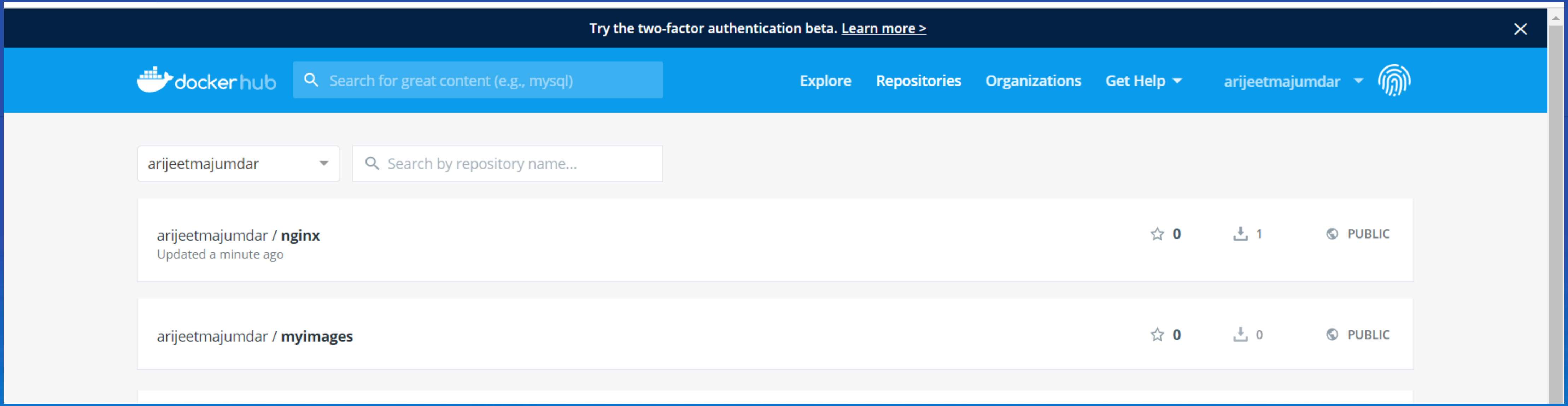
Pushing images to docker hub

$ docker image push [OPTIONS] NAME[:TAG]



```
[root@lin4am1c ~]# docker image push arijeetmajumdar/nginx
The push refers to repository [docker.io/arijeetmajumdar/nginx]
4fc1aa8003a3: Mounted from library/nginx
5fb987d2e54d: Mounted from library/nginx
831c5620387f: Mounted from library/nginx
latest: digest: sha256:189cce606b29fb2a33ebc2fcecfa8e33b0b99740da4737133cdbcee92f3aba0a size: 948
[root@lin4am1c ~]#
[root@lin4am1c ~]#
```

Docker Hub View of Images Pushed



Try the two-factor authentication beta. Learn more >

arijeetmajumdar / **nginx**
Updated a minute ago
☆ 0 ⤓ 1 PUBLIC

arijeetmajumdar / **myimages**
☆ 0 ⤓ 0 PUBLIC

Why do I care how many layers I have?

More layers mean a larger image. The larger the image, the longer that it takes to both build, and push and pull from a registry.

Smaller images mean faster builds and deploys. This also means a smaller attack surface.

OK, so how can I reduce my layers?

Sharing is caring.

- Use shared base images where possible
- Limit the data written to the container layer
- Chain *RUN* statements
- Prevent cache misses at build for as long as possible

Dockerfile Tips

Anatomy of a Dockerfile

Command	Description	Example

FROM	The FROM instruction sets the Base Image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. The image can be any valid image – it is especially easy to start by pulling an image from the Public repositories	FROM ubuntu
		FROM alpine

MAINTAINER	The MAINTAINER instruction allows you to set the Author field of the generated images.	MAINTAINER johndoe
LABEL	The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and blackslashes as you would in command-line parsing.	LABEL version="1.0” LABEL vendor=“M2”
RUN	The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.	RUN apt-get install -y curl
ADD	The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the container at the path <dest>.	ADD hom* /mydir/ ADD hom?.txt /mydir/
COPY	The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.	COPY hom* /mydir/ COPY hom?.txt /mydir/
ENV	The ENV instruction sets the environment variable <key> to the value <value>. This value will be in the environment of all "descendent" Dockerfile commands and can be replaced inline in many as well.	ENV JAVA_HOME /JDK8 ENV JRE_HOME /JRE8

Anatomy of a Dockerfile

Command	Description	Example
VOLUME	The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log	VOLUME /data/webapps
USER	The USER instruction sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.	USER johndoe
WORKDIR	The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.	WORKDIR /home/user
CMD	There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect. The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.	CMD echo "This is a test." | wc -
EXPOSE	The EXPOSE instructions informs Docker that the container will listen on the specified network ports at runtime. Docker uses this information to interconnect containers using links and to determine which ports to expose to the host when using the –P flag with docker client.	EXPOSE 8080
ENTRYPOINT	An ENTRYPOINT allows you to configure a container that will run as an executable. Command line arguments to docker run <image> will be appended after all elements in an exec form ENTRYPOINT, and will override all elements specified using CMD. This allows arguments to be passed to the entry point, i.e., docker run <image> -d will pass the -d argument to the entry point. You can override the ENTRYPOINT instruction using the docker run --entrypoint flag.	ENTRYPOINT ["top", "-b"]

ADD vs. COPY

- `ADD` & `COPY` instructions both add files/directories to the container
- `ADD` can also handle URLs as a source and will automatically extract archives
- `COPY` added in Docker 1.0 and **only** copies files/dirs
- Use `COPY` unless there is a specific feature of `ADD` you absolutely need

Minimizing Image Size

Base Images

Image Name	Size
fedora:21	241 MB
ubuntu:trusty	188 MB
debian:wheezy	85 MB
alpine:3.1	5 MB
busybox:latest	2 MB

Language Images

Image Name	Size
ruby:2.2	775 MB
python:3.4	754 MB
perl:5.20	724 MB
node:0.12	706 MB
java:7-jdk	586 MB
golang:1.4	514 MB

Tip # 1 Command Chaining

- Beware of creating unnecessary layers with your Dockerfile commands

```
FROM debian:wheezy
WORKDIR /tmp
RUN wget -nv http://foo.com/someutil-v1.0.tar.gz
RUN tar -xvf someutil-v1.0.tar.gz
RUN mv /tmp/someutil-v1.0/someutil /usr/bin/someutil
RUN rm -rf /tmp/someutility-v1.0
RUN rm /tmp/someutility-v1.0.tar.gz
```

Command Chaining (contd…)

- Chaining commands allows you to clean-up before the layer is committed

```
FROM debian:wheezy
WORKDIR /tmp
RUN wget -nv http://foo.com/someutil-v1.0.tar.gz && \
    tar -xvf someutil-v1.0.tar.gz && \
    mv /tmp/someutil-v1.0/someutil /usr/bin/someutil && \
    rm -rf /tmp/someutility-v1.0 && \
    rm /tmp/someutility-v1.0.tar.gz
```

Tip #2 Clean-up After Yourself

- Try to remove any intermediate/temporary files that you don't need in your final image

```
FROM debian:wheezy
RUN apt-get update && \
    apt-get install -y curl wget git && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

Tip# 3 Remove unnecessary dependencies

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk ~~ssh vim~~
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Tip #4 Remove package manager cache

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk \
 && rm -rf /var/lib/apt/lists/*
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Leveraging the Image Cache

Image Cache

- All built or pulled layers are saved in the local image cache
- Docker won’t rebuild an unchanged layer (unless you force it to)
- Significant increase in build speed when iterating on Dockerfile
- Cache is invalidated for a layer if either the Dockerfile instruction or the parent layer is changed

Tip #5 Order matters for caching

```
FROM debian
~~COPY . /app~~
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
COPY . /app
CMD ["java", "-jar", "/app/target/app.jar"]
```

Order from least to most frequently changing content.

Tip #6 More specific COPY to limit cache busts

```
FROM debian
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
~~COPY . /app~~
COPY target/app.jar /app
CMD ["java", "-jar", "/app~~/target~~/app.jar"]
```

Only copy what's needed. Avoid "`COPY .`" if possible

Tip #7 Line buddies: apt-get update & install

```
FROM debian
~~RUN apt-get update~~
~~RUN apt-get -y install openjdk-8-jdk ssh vim~~
RUN apt-get update \
 && apt-get -y install \
    openjdk-8-jdk ssh vim
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Prevents using an outdated package cache

Build Context

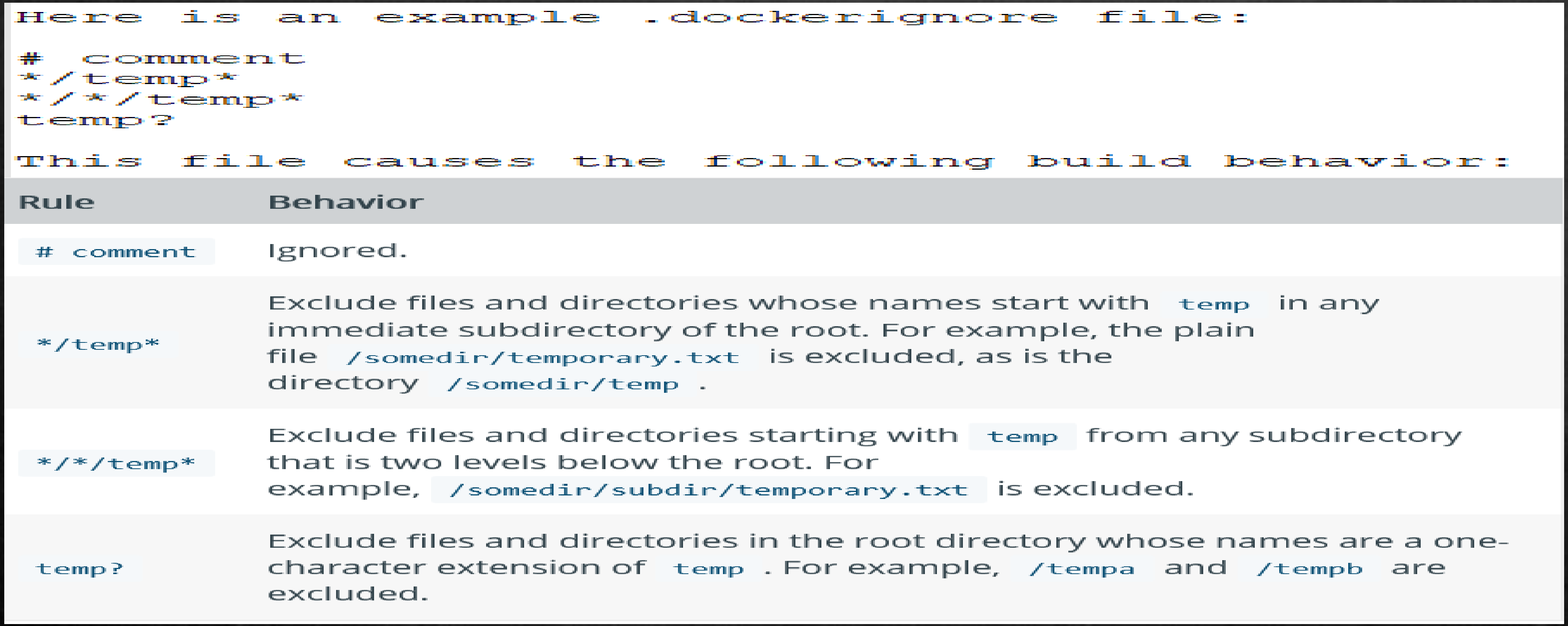
- The final argument to docker build is typically the build context
- Allows you to inject local files into your image using the `COPY` instruction
- Changes to copied files will also invalidate image cache
- Dockerfile must be located within the build context

Tip#8 Top-to-Bottom

- Place the instructions least likely to change at the top of your Dockerfile.
- Make changes/additions at the bottom.
- Place instructions you use across all of your images (`MAINTAINER`) at the top so they can be re-used across all images.

Tip#9 Using .dockerignore

- Place .dockerignore in the root of build context with list of file/directory patterns to be excluded from build context
- Very much like .gitignore
- Helpful when copying the entire build context and want to selectively ignore files.



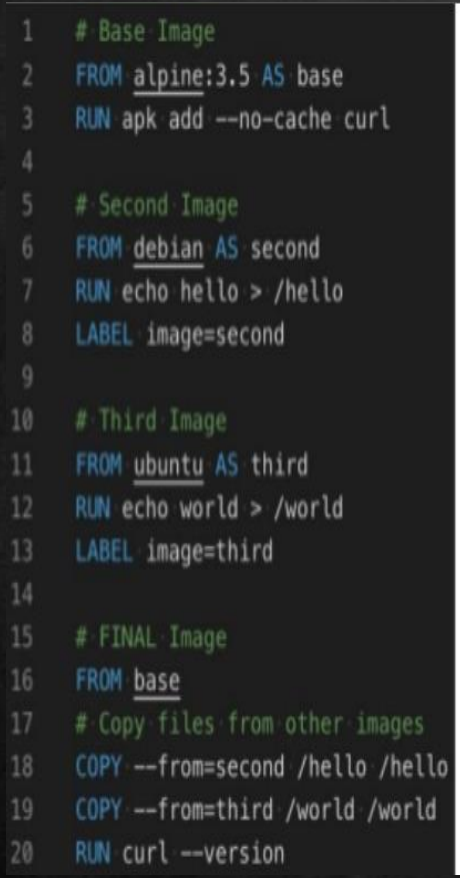
Here is an example .dockerignore file:

```
# comment
*/temp*
*/*/temp*
temp?
```

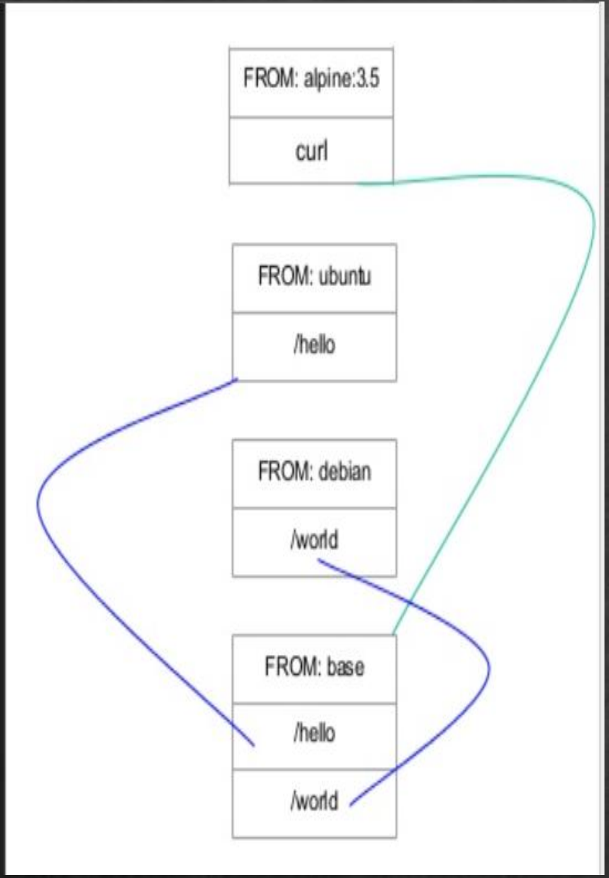
This file causes the following build behavior:

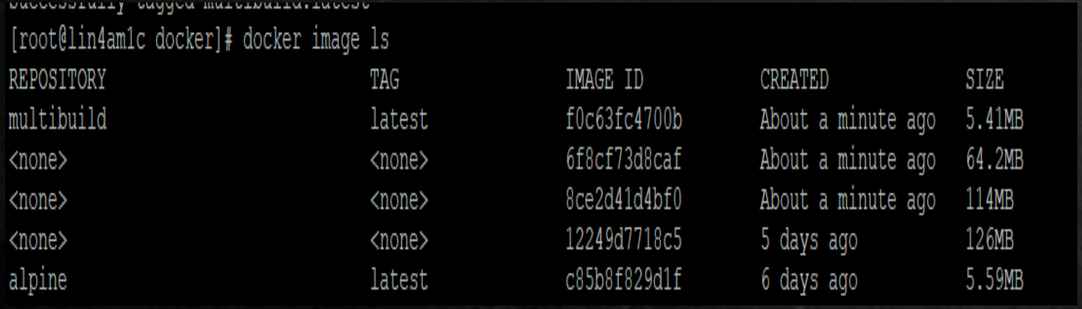
Rule	Behavior
`# comment`	Ignored.
`*/temp*`	Exclude files and directories whose names start with `temp` in any immediate subdirectory of the root. For example, the plain file `/somedir/temporary.txt` is excluded, as is the directory `/somedir/temp` .
`*/*/temp*`	Exclude files and directories starting with `temp` from any subdirectory that is two levels below the root. For example, `/somedir/subdir/temporary.txt` is excluded.
`temp?`	Exclude files and directories in the root directory whose names are a one-character extension of `temp` . For example, `/tempa` and `/tempb` are excluded.

Tip #10 Use Docker Multi Stage Build

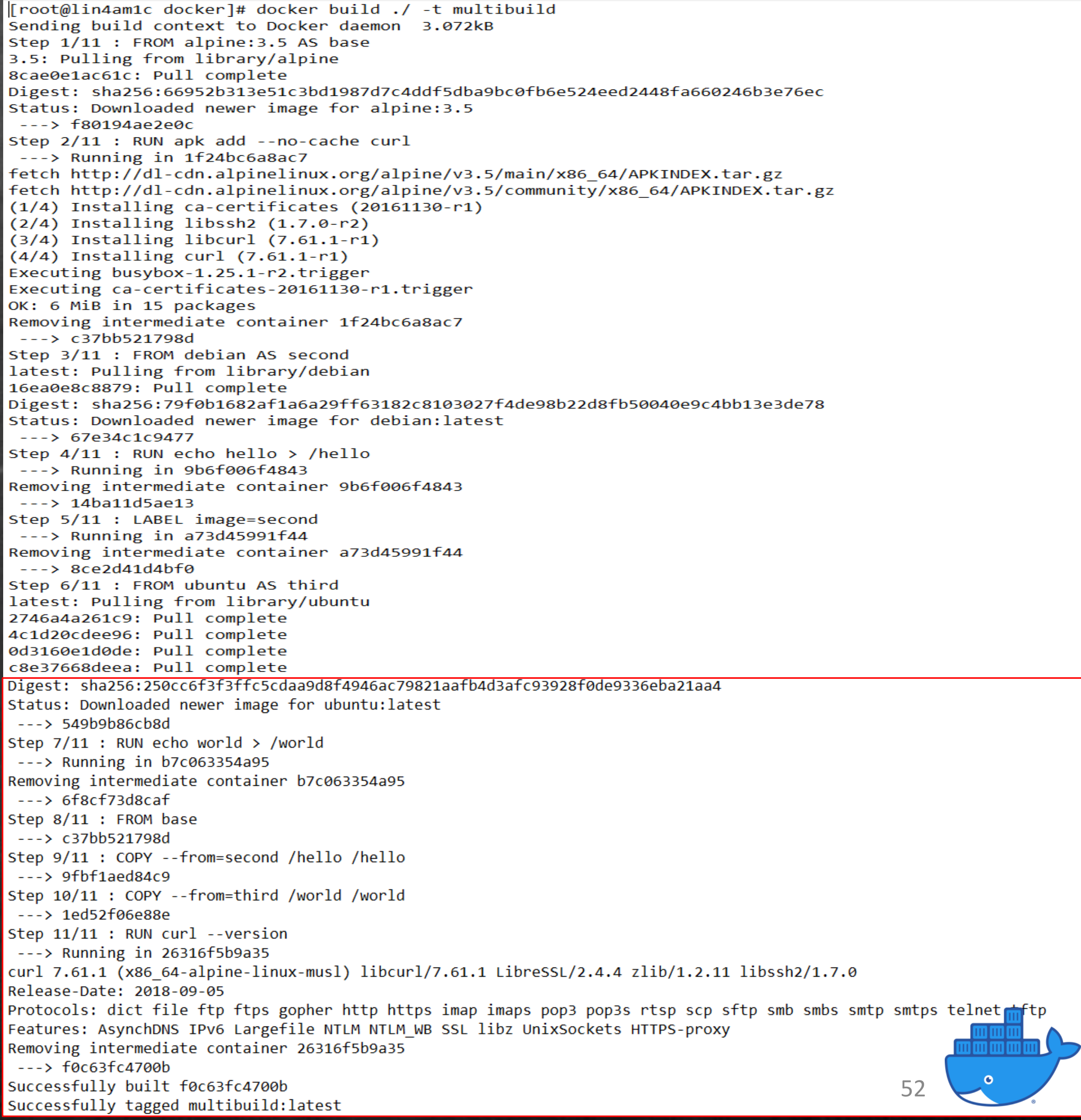


```
# Base Image
FROM alpine:3.5 AS base
RUN apk add --no-cache curl

# Second Image
FROM debian AS second
RUN echo hello > /hello
LABEL image=second

# Third Image
FROM ubuntu AS third
RUN echo world > /world
LABEL image=third

# FINAL Image
FROM base
# Copy files from other images
COPY --from=second /hello /hello
COPY --from=third /world /world
RUN curl --version
```





```
[root@lin4am1c docker]# docker image ls
REPOSITORY          TAG        IMAGE ID       CREATED            SIZE
multibuild          latest     f0c63fc4700b   About a minute ago 5.41MB
<none>              <none>     6f8cf73d8caf   About a minute ago 64.2MB
<none>              <none>     8ce2d41d4bf0   About a minute ago 114MB
<none>              <none>     12249d7718c5   5 days ago         126MB
alpine              latest     c85b8f829d1f   6 days ago         5.59MB
```



```
[root@lin4am1c docker]# docker build ./ -t multibuild
Sending build context to Docker daemon  3.072kB
Step 1/11 : FROM alpine:3.5 AS base
3.5: Pulling from library/alpine
8cae0e1ac61c: Pull complete
Digest: sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec
Status: Downloaded newer image for alpine:3.5
 ---> f80194ae2e0c
Step 2/11 : RUN apk add --no-cache curl
 ---> Running in 1f24bc6a8ac7
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/community/x86_64/APKINDEX.tar.gz
(1/4) Installing ca-certificates (20161130-r1)
(2/4) Installing libssh2 (1.7.0-r2)
(3/4) Installing libcurl (7.61.1-r1)
(4/4) Installing curl (7.61.1-r1)
Executing busybox-1.25.1-r2.trigger
Executing ca-certificates-20161130-r1.trigger
OK: 6 MiB in 15 packages
Removing intermediate container 1f24bc6a8ac7
 ---> c37bb521798d
Step 3/11 : FROM debian AS second
latest: Pulling from library/debian
16ea0e8c8879: Pull complete
Digest: sha256:79f0b1682af1a6a29ff63182c8103027f4de98b22d8fb50040e9c4bb13e3de78
Status: Downloaded newer image for debian:latest
 ---> 67e34c1c9477
Step 4/11 : RUN echo hello > /hello
 ---> Running in 9b6f006f4843
Removing intermediate container 9b6f006f4843
 ---> 14ba11d5ae13
Step 5/11 : LABEL image=second
 ---> Running in a73d45991f44
Removing intermediate container a73d45991f44
 ---> 8ce2d41d4bf0
Step 6/11 : FROM ubuntu AS third
latest: Pulling from library/ubuntu
2746a4a261c9: Pull complete
4c1d20cdee96: Pull complete
0d3160e1d0de: Pull complete
c8e37668deea: Pull complete
Digest: sha256:250cc6f3f3ffc5cdaa9d8f4946ac79821aafb4d3afc93928f0de9336eba21aa4
Status: Downloaded newer image for ubuntu:latest
 ---> 549b9b86cb8d
Step 7/11 : RUN echo world > /world
 ---> Running in b7c063354a95
Removing intermediate container b7c063354a95
 ---> 6f8cf73d8caf
Step 8/11 : FROM base
 ---> c37bb521798d
Step 9/11 : COPY --from=second /hello /hello
 ---> 9fbf1aed84c9
Step 10/11 : COPY --from=third /world /world
 ---> 1ed52f06e88e
Step 11/11 : RUN curl --version
 ---> Running in 26316f5b9a35
curl 7.61.1 (x86_64-alpine-linux-musl) libcurl/7.61.1 LibreSSL/2.4.4 zlib/1.2.11 libssh2/1.7.0
Release-Date: 2018-09-05
Protocols: dict file ftp ftps gopher http https imap imaps pop3 pop3s rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile NTLM NTLM_WB SSL libz UnixSockets HTTPS-proxy
Removing intermediate container 26316f5b9a35
 ---> f0c63fc4700b
Successfully built f0c63fc4700b
Successfully tagged multibuild:latest
```

Repeatable Image Builds

- Ideally, anyone should be able to build your Dockerfile at any time and get the same result
- Vague dependencies can result in unpredictable builds

```
RUN apt-get update && apt-get install -y hello
```

vs

```
RUN apt-get update && apt-get install -y hello=2.8-4
```

Thank You all .. Happy Dockering