Docker Basics

Microservices

- Microservices a variant of service-oriented architecture(SOA), arranges an application as a collection of loosely coupled isolated services.
- Microservices is a unique method of software development that focuses on developing isolated individual services which combines as a single application.
- Microservices help to build an applications as combination of multiple small services, each running in its own environment and are independently deployable.
- These services can be written in different programming languages and also have its own storage and databases.
- Changes done to one service will not have any downtime on other services, which also enables developers to release there changes without any dependency on other services.

Microservices Advantages

Below are some of the advantages of using Microservices:

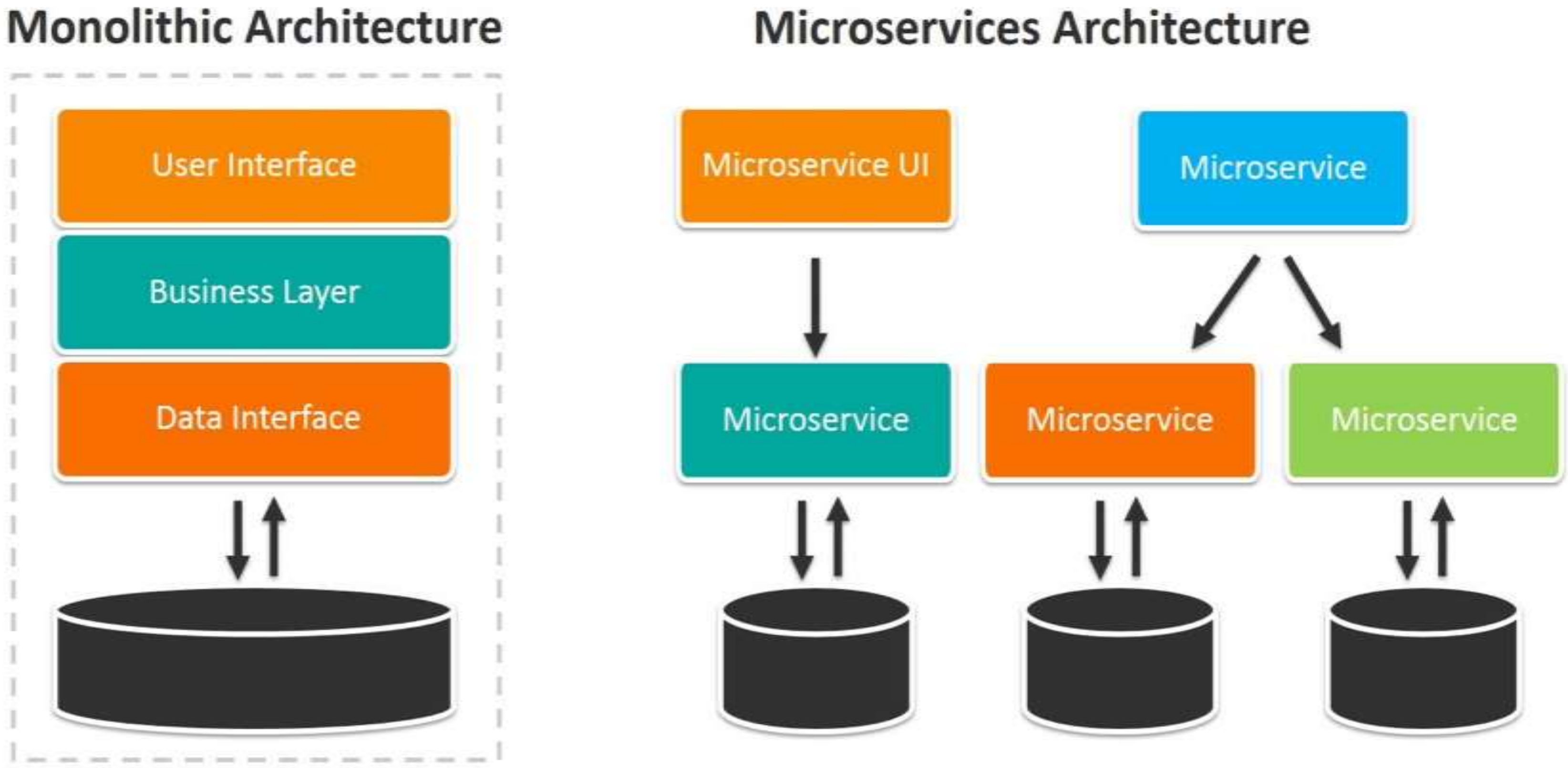
- It gives developers freedom to develop and deploy services independently.
- These independent services can be developed using different programming languages.
- When new changes are required, it only needs to be done to specific services only instead of bring down whole application.
- It provides better fault tolerance, where one service going on does not have much impact on whole application.

Microservices Disadvantages

Below are some of the disadvantages of using Microservices:

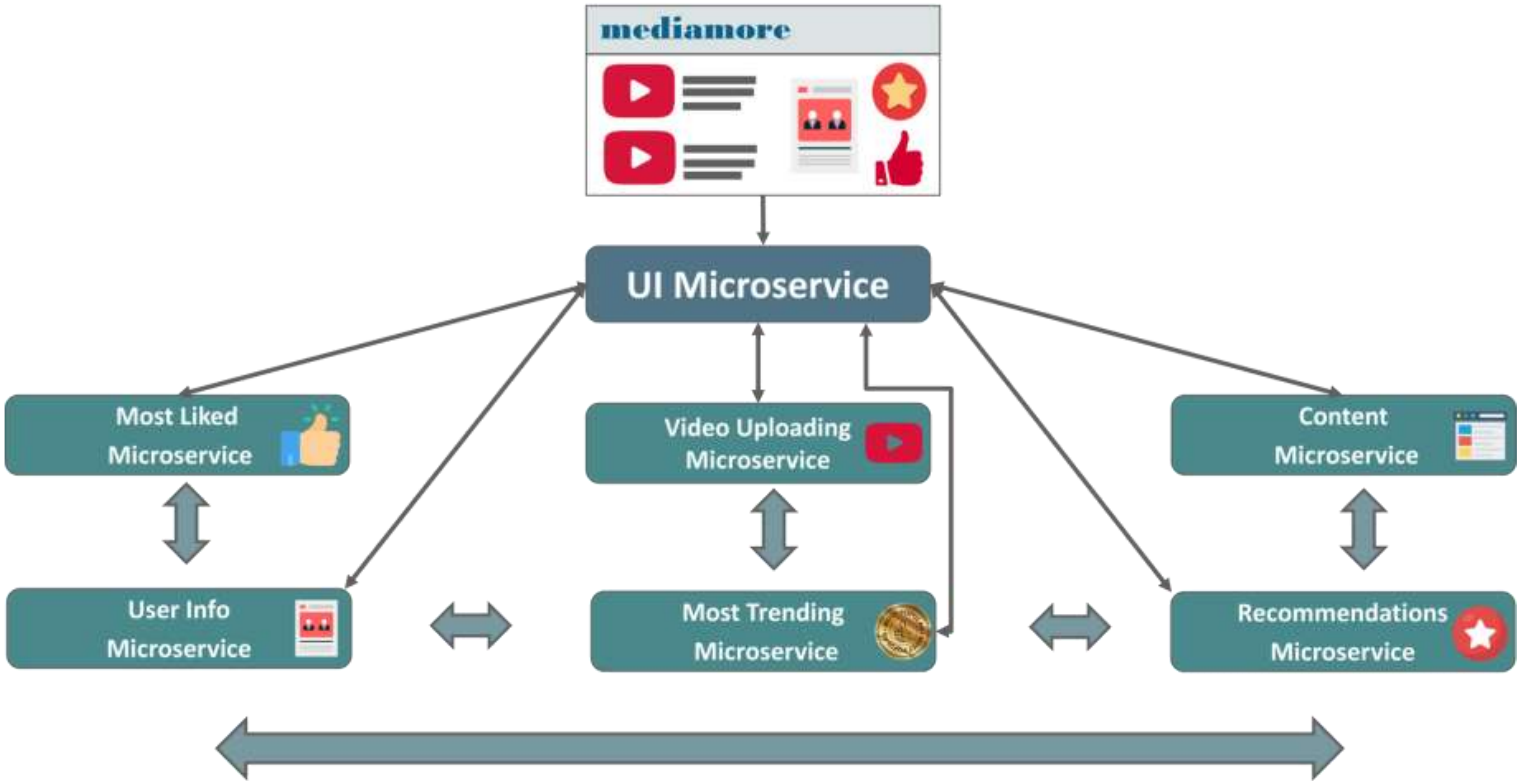
- Due to multiple services, it becomes sometimes complicated to perform end to end testing.
- Developers need to put additional efforts to establish connectivity between multiple services.
- With multiple services utilization of resources would also increase.
- Debugging errors for services can be harder since each service has its own service logs.

Microservices Architecture

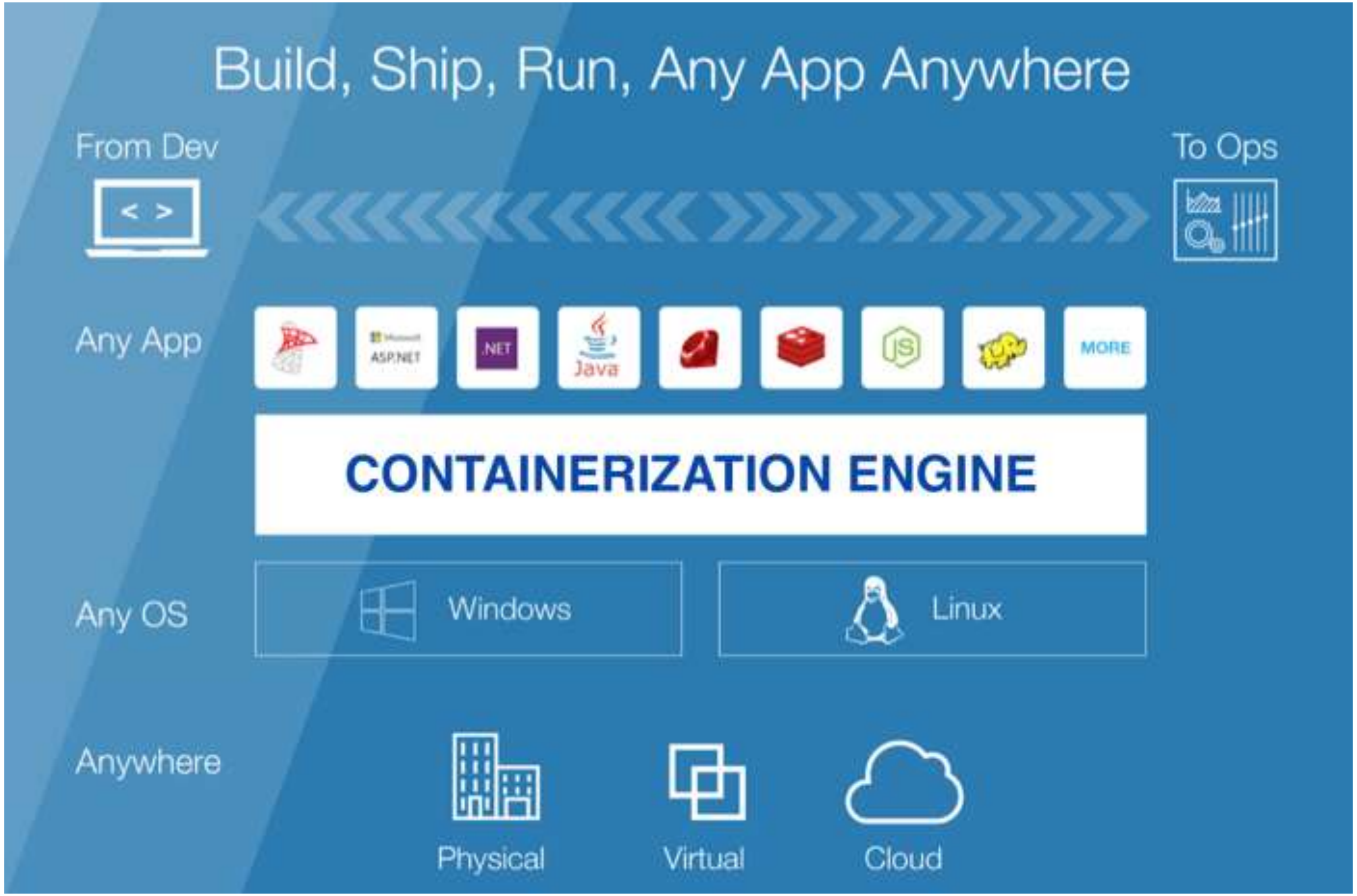


Source: Rancher

Microservices Architecture



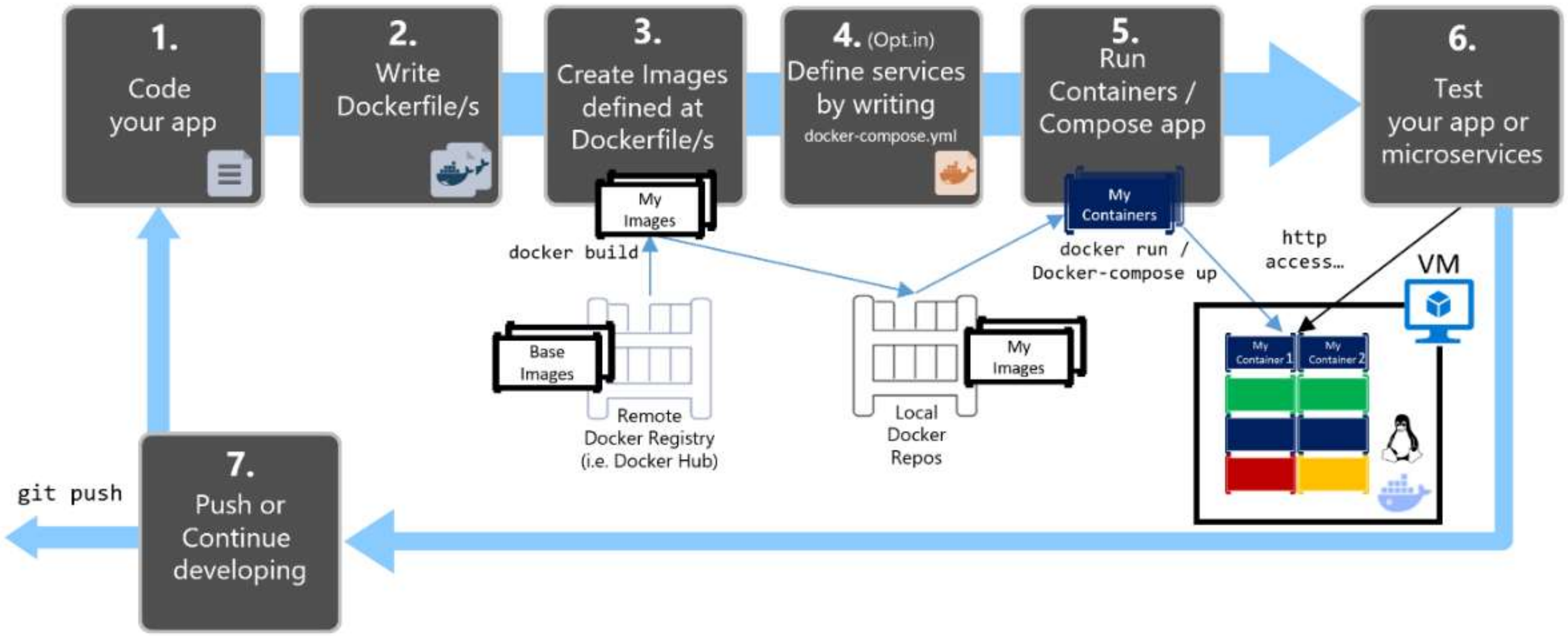
Containerization



Source: HackerNoon

Containerization Pipeline

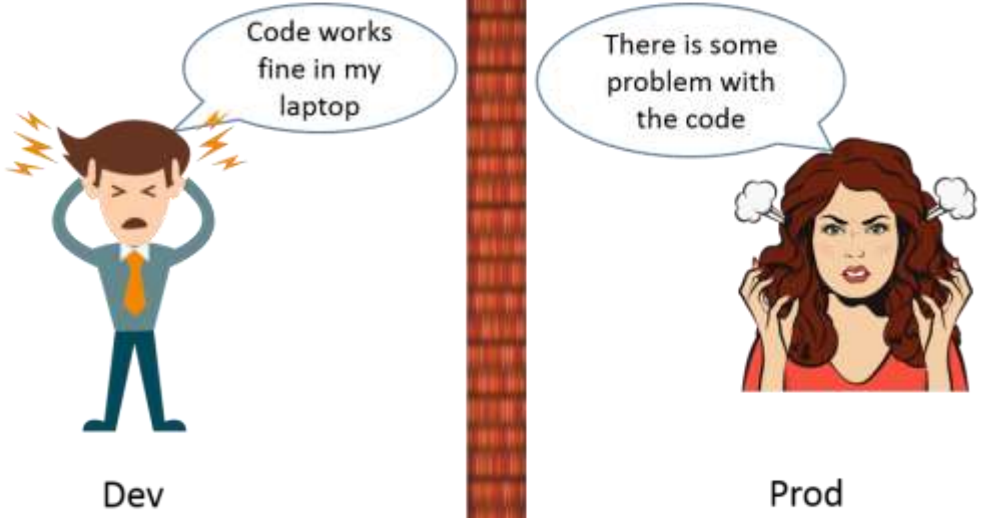
Inner-Loop development workflow for Docker apps



1. Code your app
2. Write Dockerfile/s
3. Create Images defined at Dockerfile/s
4. (Opt.in) Define services by writing docker-compose.yml
5. Run Containers / Compose app
6. Test your app or microservices
7. Push or Continue developing
My Images
docker build
Base Images
Remote Docker Registry (i.e. Docker Hub)
Local Docker Repos
My Images
My Containers
docker run / Docker-compose up
http access...
VM
My Container 1
My Container 2
git push

Need of Docker - Versioning Problem

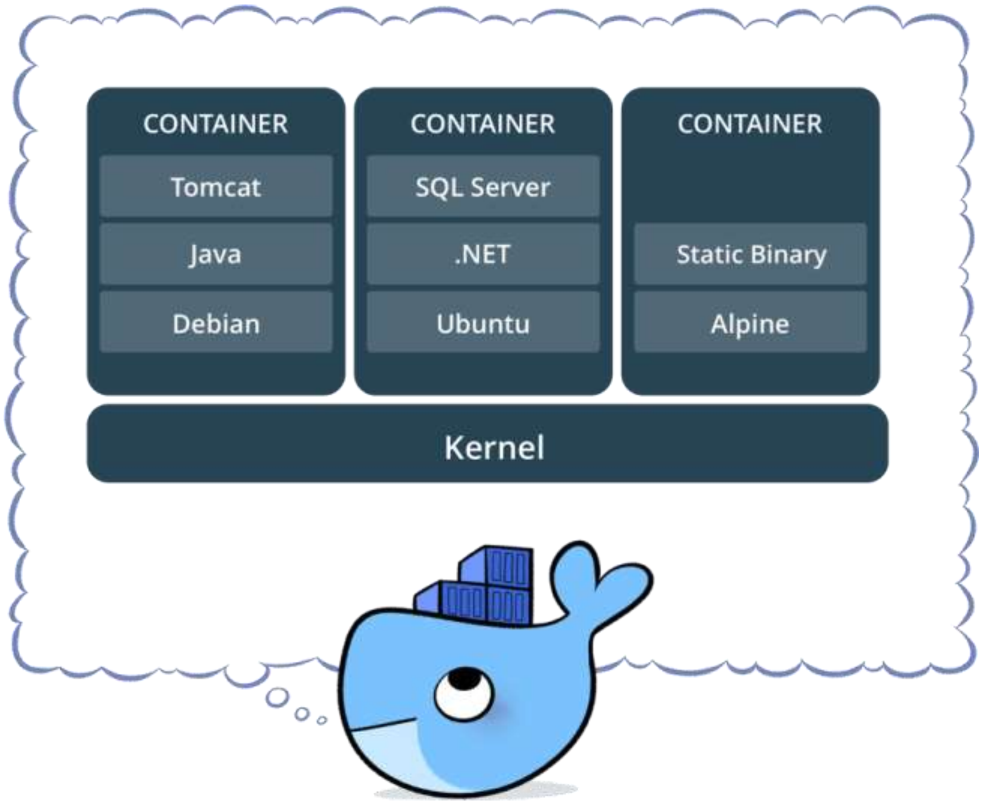
In Dev there can be a software that is upgraded and in Prod the old version of that software might be present


Code works fine in my laptop
There is some problem with the code
Dev
Prod

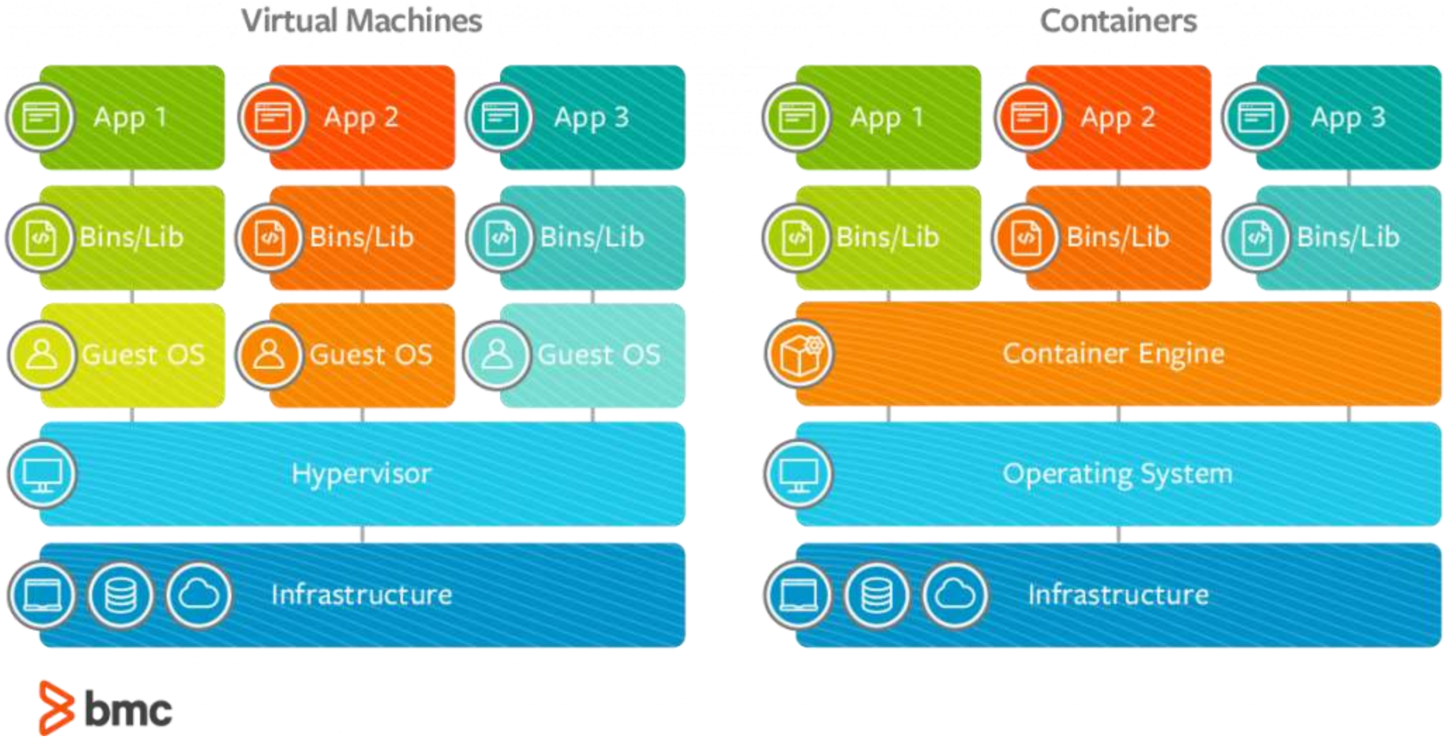
Source: Quora

Need of Docker - Versioning Problem

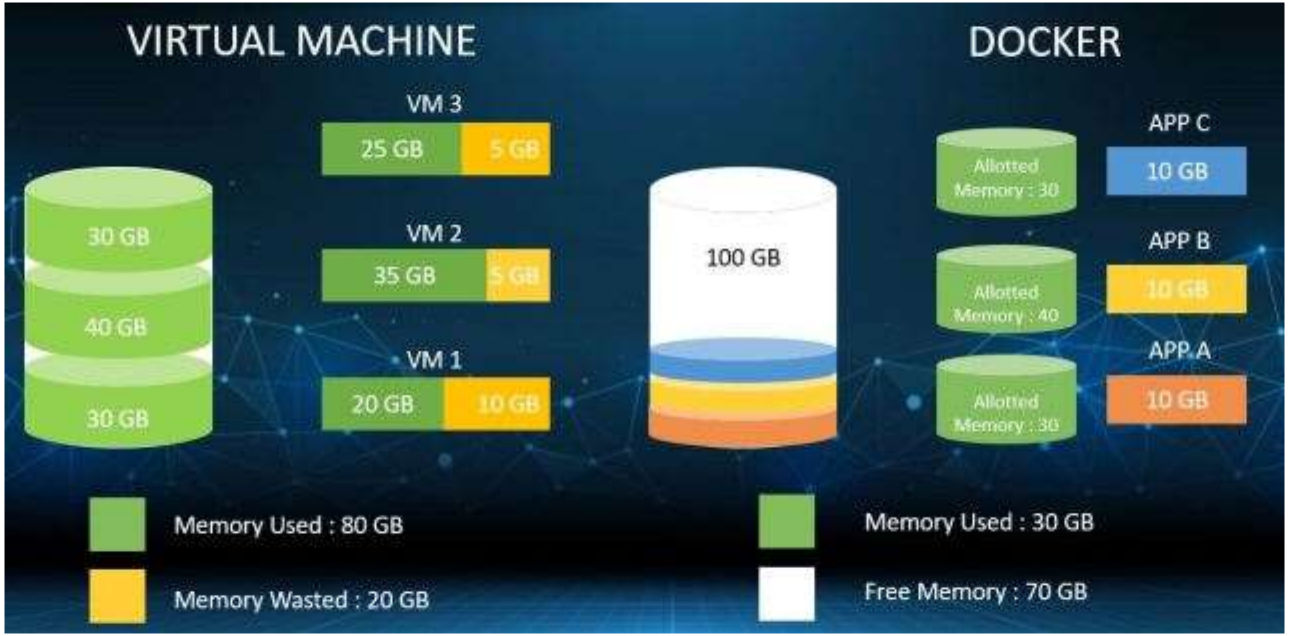
Why to use Docker?



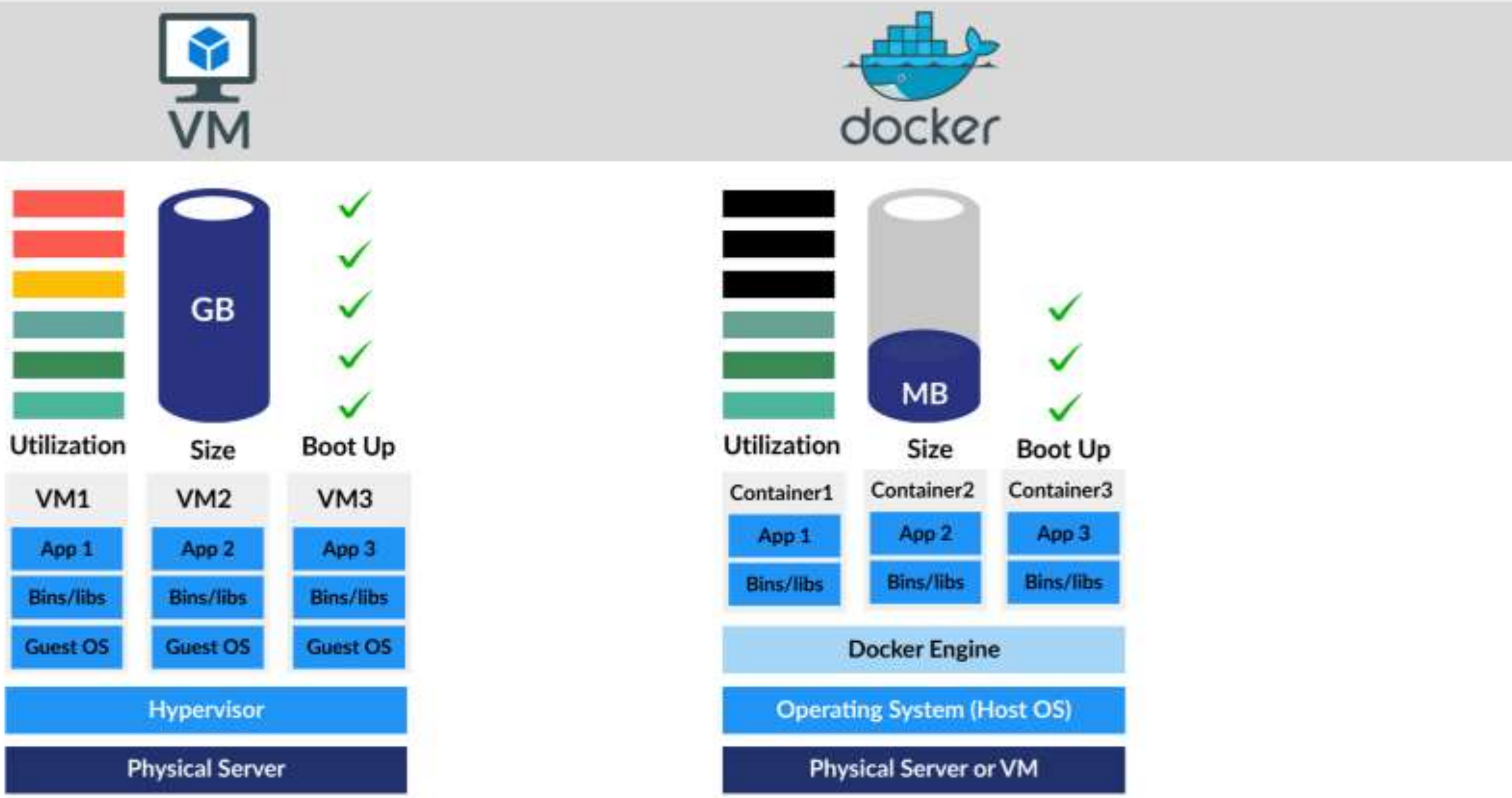
Why to use Docker?



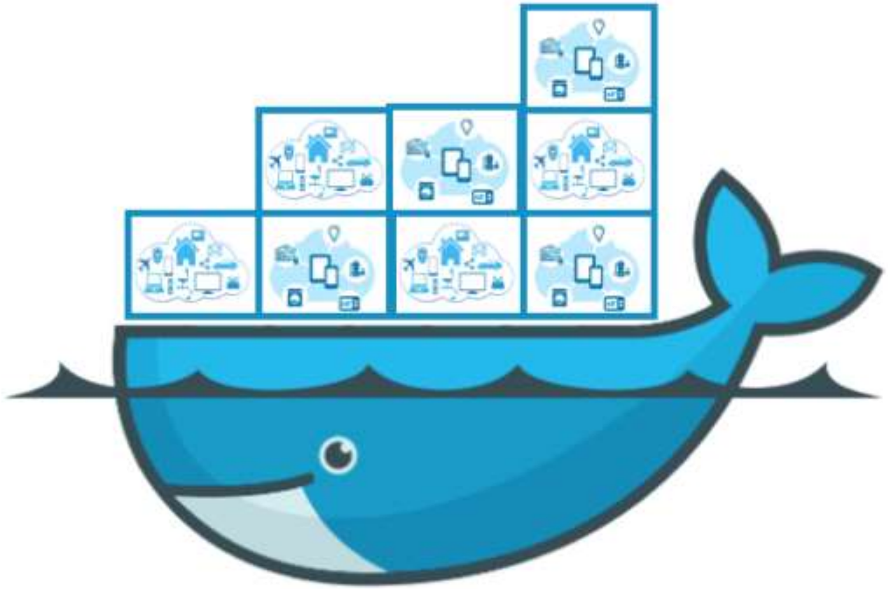
Why to use Docker?



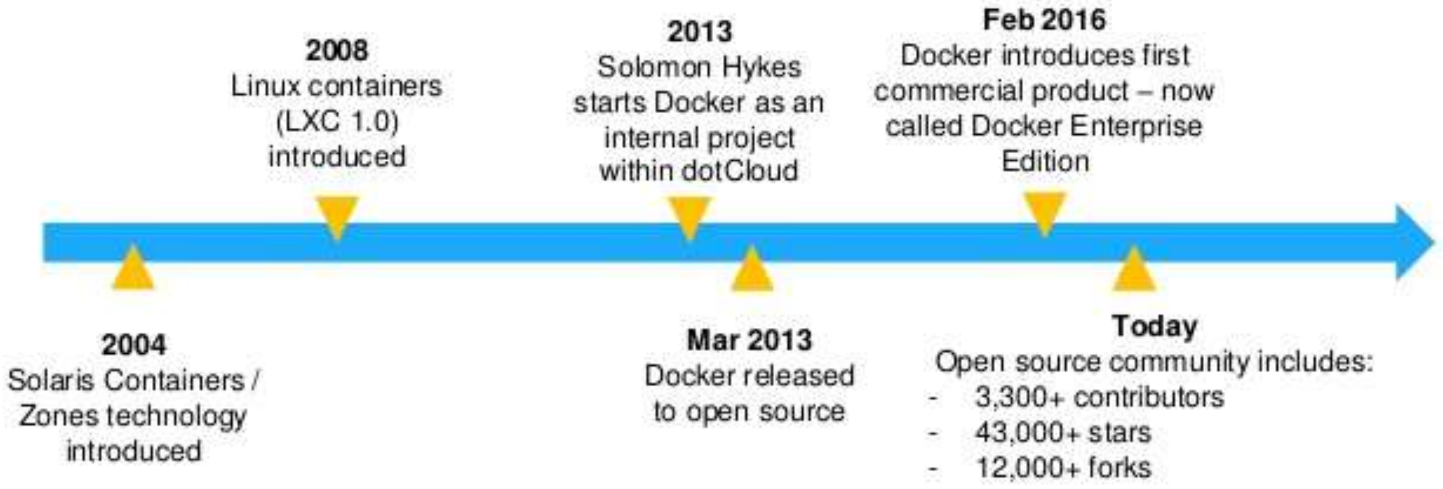
Why to use Docker?



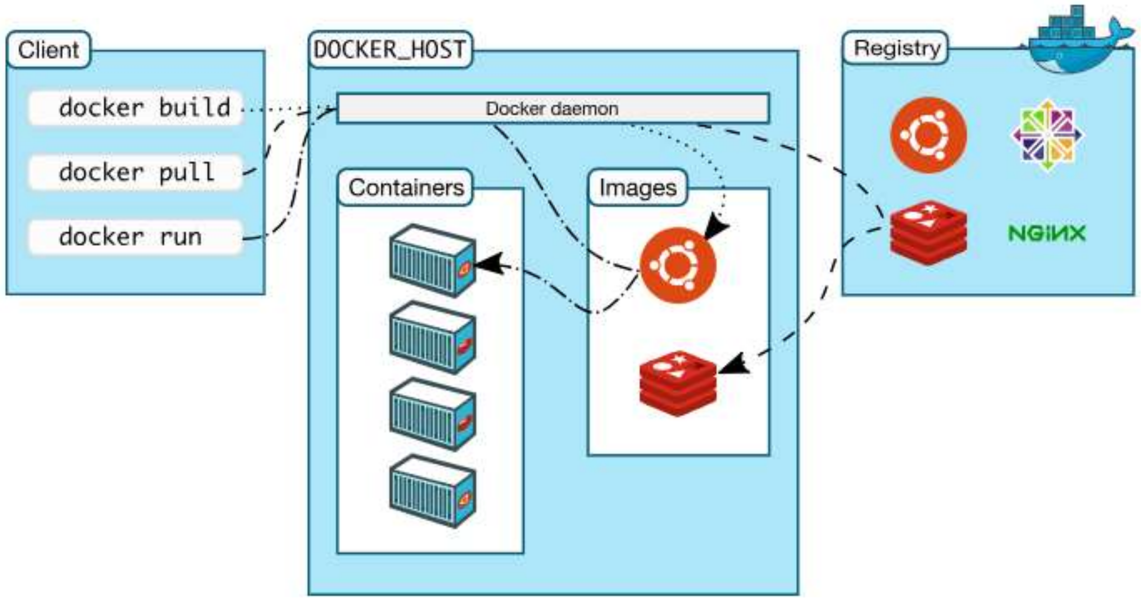
What is Docker ?



Docker History


2008
Linux containers (LXC 1.0) introduced
2013
Solomon Hykes starts Docker as an internal project within dotCloud
Feb 2016
Docker introduces first commercial product – now called Docker Enterprise Edition
2004
Solaris Containers / Zones technology introduced
Mar 2013
Docker released to open source
Today
Open source community includes:
- 3,300+ contributors
- 43,000+ stars
- 12,000+ forks

Docker Architecture



Source: Docker

Docker Features

Below are some of benefits which we get when we implement docker

01 Easy and Faster Configuration

02 Increase productivity

03 Application Isolation

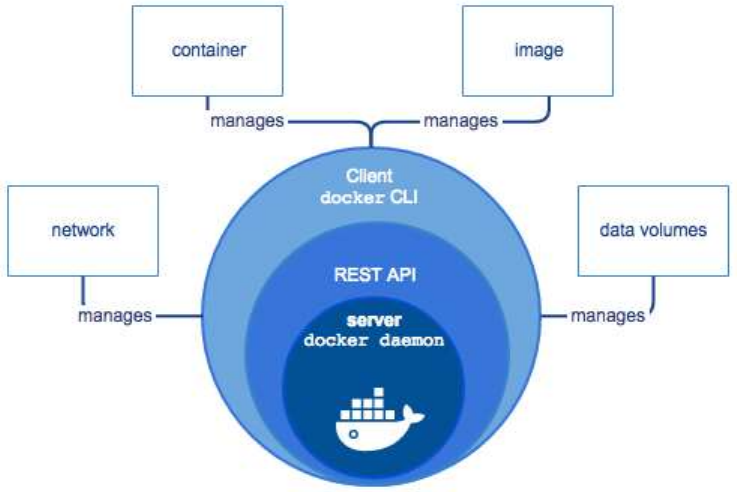
04 Routing Mesh

05 Services

Docker Components

Below are some of components of docker architecture:

- **The Docker daemon**
- **The Docker Client**
- **Docker Registries**
- **Docker Images**
- **Docker Containers**

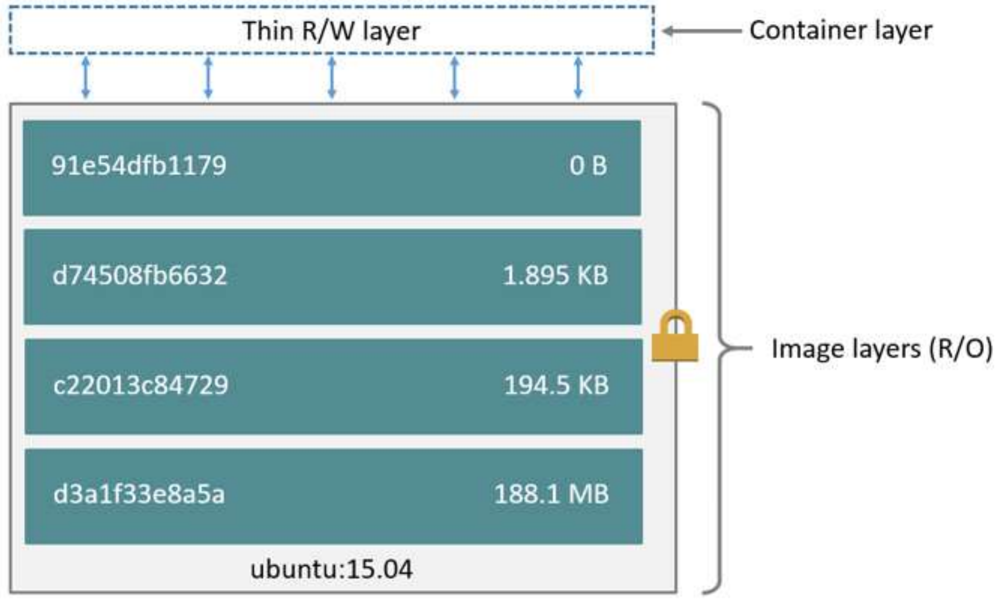


Source: Docker

Docker Image

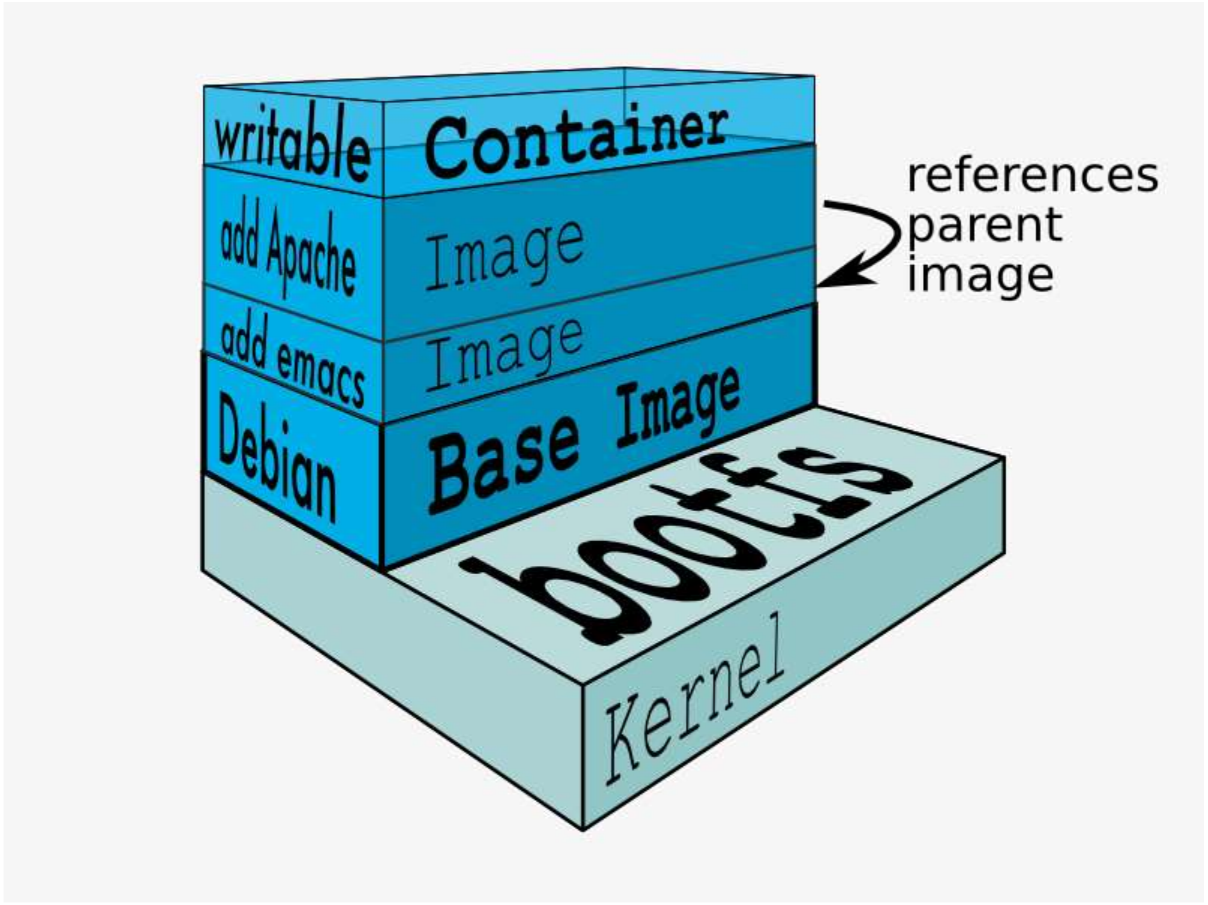
- Docker image is a read-only template with instructions for creating a Docker container.
- An image is a combination of a file system, binaries and network configurations.
- An image is an inert, immutable, file that's essentially a snapshot of a container.
- We can use ready made images or custom images to create docker containers using docker run command.
- Docker images can be stored on Docker Hub, ECR or any artifactory docker repository.

Docker Image



Container
(based on ubuntu:15.04 image)

Docker Custom Image

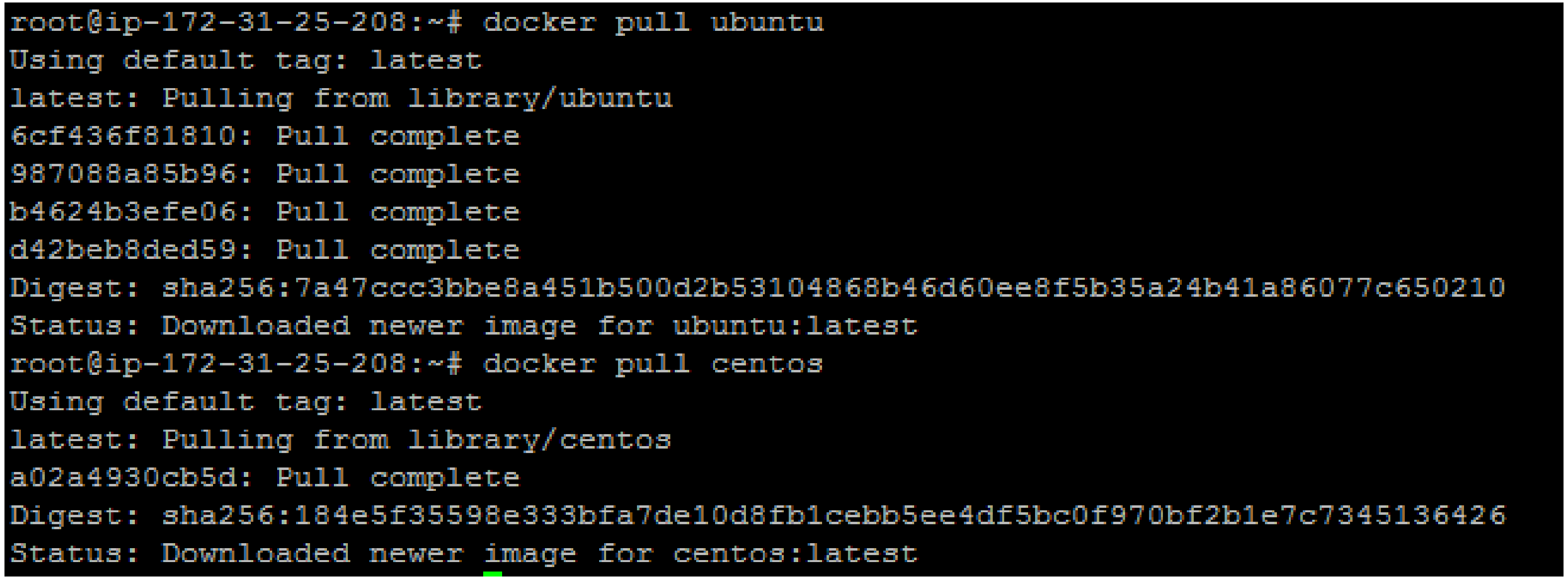


Docker Image Commands

Docker Engine uses below mentioned commands to manage docker images on server:

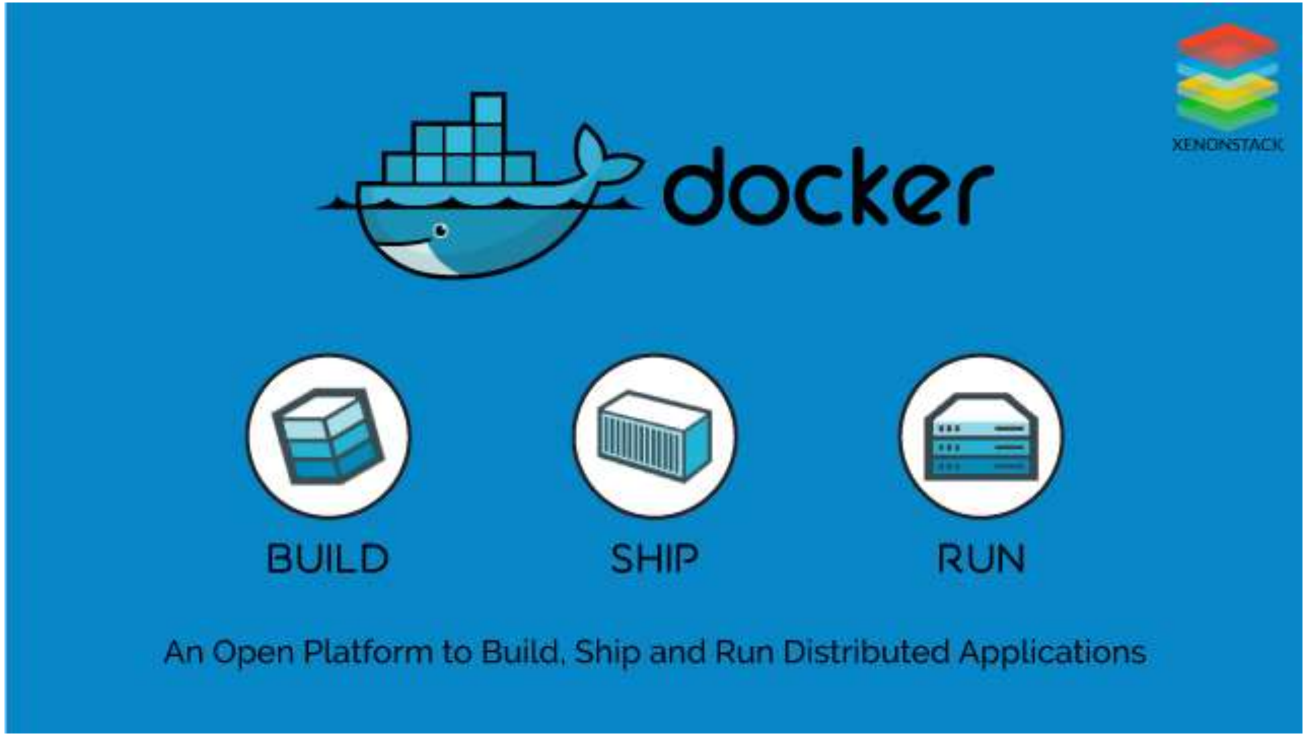
- docker images
- docker run
- docker tag
- docker pull
- docker push
- docker commit
- docker rmi

Docker Base Image



```
root@ip-172-31-25-208:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6cf436f81810: Pull complete
987088a85b96: Pull complete
b4624b3efe06: Pull complete
d42beb8ded59: Pull complete
Digest: sha256:7a47ccc3bbe8a451b500d2b53104868b46d60ee8f5b35a24b41a86077c650210
Status: Downloaded newer image for ubuntu:latest
root@ip-172-31-25-208:~# docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a02a4930cb5d: Pull complete
Digest: sha256:184e5f35598e333bfa7de10d8fb1cebb5ee4df5bc0f970bf2b1e7c7345136426
Status: Downloaded newer image for centos:latest
```

Docker Containers

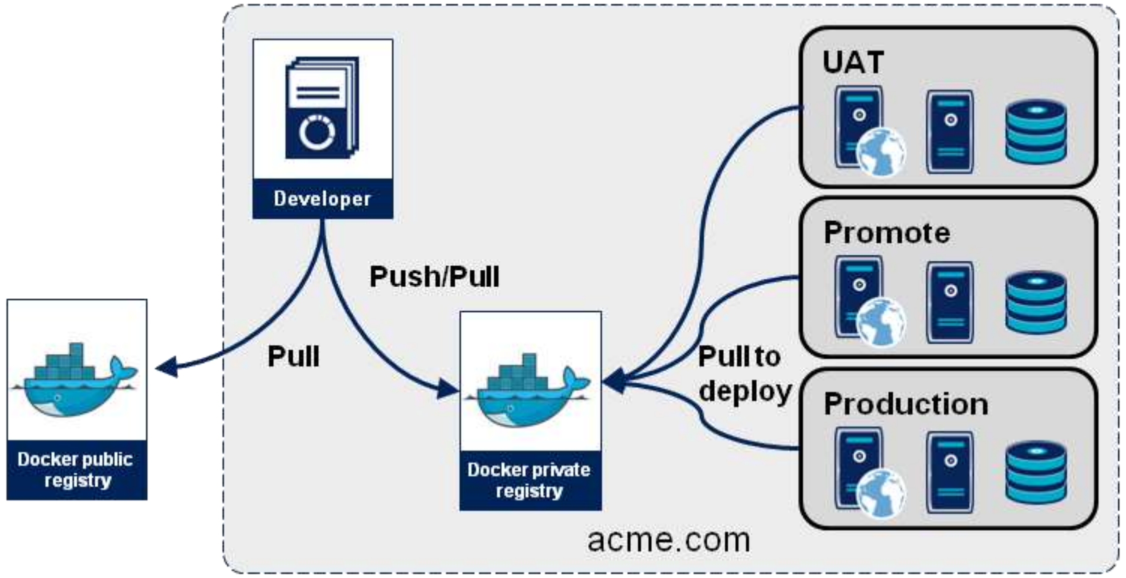

XENONSTACK
docker
BUILD
SHIP
RUN
An Open Platform to Build, Ship and Run Distributed Applications

Source: Docker

Docker Registry

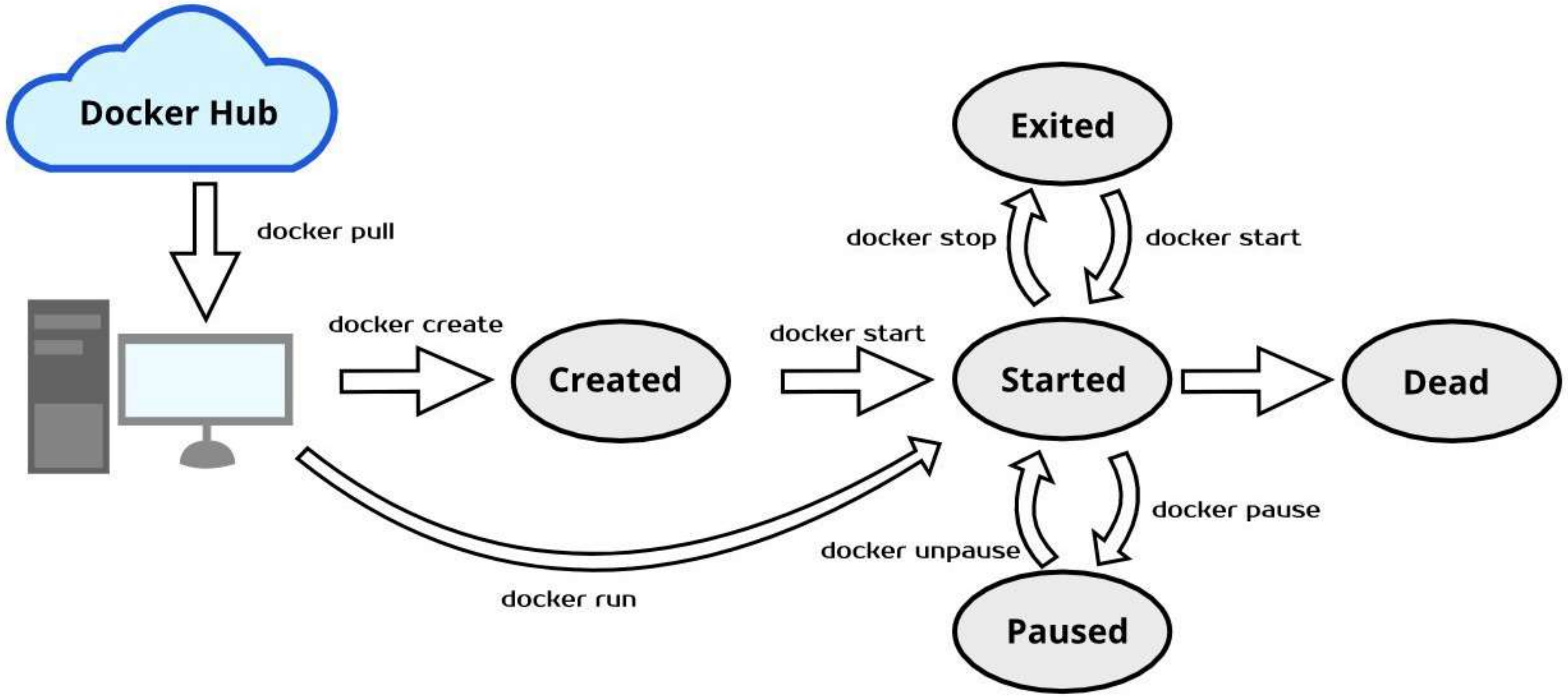
- Docker registry is an application which stores and deliver docker images.
- As part of CI CD pipelines once docker images are built up, we can store them on Docker registry.
- We have various application providing Docker registries such as docker hub, ECR, Jfrog artifactory etc.
- By default we can use free docker registry called Docker hub, where all application based docker images are available to download.

Docker Registry



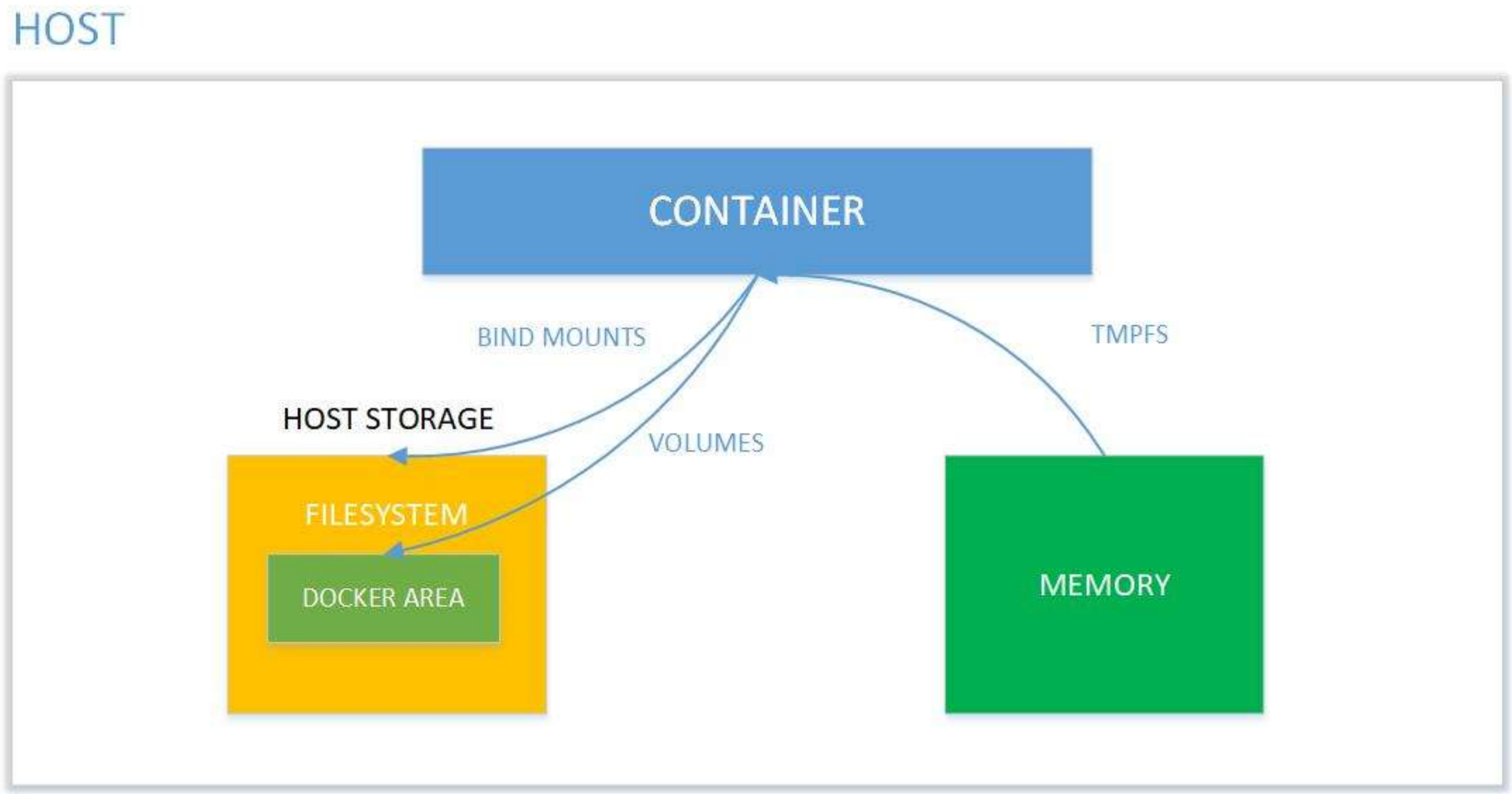
Source: octo

Docker Container Lifecycle



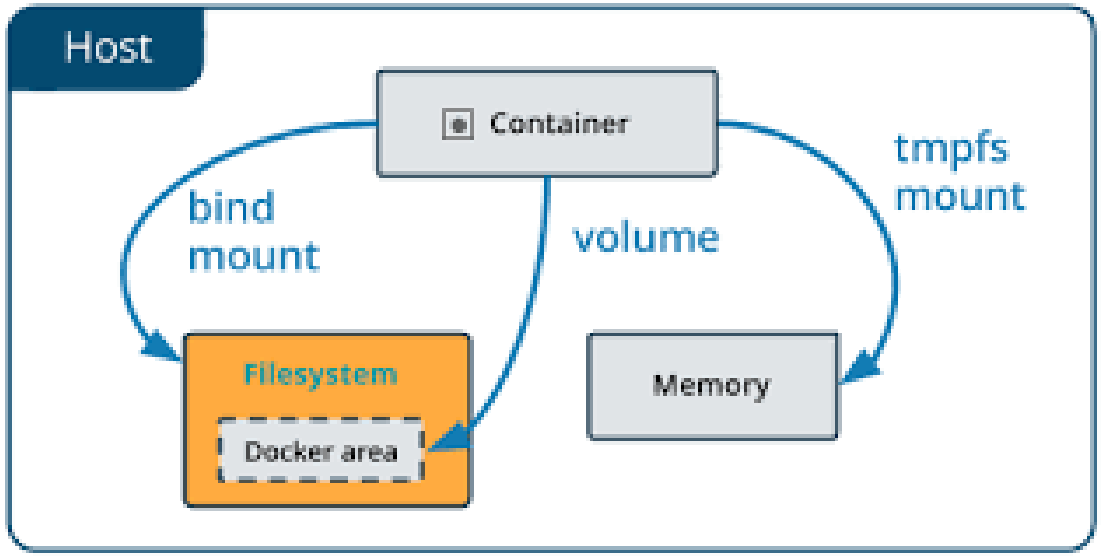
Source: Medium

Filesystem in Containers

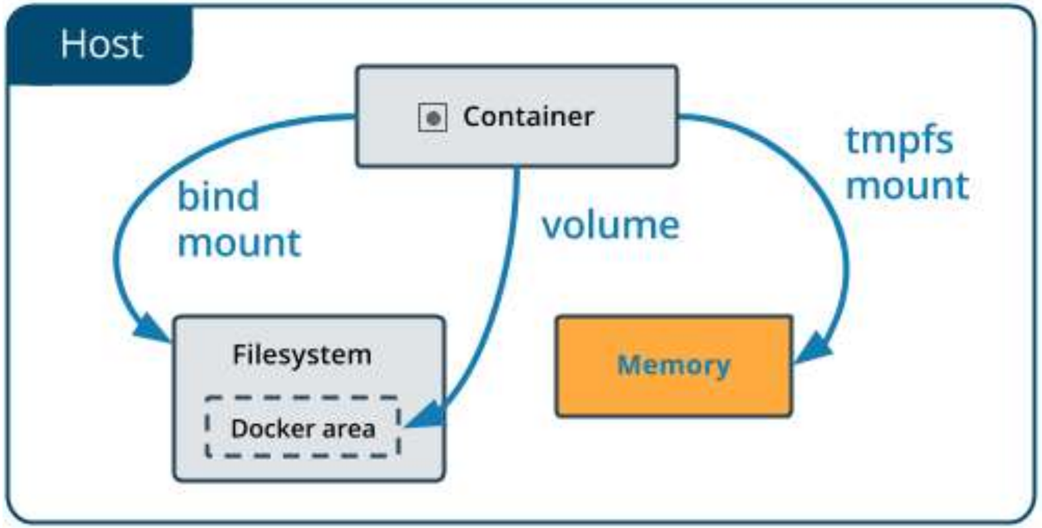


Source: Binary Maps

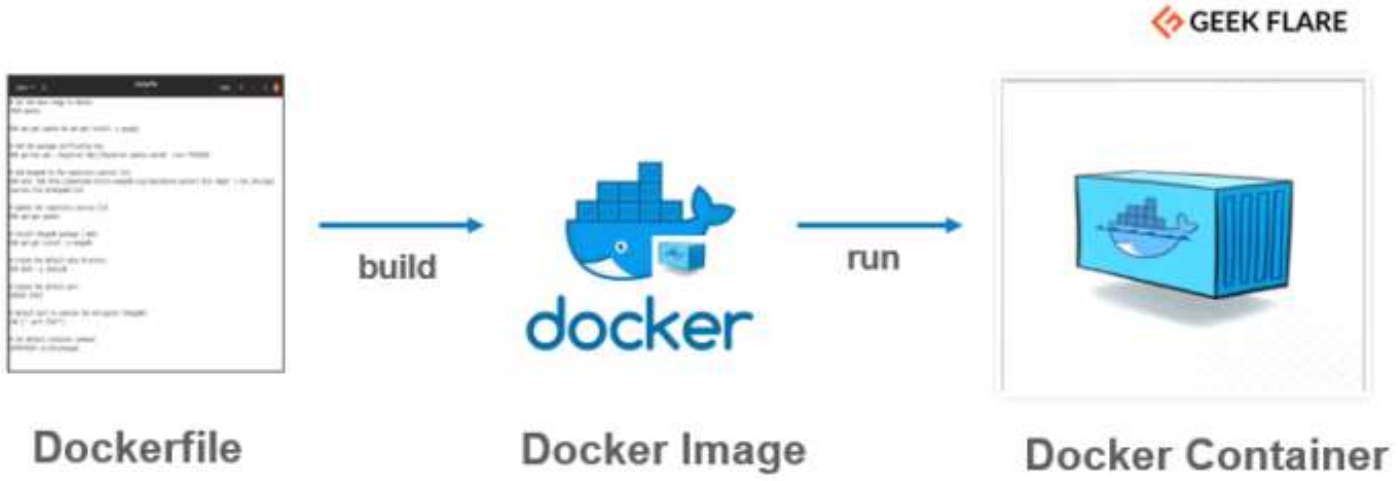
Bind Mounts Docker



Tmpfs Docker



Dockerfile


GEEK FLARE
build
docker
run
Dockerfile
Docker Image
Docker Container

Creating Dockerfile

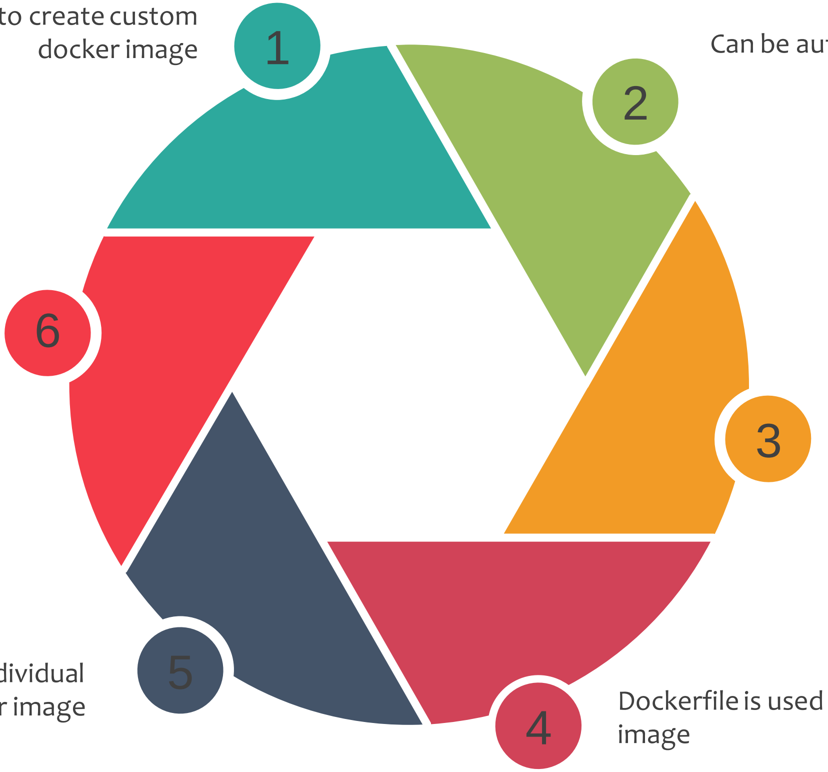
- In order to create Dockerfile, we need to create a new working directory where we will run docker build.
- Follow below commands to create Dockerfile for docker image build.

```
mkdir dockerbuild

cd dockerbuild
```

- Now we will create Dockerfile in build directory:

```
nano Dockerfile
```

Dockerfile Benefits



1. Automated Build process to create custom docker image
2. Can be automated using CI/CD pipelines
3. It provides step by step procedure for image build
4. Dockerfile is used as build script for docker image
5. Each step in Dockerfile creates individual layers in docker image
6. Dockerfile can be stored in git repository with source code

Dockerfile Directives

- Dockerfile is a build script that contains step by step instructions to build docker image. In order to create Dockerfile we can use below Dockerfile directives.

FROM	RUN
CMD	LABEL
EXPOSE	ENV
ADD	COPY
ENTRYPOINT	VOLUME
USER	WORKDIR

Dockerfile Directives: USER and RUN

- USER directive is used to specify a specific user while running RUN, CMD and ENTRYPOINT instructions using Dockerfile.

```
USER <username>

USER jenkins
```

- RUN directive is used to execute commands specified in a new layer on top of base docker image.

```
RUN <shell command>

RUN apt -y install openjdk-8-jdk
```

Dockerfile Directives: RUN Order of Execution

- We can use Dockerfile RUN directive to execute build steps during docker image build process.
- Order of commands execution is also important in order to perform proper configurations.
- We can run software packages installation, custom shell script and any random commands in RUN directive.
- This is a generic directive which can be customized according to developers requirements.

Example:

```
RUN apt install -y <package_name>

RUN <command execution>
```

Dockerfile Directives: ENV

- ENV directive is used to specify environment variables while running Dockerfile.
- We can specify environment specific values in ENV directive which will be stored as a inseparable layer in docker image.
- Once environment variable is configured with this directive we can not unset it even if run a unset command in RUN directive after ENV directive.
- These are key pair values which can be configured in Dockerfile for creating environment variables.

  Example: ENV variable_name value

Dockerfile Directives: CMD v/s RUN

- RUN directive is used to execute commands in docker image but CMD directive is used to specify command for running applications inside docker containers.
- RUN directive can be executed multiple times in Dockerfile whereas CMD can only be configured once.
- RUN directive is executed during docker image build process whereas CMD is executed only when container stats.

Example: CMD <startup_script>

RUN <configuration command>

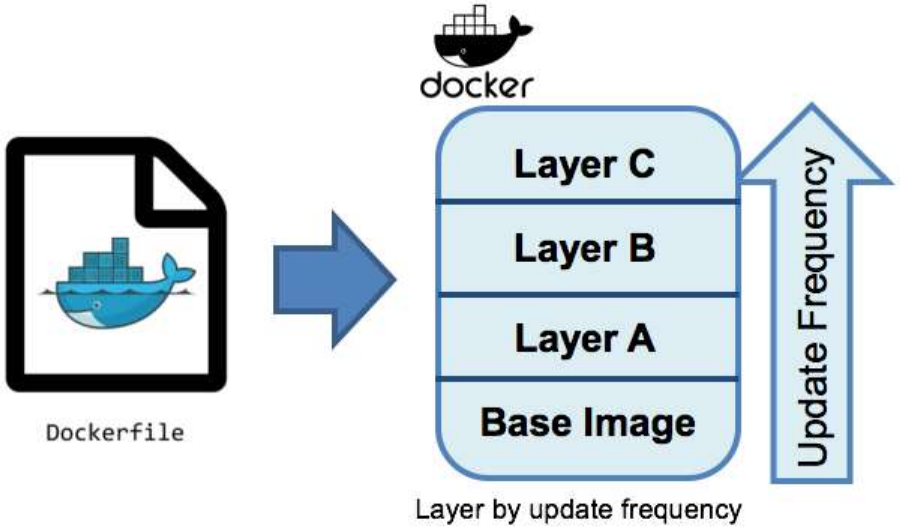
Dockerfile Directives: ENTRYPOINT

- ENTRYPOINT directive is used to configure container as a executable using which we can run any command inside docker container.
- It helps us to run commands with parameters which can be overwritten during run time.
- With ENTRYPOINT, the command and parameters are not ignored during run time.

ENTRYPOINT [“command", "parameter"]

Example: docker run -it --entrypoint <command>

Docker build



Publishing to Docker Hub

- Docker hub is default repository interface integrated with Docker installation which is used to share images between different users.
- Using Docker hub we can build Docker image on one system and deploy it on another system.
- All base OS images and custom software's images are available in Docker hub and we can easily pull those on any server.
- To push a repository to the Docker Hub, you must name your local image using your Docker Hub username, and the repository name that you created through Docker Hub on the web.
- We can use set of commands involved in pushing and pulling Docker images:

  docker push: To push Docker image to Docker hub

  docker pull: To pull Docker image from Docker hub

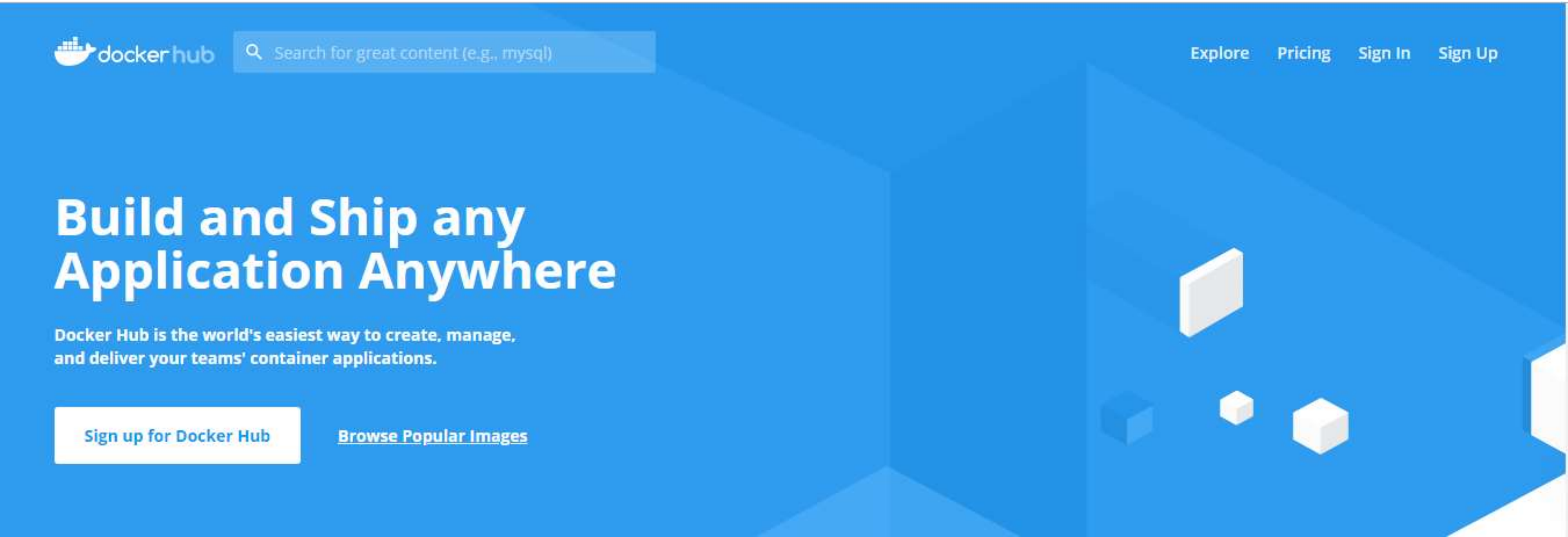
  docker tag: To create a different tag of an existing image for pushing to Docker hub

Publishing to Docker Hub

- Docker hub account can be easily created using below link.

https://hub.docker.com/

Once account is created login to Docker hub and create your private/public repository.

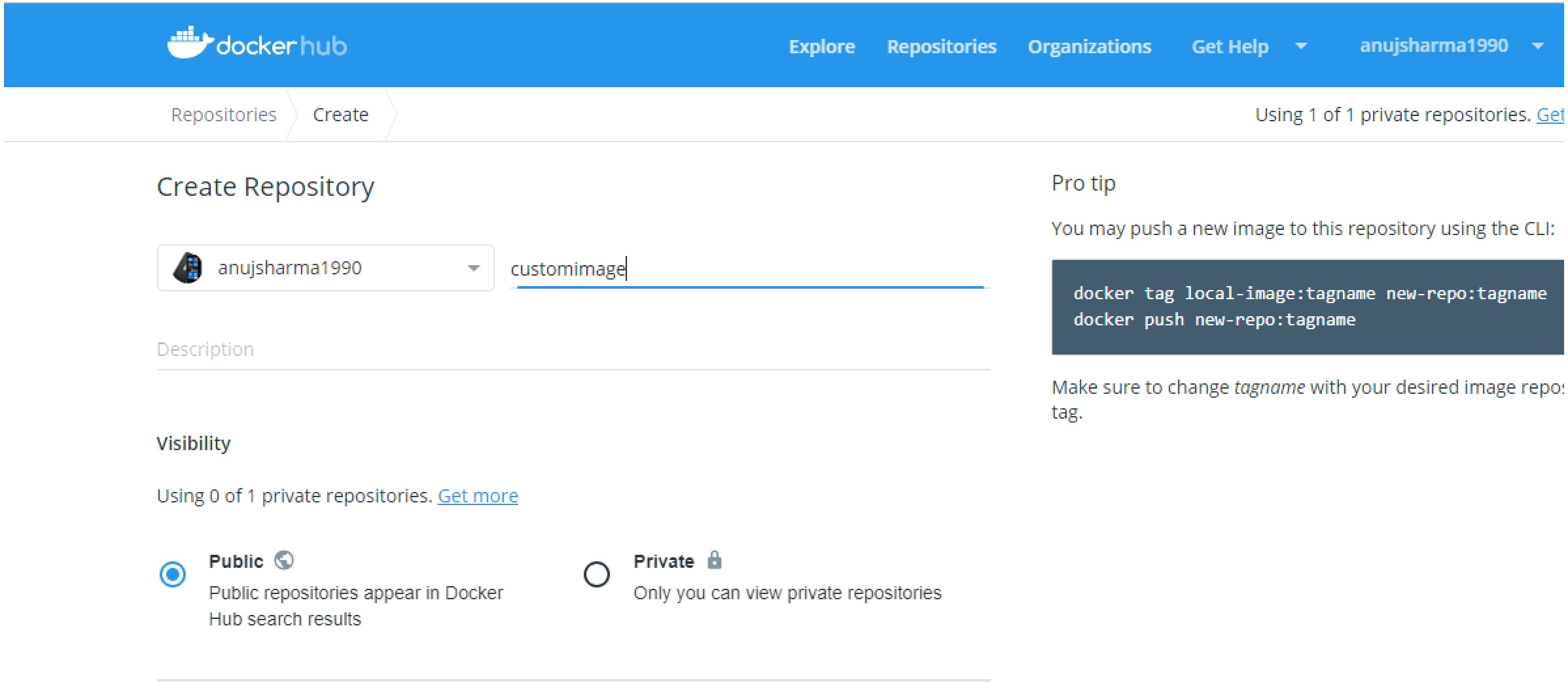


Build and Ship any Application Anywhere

Docker Hub is the world's easiest way to create, manage, and deliver your teams' container applications.

Publishing to Docker Hub

- We can provide repository name and visibility to create a new repository in Docker hub which can be used to push Docker images different versions.



Using 1 of 1 private repositories. Get

Create Repository

anujsharma1990 customimage

Description

Visibility

Using 0 of 1 private repositories. Get more

(●) **Public** Public repositories appear in Docker Hub search results

( ) **Private** Only you can view private repositories

Pro tip

You may push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired image repo
tag.

Publishing to Docker Hub

- Once repo is created we need to rename existing image according to this Docker hub repository name.
- Use below commands to create a different tag with repo name.

```
docker tag local-image:tagname reponame:tagname

docker push reponame:tagname
```

local-image Name of the Docker image which we want to modify

reponame Name of the repository created in Docker hub

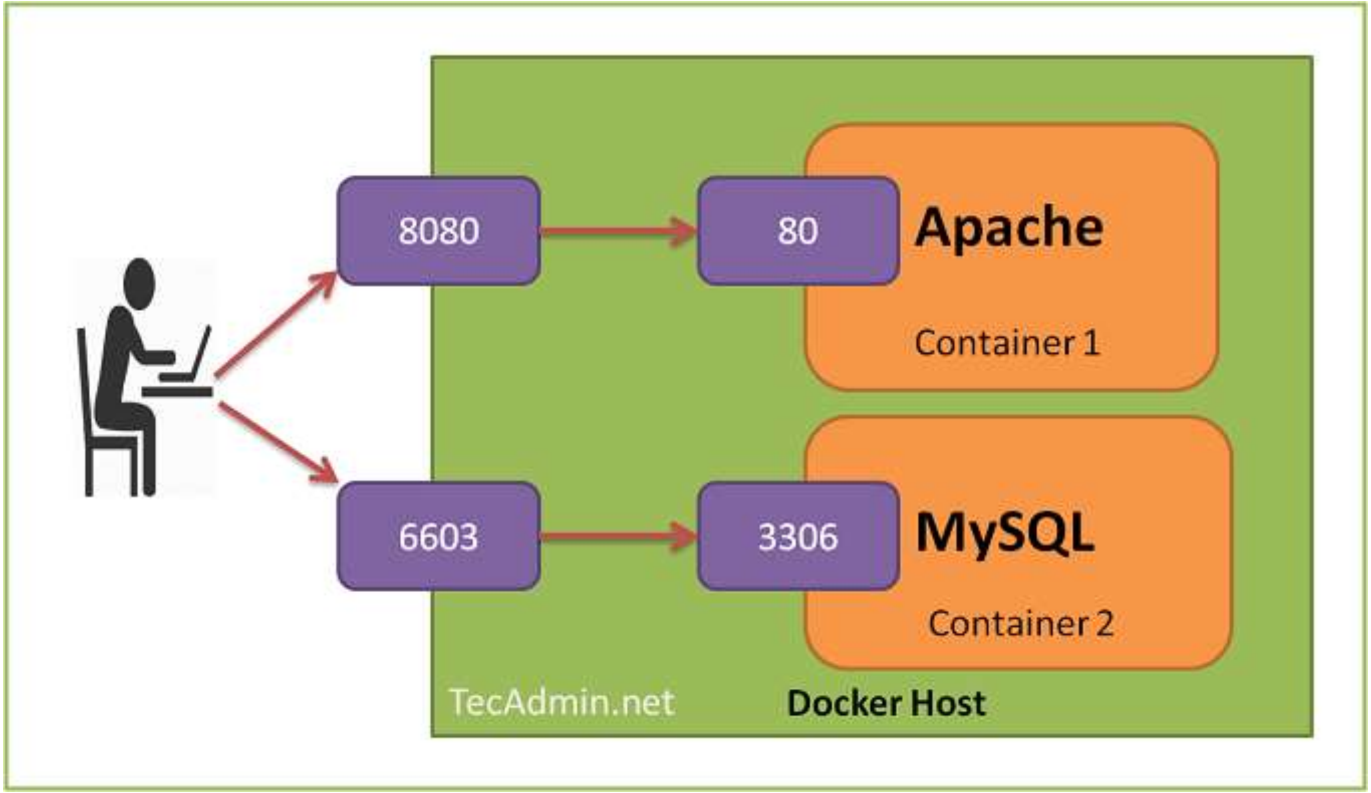
- In order to push Docker image to Docker hub we need to login to save credentials.

docker login Command to save credentials used to push Docker image from local to Docker hub

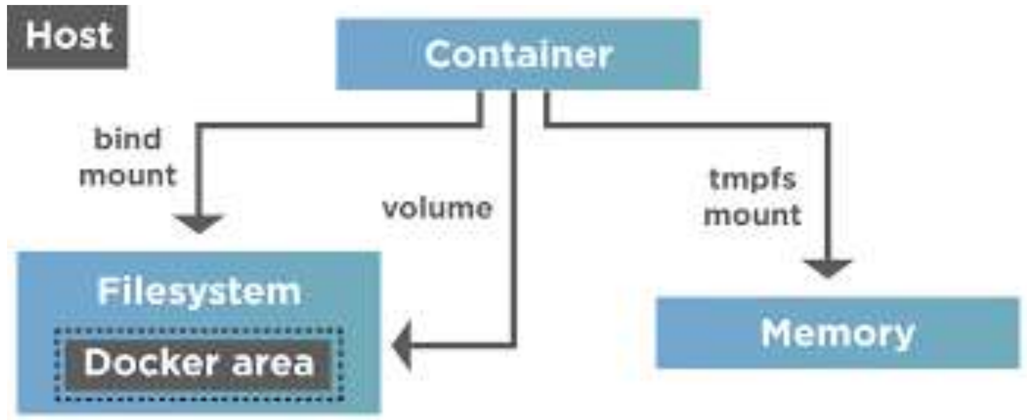
Binding containers with host/port

- Docker containers by default can connect to outside world, but we will not be able to reach out to them from outside server.
- We may have to expose our docker containers to outside world so that we can access applications deployed inside containers.
- We can use –P flag with docker run command to expose all ports built up in docker image.
- In case we want to expose only a specific port we can do that with –p flag.

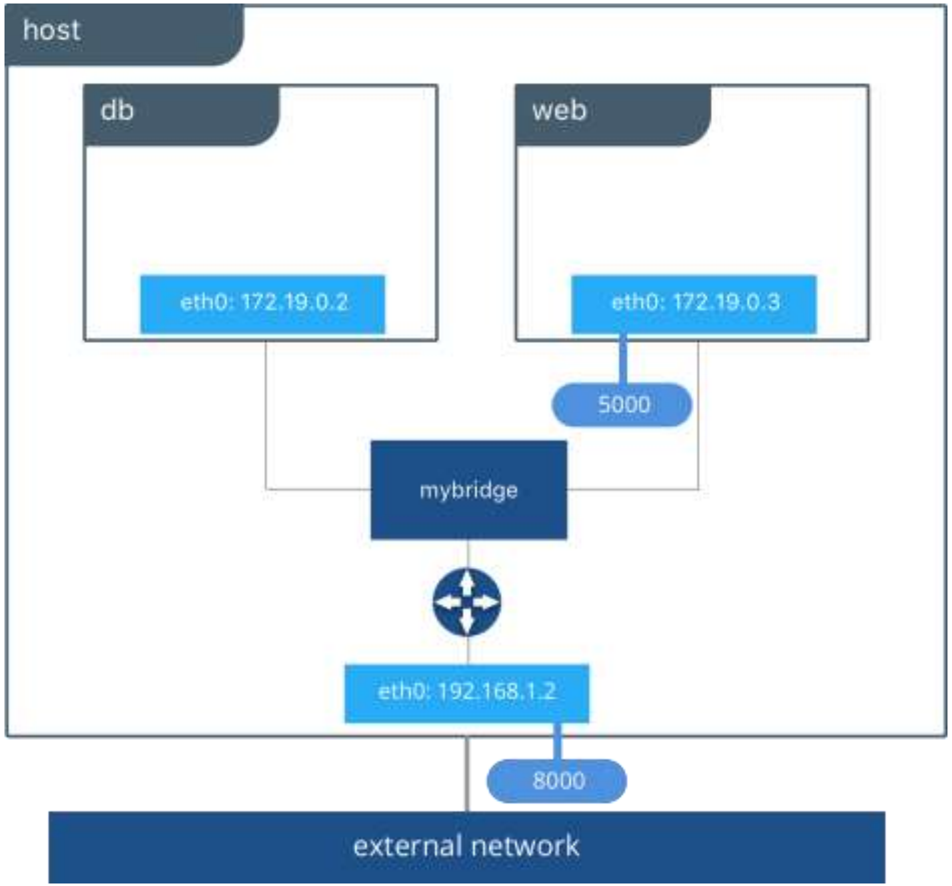
Binding containers with host/port



Filesystem in Containers



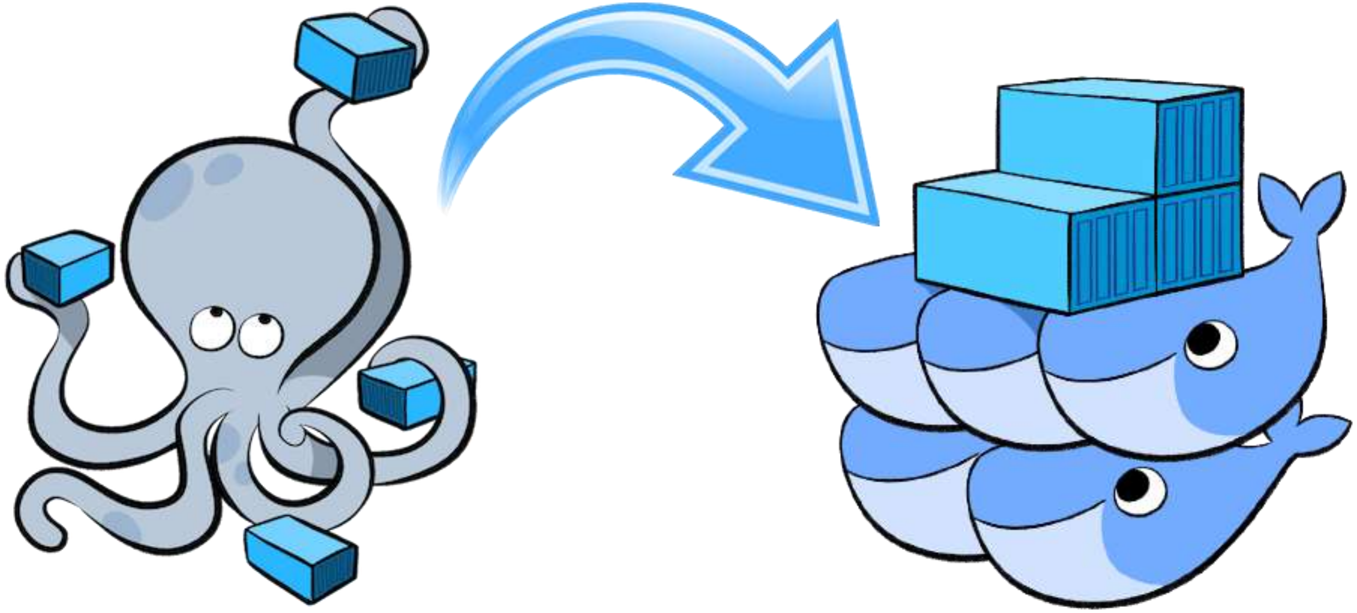
Docker Network



Docker Network Types

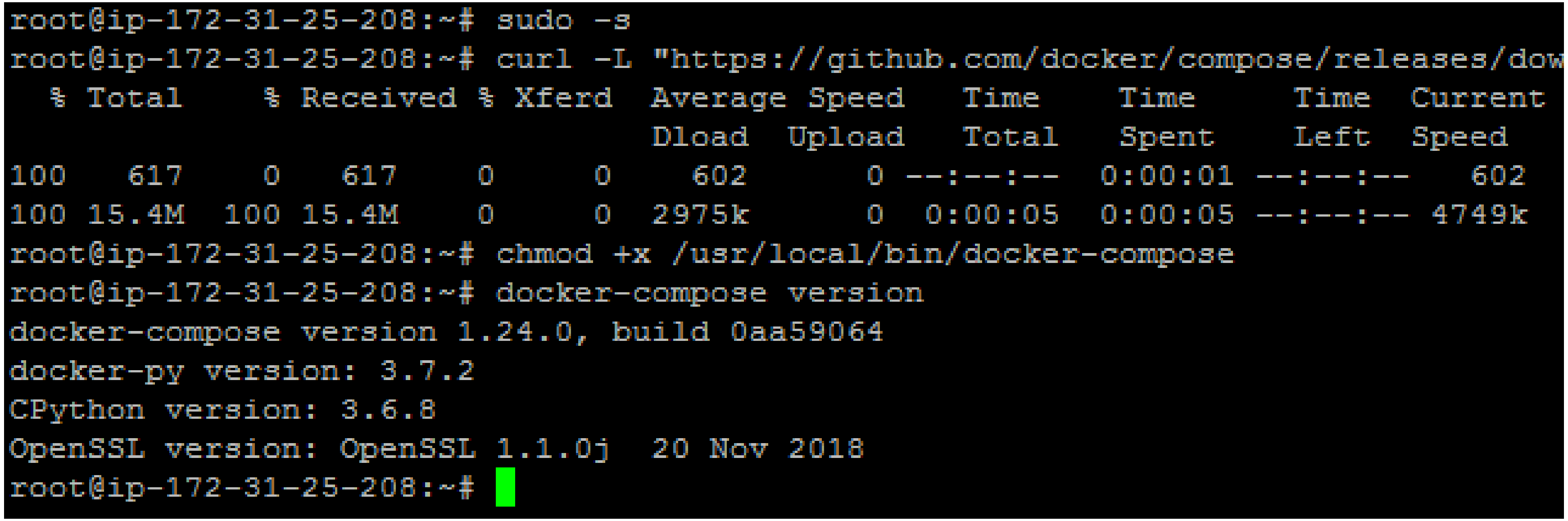
- There are mainly three types of docker networks which can be attached to docker container.
- Depending on docker network, we will be able to interact with docker container accordingly.
- Docker mainly supports four types of network drivers listed as below:
    - Bridge
    - Host
    - Null

Docker Compose



Source: Docker

Docker Compose Setup

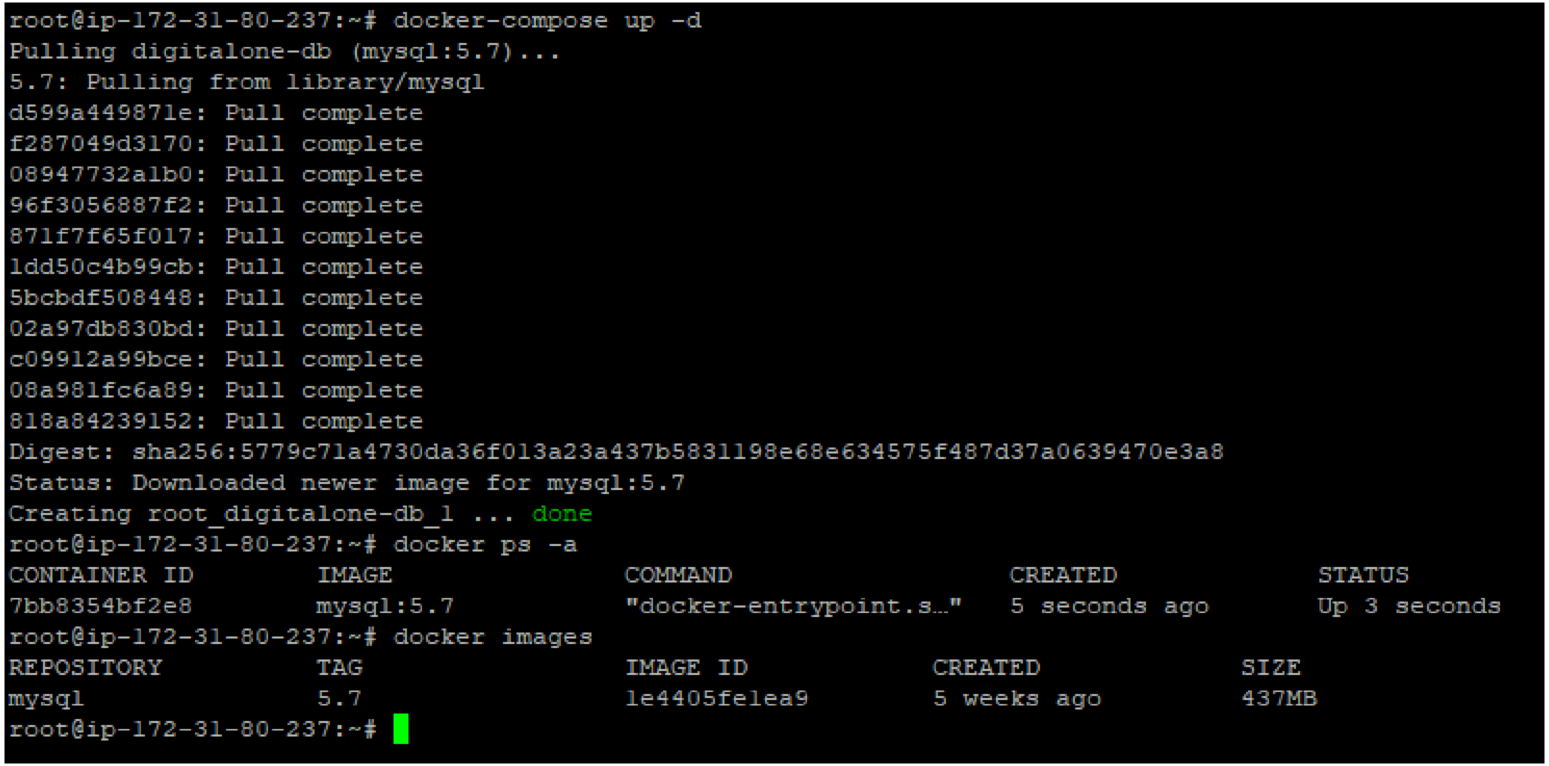


```
root@ip-172-31-25-208:~# sudo -s
root@ip-172-31-25-208:~# curl -L "https://github.com/docker/compose/releases/dow
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   617    0   617    0     0    602      0 --:--:--  0:00:01 --:--:--   602
100 15.4M  100 15.4M    0     0  2975k      0  0:00:05  0:00:05 --:--:-- 4749k
root@ip-172-31-25-208:~# chmod +x /usr/local/bin/docker-compose
root@ip-172-31-25-208:~# docker-compose version
docker-compose version 1.24.0, build 0aa59064
docker-py version: 3.7.2
CPython version: 3.6.8
OpenSSL version: OpenSSL 1.1.0j  20 Nov 2018
root@ip-172-31-25-208:~#
```

Docker Compose Example

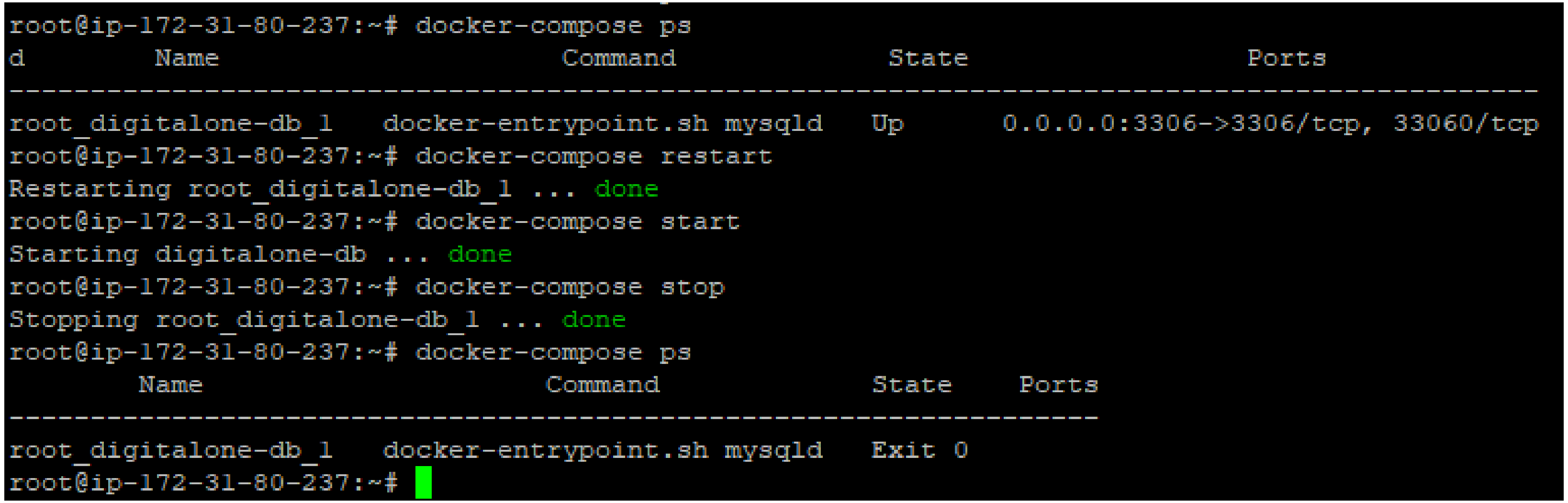
```
version: '3.3'
services:
  digitalone-db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: 'db'
      MYSQL_USER: 'user'
      MYSQL_PASSWORD: 'password'
      MYSQL_ROOT_PASSWORD: 'password'
    ports:
      - '3306:3306'
    volumes:
      - digitalone-volume:/var/lib/mysql

volumes:
  digitalone-volume:
```

Docker Compose Commands



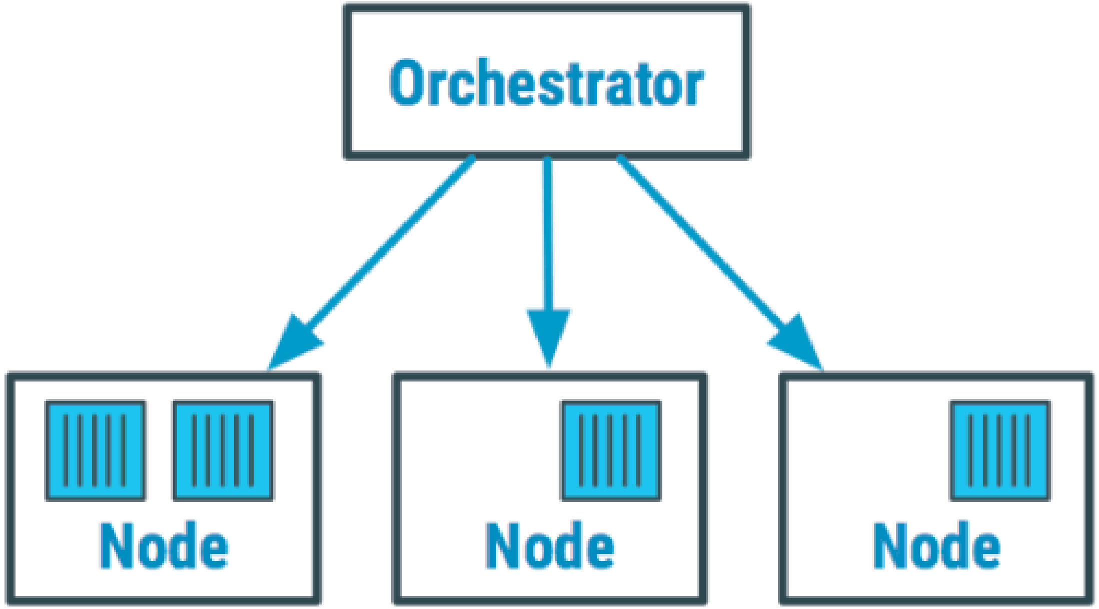
```
root@ip-172-31-80-237:~# docker-compose up -d
Pulling digitalone-db (mysql:5.7)...
5.7: Pulling from library/mysql
d599a449871e: Pull complete
f287049d3170: Pull complete
08947732a1b0: Pull complete
96f3056887f2: Pull complete
871f7f65f017: Pull complete
1dd50c4b99cb: Pull complete
5bcbdf508448: Pull complete
02a97db830bd: Pull complete
c09912a99bce: Pull complete
08a981fc6a89: Pull complete
818a84239152: Pull complete
Digest: sha256:5779c71a4730da36f013a23a437b5831198e68e634575f487d37a0639470e3a8
Status: Downloaded newer image for mysql:5.7
Creating root_digitalone-db_1 ... done
root@ip-172-31-80-237:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
7bb8354bf2e8        mysql:5.7           "docker-entrypoint.s…"   5 seconds ago       Up 3 seconds
root@ip-172-31-80-237:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql               5.7                 1e4405fe1ea9        5 weeks ago         437MB
root@ip-172-31-80-237:~#
```

Docker Compose Commands

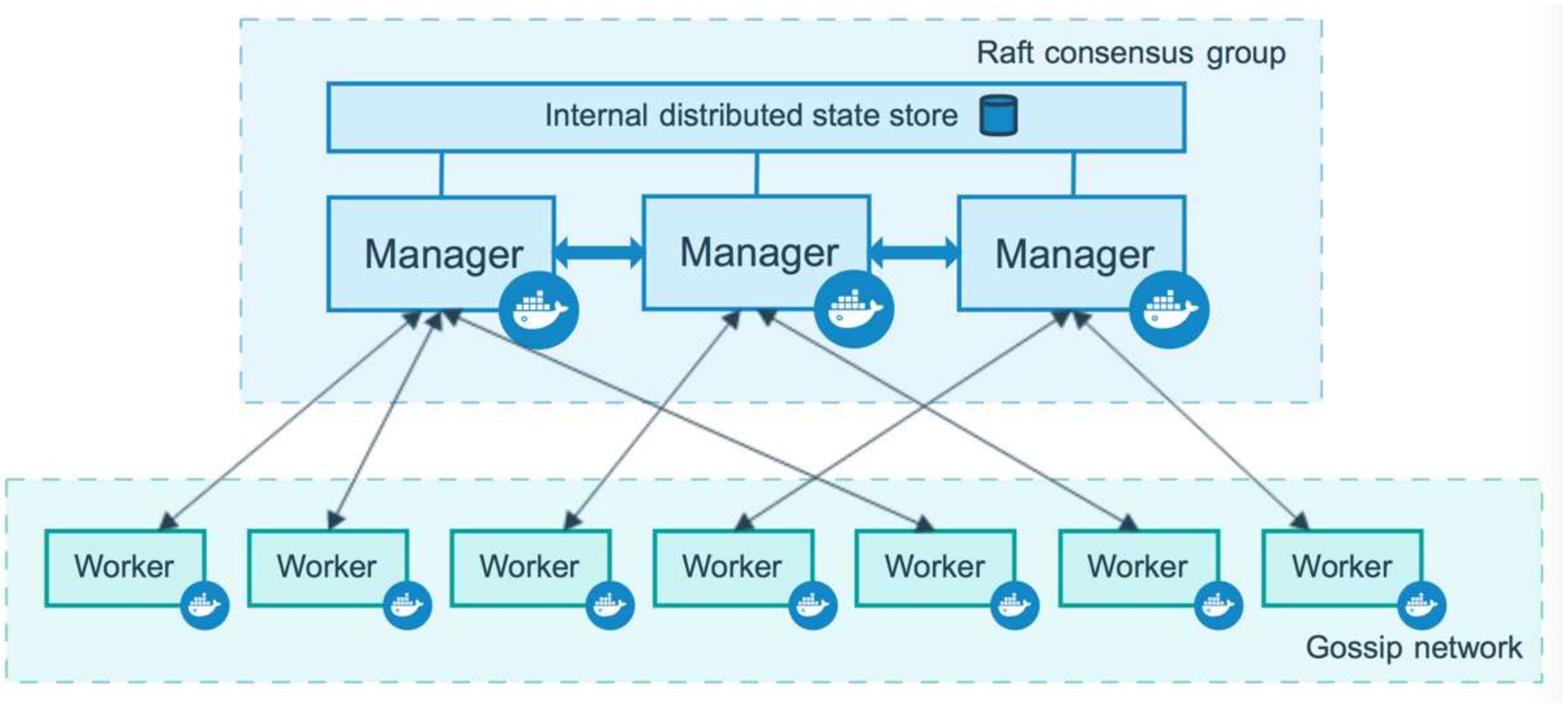


```
root@ip-172-31-80-237:~# docker-compose ps
d         Name                        Command                State                  Ports
----------------------------------------------------------------------------------------------
root_digitalone-db_1   docker-entrypoint.sh mysqld   Up      0.0.0.0:3306->3306/tcp, 33060/tcp
root@ip-172-31-80-237:~# docker-compose restart
Restarting root_digitalone-db_1 ... done
root@ip-172-31-80-237:~# docker-compose start
Starting digitalone-db ... done
root@ip-172-31-80-237:~# docker-compose stop
Stopping root_digitalone-db_1 ... done
root@ip-172-31-80-237:~# docker-compose ps
        Name                        Command                State     Ports
-------------------------------------------------------------------
root_digitalone-db_1   docker-entrypoint.sh mysqld   Exit 0
root@ip-172-31-80-237:~#
```

Docker Containers Orchestration



Docker Swarm

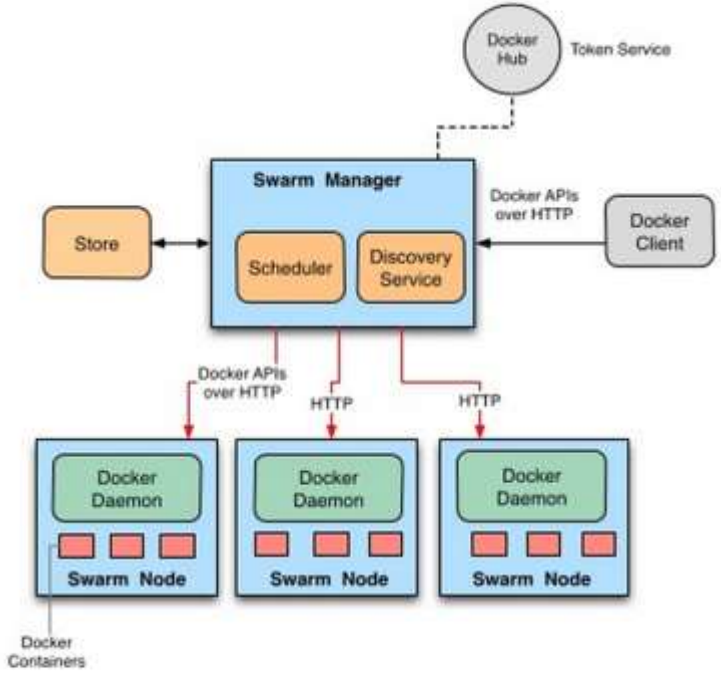


Source: Docker

Docker Swarm Features

Docker swarm supports lot of features which we get with orchestration deployment. Some of them are listed below:

- Scaling
- Multi node cluster
- Service discovery
- Load Balancing



Docker Swarm Advantages

- Below are some of the benefits which we get with swarm implementation:
- Docker swarm is tightly integrated with Docker ecosystem, and implements Docker API.
- Since developed by Docker itself, Docker swarm gets smoothly integrated with Docker.
- Swarm is less expensive as compared to any other container orchestration tool.
- Swarm supports low fault tolerance feature as compared to Kubernetes.
- Docker swarm has low learning curve which means swarm is easy to learn without any complexity.

Docker Swarm Disadvantages

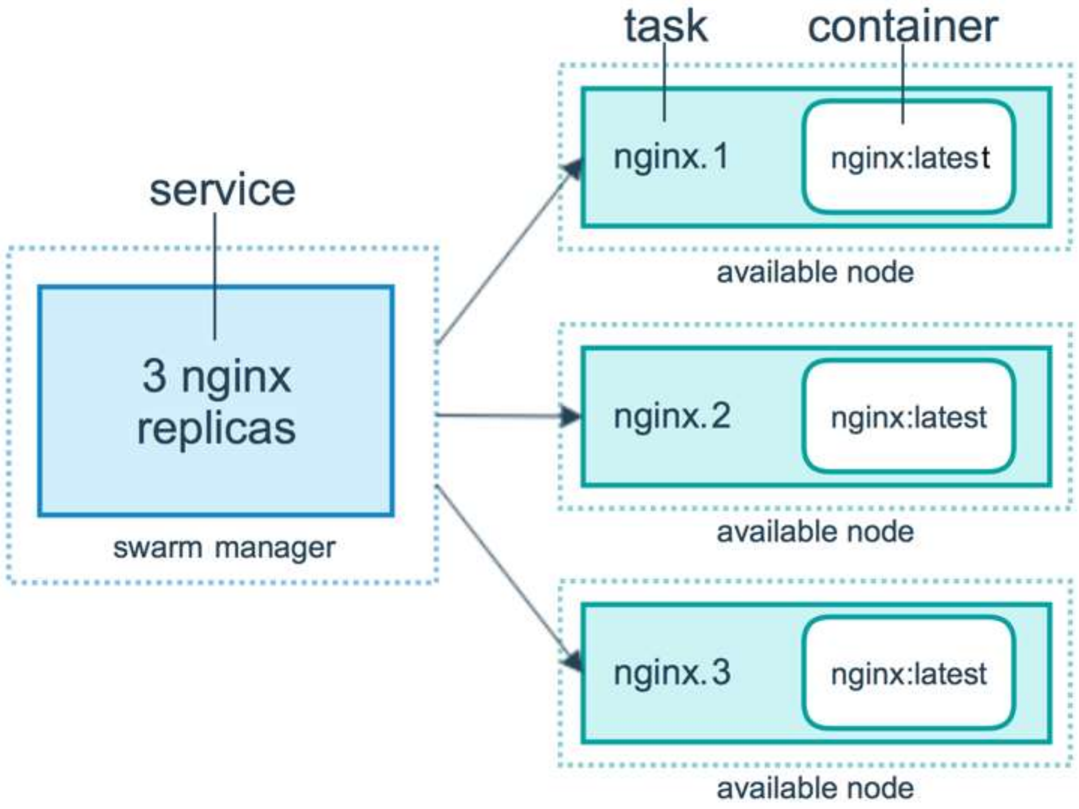
- Although Docker swarm provides some advantages but still there are some disadvantages with swarm cluster:
- Docker Swarm provides limited functionality.
- Docker Swarm has limited fault tolerance.
- Docker Swarm have smaller community as compared to Kubernetes community.
- In Docker Swarm, services can only be scaled manually, we don’t have any autoscaling feature with Docker swarm.

Docker Swarm Nodes

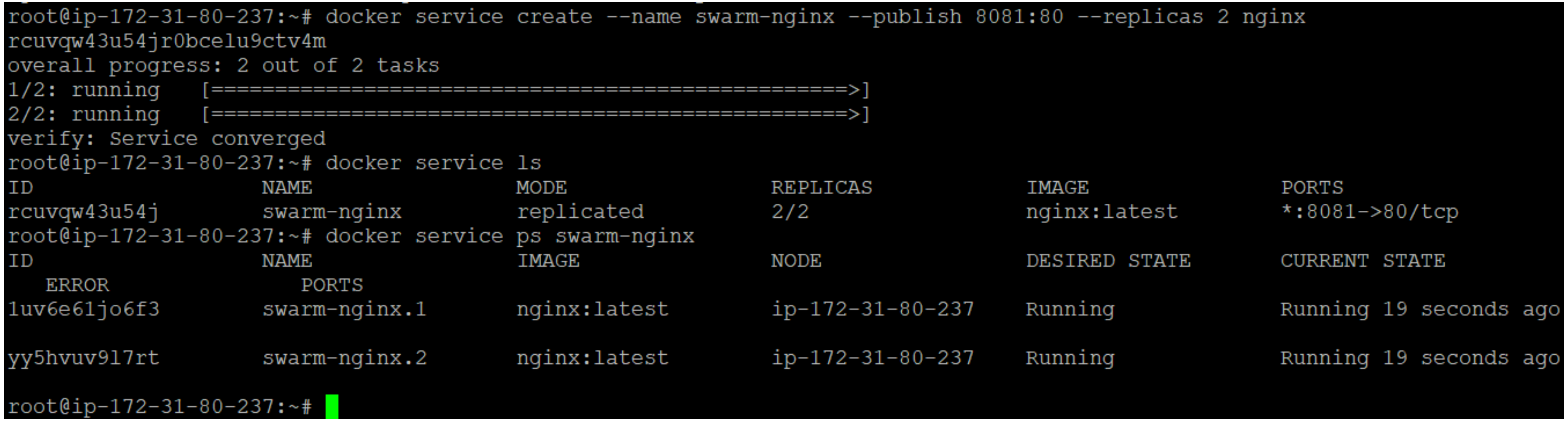
Docker Swarm architecture comprises of various machines and some of them are listed as below:

- Manager Node
- Leader Node
- Worker Node

Docker Service

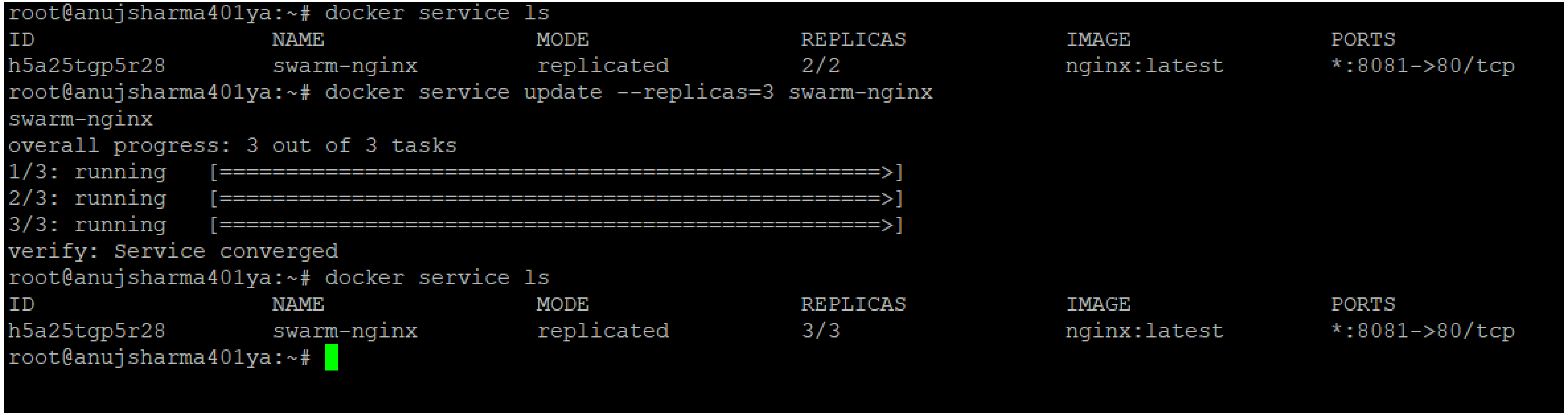


Docker Service Setup



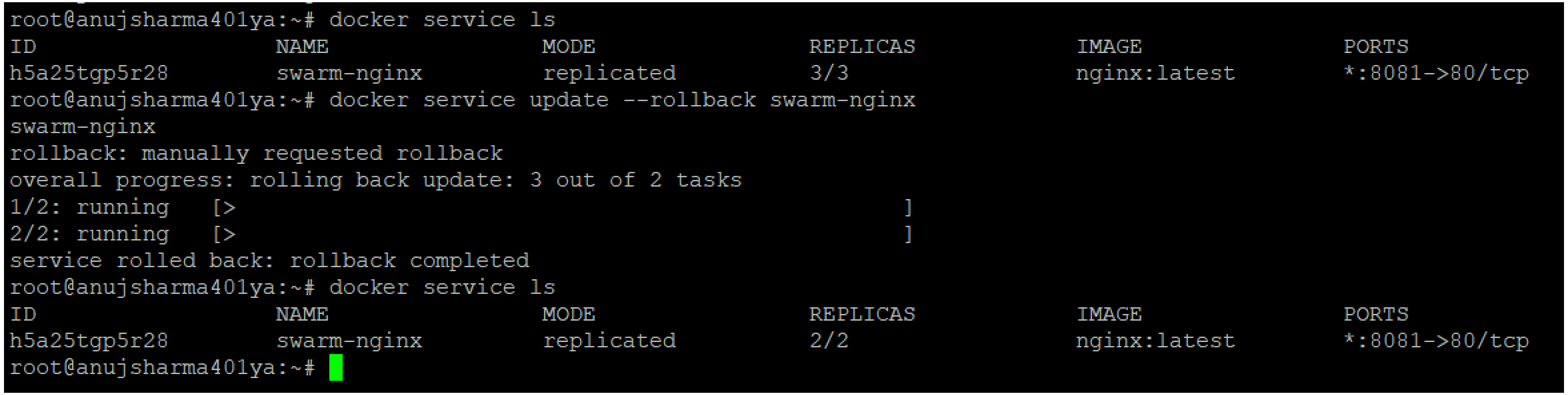
```
root@ip-172-31-80-237:~# docker service create --name swarm-nginx --publish 8081:80 --replicas 2 nginx
rcuvqw43u54jr0bcelu9ctv4m
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
root@ip-172-31-80-237:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
rcuvqw43u54j        swarm-nginx         replicated          2/2                 nginx:latest        *:8081->80/tcp
root@ip-172-31-80-237:~# docker service ps swarm-nginx
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
   ERROR               PORTS
luv6e61jo6f3        swarm-nginx.1       nginx:latest        ip-172-31-80-237    Running             Running 19 seconds ago

yy5hvuv9l7rt        swarm-nginx.2       nginx:latest        ip-172-31-80-237    Running             Running 19 seconds ago

root@ip-172-31-80-237:~#
```

Docker Service Update



```
root@anujsharma401ya:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
h5a25tgp5r28        swarm-nginx         replicated          2/2                 nginx:latest        *:8081->80/tcp
root@anujsharma401ya:~# docker service update --replicas=3 swarm-nginx
swarm-nginx
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@anujsharma401ya:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
h5a25tgp5r28        swarm-nginx         replicated          3/3                 nginx:latest        *:8081->80/tcp
root@anujsharma401ya:~#
```

Docker Service Update Rollback



```
root@anujsharma401ya:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
h5a25tgp5r28        swarm-nginx         replicated          3/3                 nginx:latest        *:8081->80/tcp
root@anujsharma401ya:~# docker service update --rollback swarm-nginx
swarm-nginx
rollback: manually requested rollback
overall progress: rolling back update: 3 out of 2 tasks
1/2: running   [>                                                  ]
2/2: running   [>                                                  ]
service rolled back: rollback completed
root@anujsharma401ya:~# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
h5a25tgp5r28        swarm-nginx         replicated          2/2                 nginx:latest        *:8081->80/tcp
root@anujsharma401ya:~#
```