

Containers vs VMs



Operating Systems
have **2 Layers**

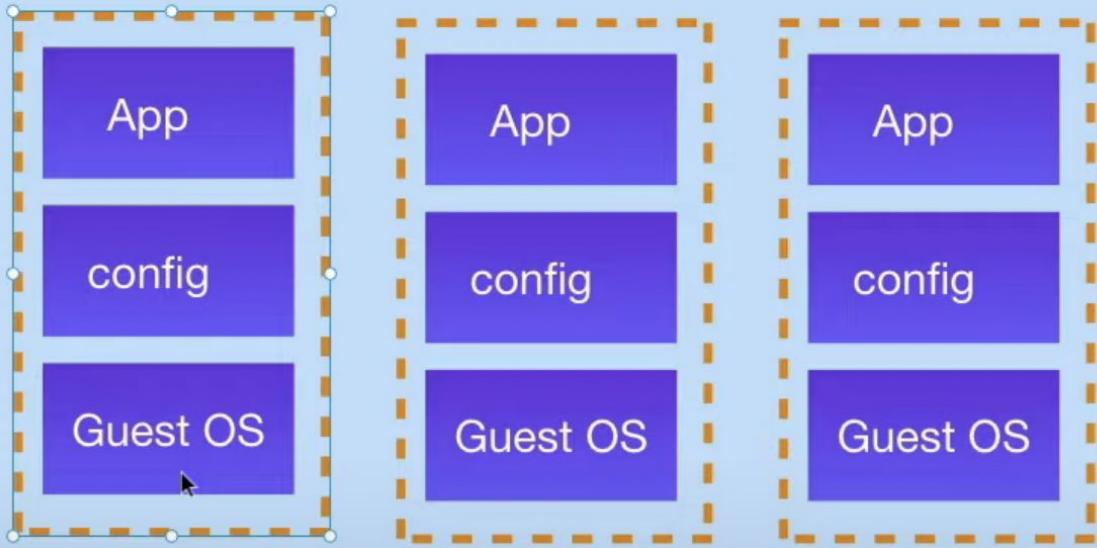
Applications

2. Layer

OS Kernel

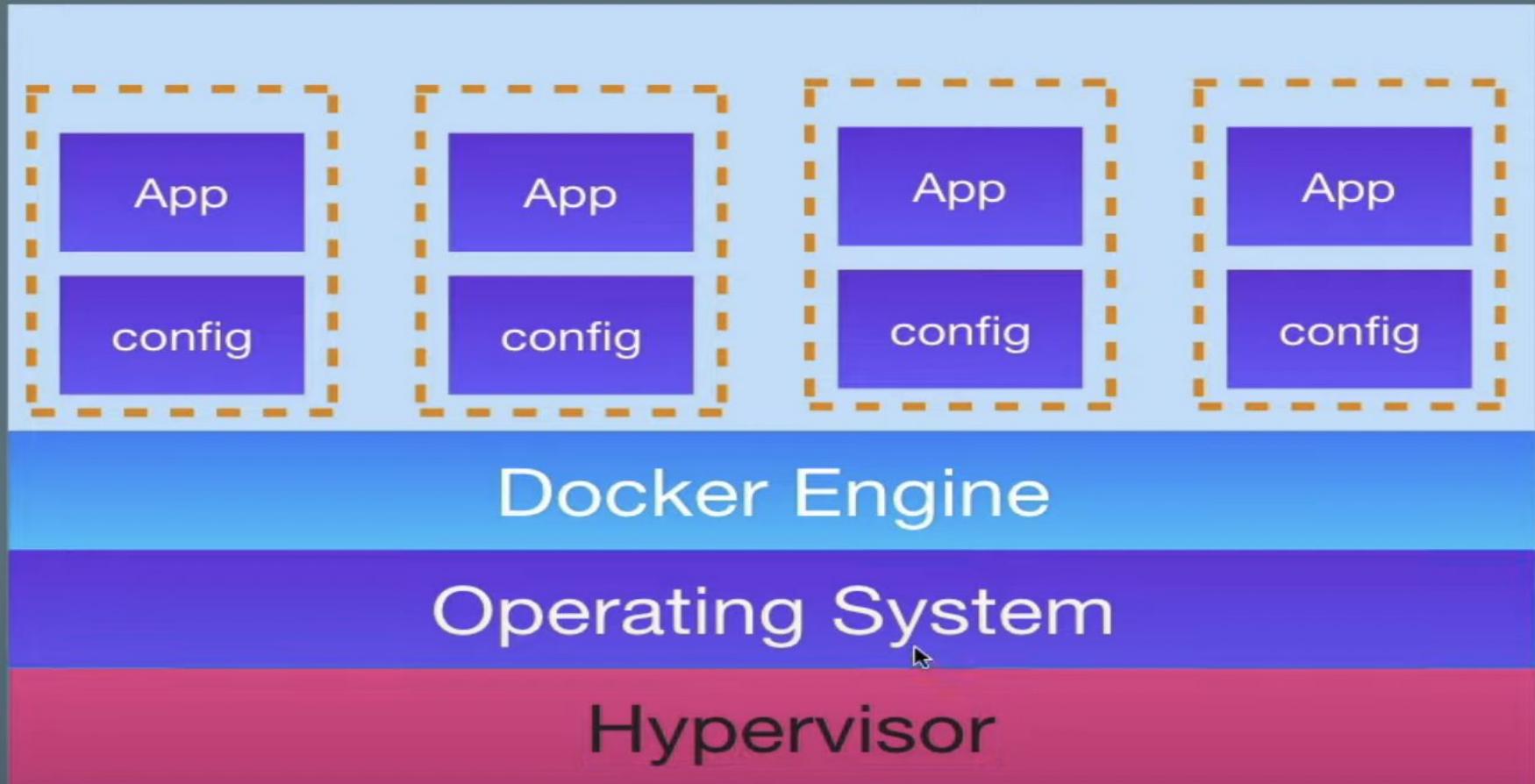
I. Layer

Hardware

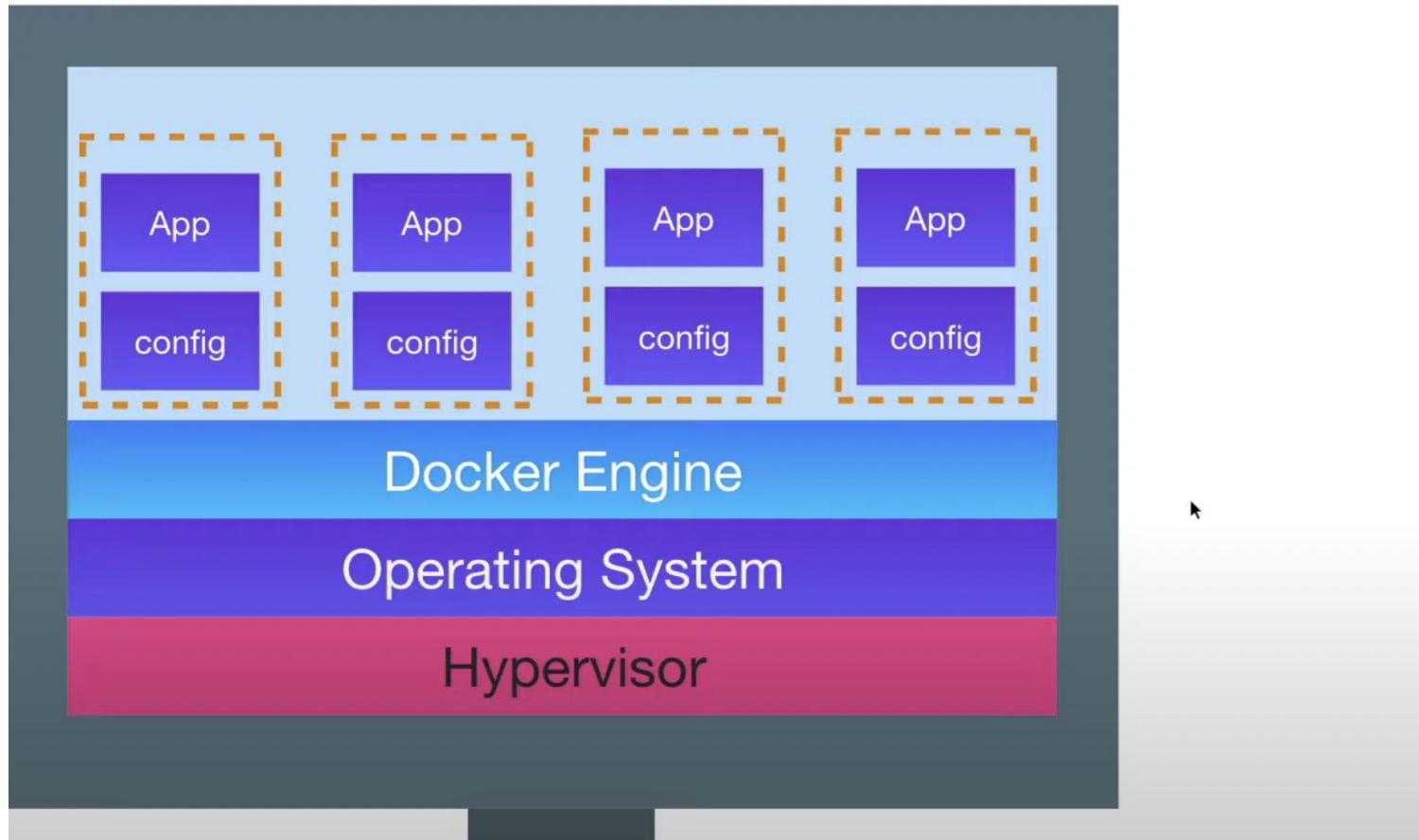


Hypervisor

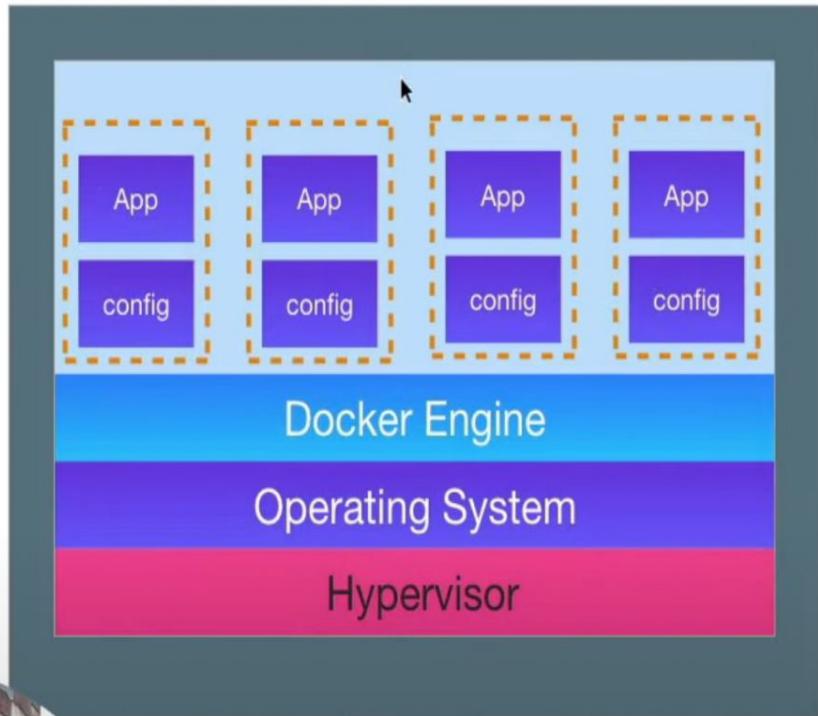
- 1) Its resource hungry and consume lot of resources
- 2) The guest os can be ubuntu its wastage of many resources
- 3)



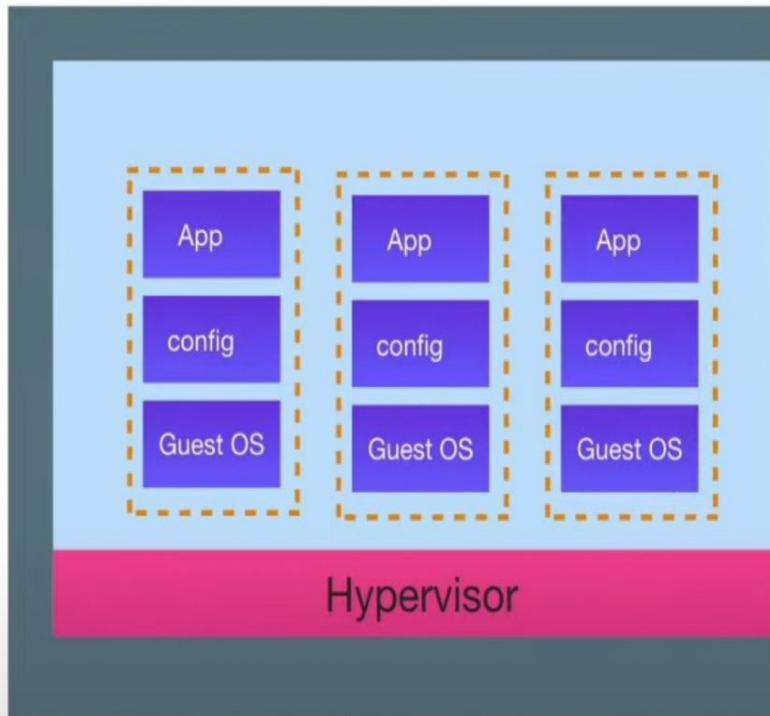
Container



Container



VM



App Dependencies

- Java App
- NodeJS App
- Python App
- Which version ?

Multiple Versions



5.1



6.13



7.4



5.7



8.2

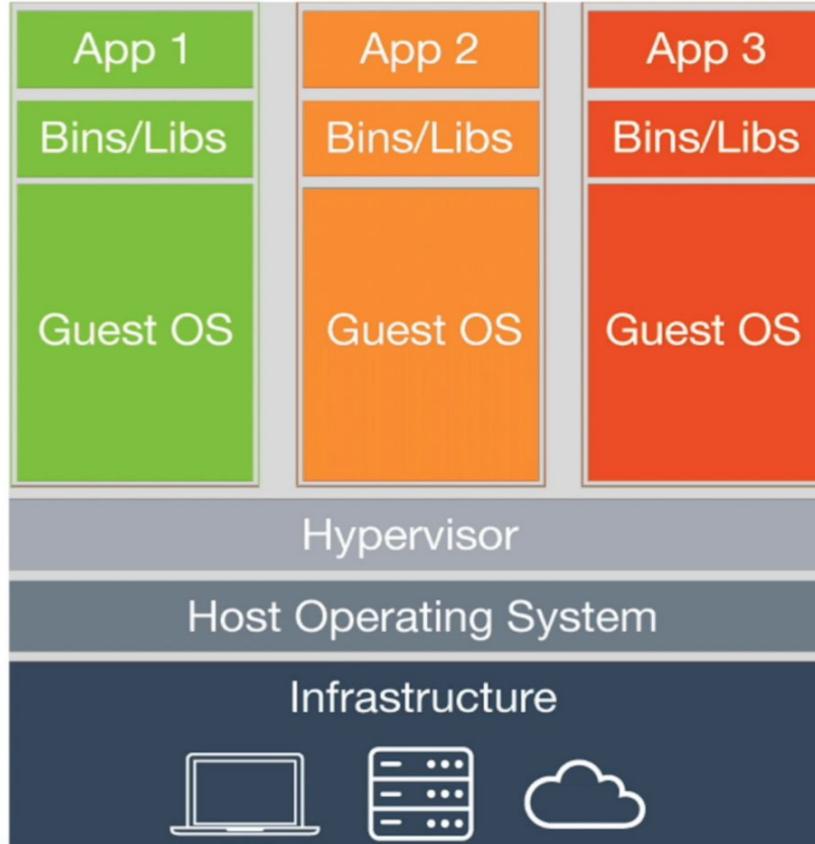


9.1

1. We can also add the dependencies along with application itself
- 2.

Virtual Machines

- OS run on OS



Virtual Machine Example



Virtual Machine Example



Guest OS / Virtual Machine



Hypervisor

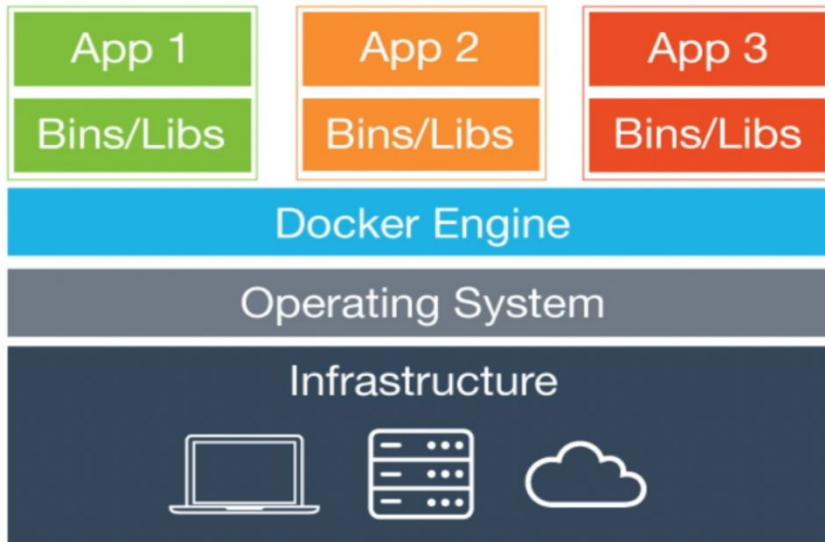


Host OS



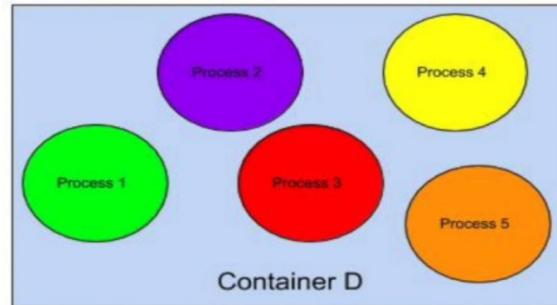
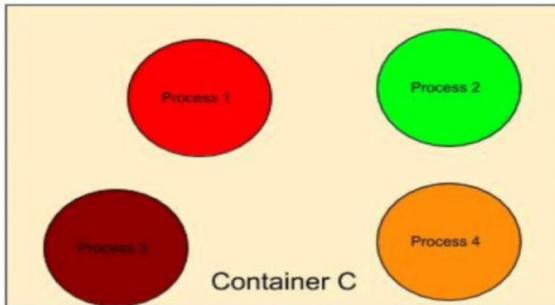
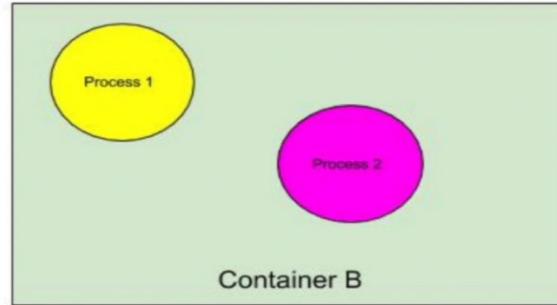
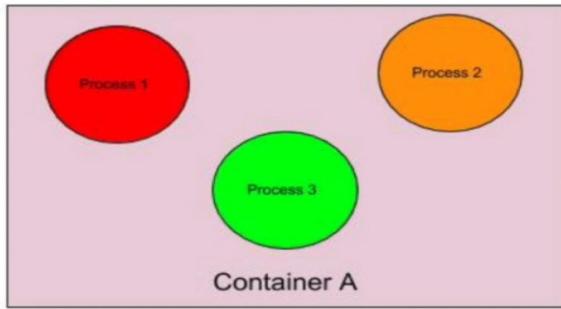
Physical Machine

Containers



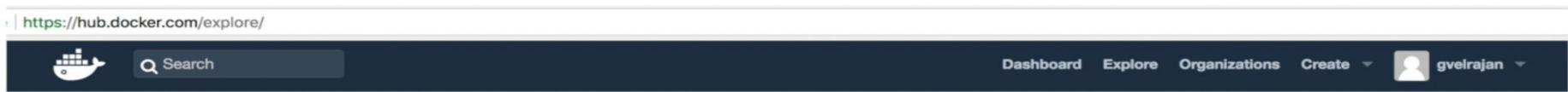
- OS - Kernel - libs/bins
- Container = App + Deps
- **Container Hypervisors:**
 - A. Docker Engine,
 - B. Core OS Rocket,
 - C. Canonical LXC/LXD

Containers



Docker Hub

https://hub.docker.com/explore/



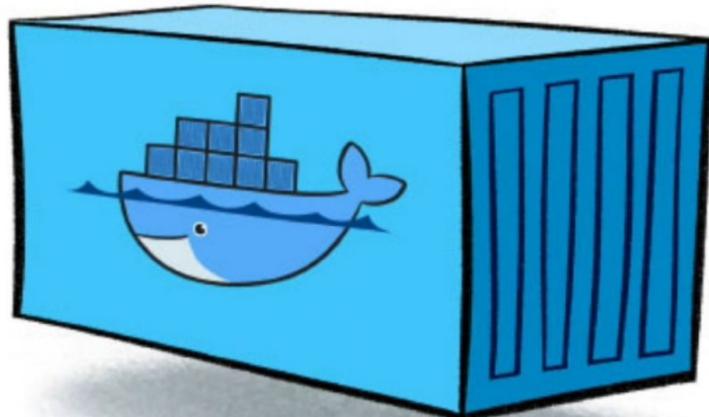
The navigation bar at the top of the Docker Hub page includes a search icon and field, a user profile for 'gvelrajan', and links for Dashboard, Explore, Organizations, and Create.

Explore Official Repositories

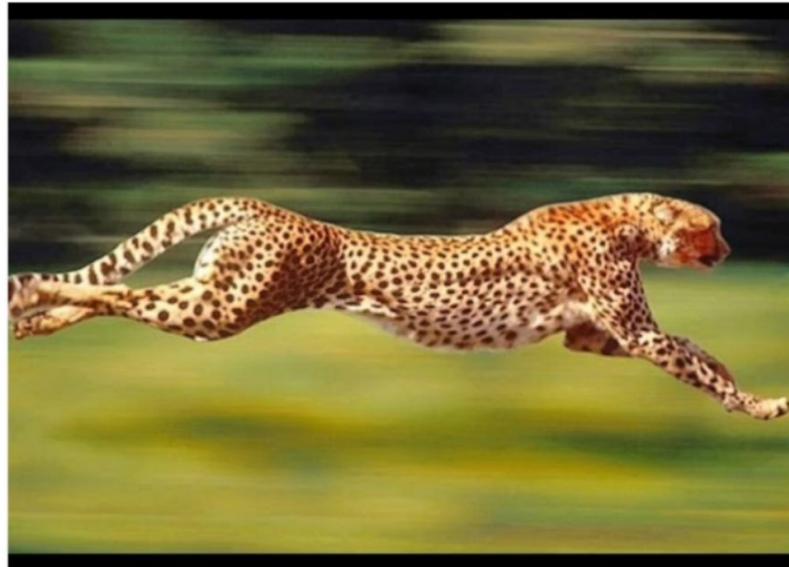
 nginx official	9.3K STARS	10M+ PULLS	DETAILS
 alpine official	4.1K STARS	10M+ PULLS	DETAILS
 busybox official	1.3K STARS	10M+ PULLS	DETAILS
 httpd official	1.9K STARS	10M+ PULLS	DETAILS
 redis official	5.6K STARS	10M+ PULLS	DETAILS

Docker Benefits

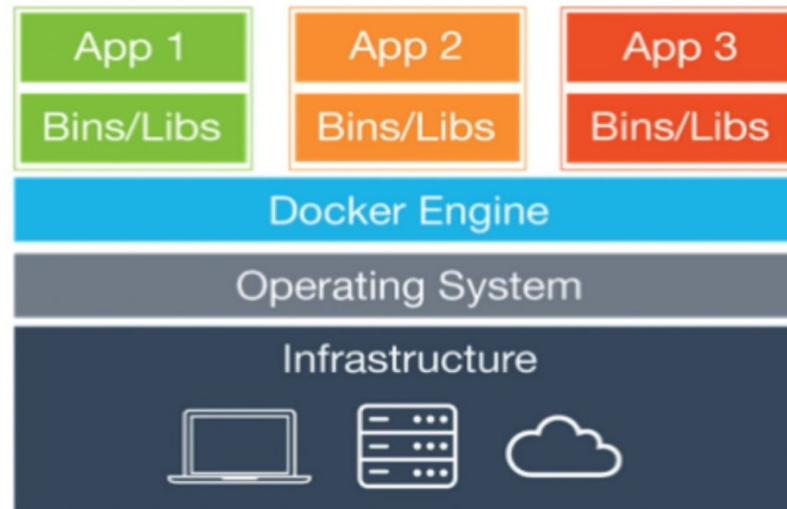
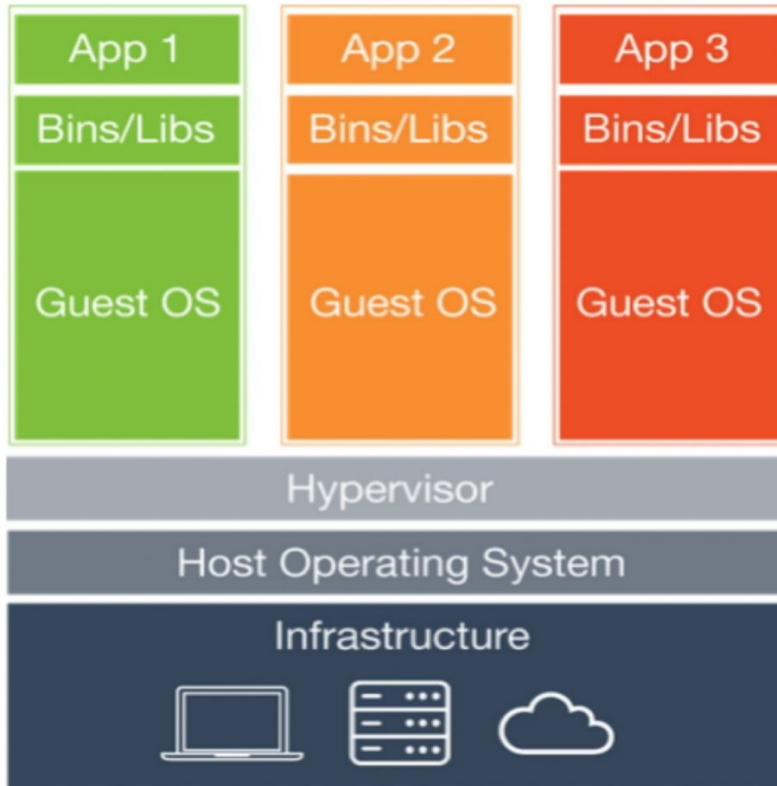
- Isolation
- App + Dependencies
- Run anywhere



Virtual Machines vs. Containers



Virtual Machine vs. Containers



Virtual Machines vs. Containers

- Virtual Machines
- Huge in size(in GB)
- Have a Kernel
- Have unwanted Libs, Bins, and Utils
- More CPU & Memory
- More time to boot up
- Containers
- Small in size(in MB)
- Have no Kernel
- Have bare-minimal Libs, Bins and Utils
- Light weight & fast
- Less CPU & Memory

```
1 Docker Container Basics:  
2 =====  
3  
4 docker pull <image name>  
5 docker pull nginx  
6  
7 docker run -d -p 80:80 nginx  
8  
9 docker run -it -p 80:80 nginx  
0  
1 docker container logs <id>  
2  
3 docker exec <container name or id > <command to run>  
4 docker exec afbg ls  
5  
6 docker exec -it <container id> /bin/bash  
7  
8 docker container stop <id>  
9  
0 docker container start <id>  
1
```

Line 11, Column 27

```
① https://ssh.cloud.google.com/projects/docker-demo-218413/zones/asia-south1-c/instances/docker?authuser=1 ...  
gannygans@docker:~$ docker container logs  
"docker container logs" requires exactly 1 argument.  
See 'docker container logs --help'.  
  
Usage: docker container logs [OPTIONS] CONTAINER  
  
Fetch the logs of a container  
gannygans@docker:~$ docker pull  
"docker pull" requires exactly 1 argument.  
See 'docker pull --help'.  
  
Usage: docker pull [OPTIONS] NAME[:TAG|@DIGEST]  
  
Pull an image or a repository from a registry  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$ docker image ls  
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE  
gannygans@docker:~$ docker container ls  
CONTAINER ID        IMAGE      COMMAND      CREATED       STATUS  
PORTS               NAMES  
gannygans@docker:~$
```

```
1  
2  
3 docker container run -it alpine  
4  
5 docker ps -a | grep alpine  
6  
7 docker container diff <id>  
8  
9 docker container commit <id>  
0  
1 docker image ls  
2  
3 docker image tag <id> gvelrajan/helloworld:  
  
Get cloud support with Ubuntu Advantage Cloud Guest:  
http://www.ubuntu.com/business/services/cloud  
  
5 packages can be updated.  
0 updates are security updates.  
  
New release '18.04.1 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Fri Oct  5 08:08:12 2018 from 173.194.93.34  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$  
gannygans@docker:~$ docker ps -a  
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS              PORTS  
      NAMES  
gannygans@docker:~$ docker image ls  
REPOSITORY          TAG                 IMAGE ID         CREATED          SIZE  
gannygans@docker:~$  
Line 13, Column 33
```

[raghav]\$

[raghav]\$ docker -v

Docker version 18.03.1-ce, build 9ee9f40

[raghav]\$

[raghav]\$ docker images

REPOSITORY	Today you will learn :	IMAGE ID	CREATED
------------	------------------------	----------	---------

SIZE

[raghav]\$ clear

1. What are images
2. How to pull image
3. How to run a container using an image
4. Basic Commands

```
[raghav$ docker images --help
Usage: docker images [OPTIONS] [REPOSITORY[:TAG]]
      Today we will learn :
List images
Docker Basic Command...
Yesterday Step by Step f...
Options:
How to create Dockerfile
Monday Docker Basic Com... -a, --all Show all images (default hides intermediate images)
--digests Show digests
How -f, --filter filter Filter output based on conditions provided
Monday No additional text --format string RIC Pretty-print images using a Go template
--no-trunc Don't truncate output
--quiet Only show numeric IDs
raghav$
```

Lists images

BEGINNERS

Docker Volumes

Options: Step f...

-a, --all

Show all images (default hides intermediate images)

Docker Basic Comma...

Yesterday Step by...

How to create Dockerf...

Monday Docker Basic Co...

--digests

1. What are images

2. How to pull images

3. How to run images

4. Basic Con...

5. Don'ts

6. Truncate output

7. No additional text

8. Create Dockerf...

9. Docker Compose

10. Docker Swarm

11. Docker Compose

12. Docker Compose

13. Docker Compose

14. Docker Compose

15. Docker Compose

16. Docker Compose

17. Docker Compose

18. Docker Compose

19. Docker Compose

20. Docker Compose

21. Docker Compose

22. Docker Compose

23. Docker Compose

24. Docker Compose

25. Docker Compose

26. Docker Compose

27. Docker Compose

28. Docker Compose

29. Docker Compose

30. Docker Compose

31. Docker Compose

32. Docker Compose

33. Docker Compose

34. Docker Compose

35. Docker Compose

36. Docker Compose

37. Docker Compose

38. Docker Compose

39. Docker Compose

40. Docker Compose

41. Docker Compose

42. Docker Compose

43. Docker Compose

44. Docker Compose

45. Docker Compose

46. Docker Compose

47. Docker Compose

48. Docker Compose

49. Docker Compose

50. Docker Compose

51. Docker Compose

52. Docker Compose

53. Docker Compose

54. Docker Compose

55. Docker Compose

56. Docker Compose

57. Docker Compose

58. Docker Compose

59. Docker Compose

60. Docker Compose

61. Docker Compose

62. Docker Compose

63. Docker Compose

64. Docker Compose

65. Docker Compose

66. Docker Compose

67. Docker Compose

68. Docker Compose

69. Docker Compose

70. Docker Compose

71. Docker Compose

72. Docker Compose

73. Docker Compose

74. Docker Compose

75. Docker Compose

76. Docker Compose

77. Docker Compose

78. Docker Compose

79. Docker Compose

80. Docker Compose

81. Docker Compose

82. Docker Compose

83. Docker Compose

84. Docker Compose

85. Docker Compose

86. Docker Compose

87. Docker Compose

88. Docker Compose

89. Docker Compose

90. Docker Compose

91. Docker Compose

92. Docker Compose

93. Docker Compose

94. Docker Compose

95. Docker Compose

96. Docker Compose

97. Docker Compose

98. Docker Compose

99. Docker Compose

100. Docker Compose

101. Docker Compose

102. Docker Compose

103. Docker Compose

104. Docker Compose

105. Docker Compose

106. Docker Compose

107. Docker Compose

108. Docker Compose

109. Docker Compose

110. Docker Compose

111. Docker Compose

112. Docker Compose

113. Docker Compose

114. Docker Compose

115. Docker Compose

116. Docker Compose

117. Docker Compose

118. Docker Compose

119. Docker Compose

120. Docker Compose

121. Docker Compose

122. Docker Compose

123. Docker Compose

124. Docker Compose

125. Docker Compose

126. Docker Compose

127. Docker Compose

128. Docker Compose

129. Docker Compose

130. Docker Compose

131. Docker Compose

132. Docker Compose

133. Docker Compose

134. Docker Compose

135. Docker Compose

136. Docker Compose

137. Docker Compose

138. Docker Compose

139. Docker Compose

140. Docker Compose

141. Docker Compose

142. Docker Compose

143. Docker Compose

144. Docker Compose

145. Docker Compose

146. Docker Compose

147. Docker Compose

148. Docker Compose

149. Docker Compose

150. Docker Compose

151. Docker Compose

152. Docker Compose

153. Docker Compose

154. Docker Compose

155. Docker Compose

156. Docker Compose

157. Docker Compose

158. Docker Compose

159. Docker Compose

160. Docker Compose

161. Docker Compose

162. Docker Compose

163. Docker Compose

164. Docker Compose

165. Docker Compose

166. Docker Compose

167. Docker Compose

168. Docker Compose

169. Docker Compose

170. Docker Compose

171. Docker Compose

172. Docker Compose

173. Docker Compose

174. Docker Compose

175. Docker Compose

176. Docker Compose

177. Docker Compose

178. Docker Compose

179. Docker Compose

180. Docker Compose

181. Docker Compose

182. Docker Compose

183. Docker Compose

184. Docker Compose

185. Docker Compose

186. Docker Compose

187. Docker Compose

188. Docker Compose

189. Docker Compose

190. Docker Compose

191. Docker Compose

192. Docker Compose

193. Docker Compose

194. Docker Compose

195. Docker Compose

196. Docker Compose

197. Docker Compose

198. Docker Compose

199. Docker Compose

200. Docker Compose

201. Docker Compose

202. Docker Compose

203. Docker Compose

204. Docker Compose

205. Docker Compose

206. Docker Compose

207. Docker Compose

208. Docker Compose

209. Docker Compose

210. Docker Compose

211. Docker Compose

212. Docker Compose

213. Docker Compose

214. Docker Compose

215. Docker Compose

216. Docker Compose

217. Docker Compose

218. Docker Compose

219. Docker Compose

220. Docker Compose

221. Docker Compose

222. Docker Compose

223. Docker Compose

224. Docker Compose

225. Docker Compose

226. Docker Compose

227. Docker Compose

228. Docker Compose

229. Docker Compose

230. Docker Compose

231. Docker Compose

232. Docker Compose

233. Docker Compose

234. Docker Compose

235. Docker Compose

236. Docker Compose

237. Docker Compose

238. Docker Compose

239. Docker Compose

240. Docker Compose

241. Docker Compose

242. Docker Compose

243. Docker Compose

244. Docker Compose

245. Docker Compose

246. Docker Compose

247. Docker Compose

248. Docker Compose

249. Docker Compose

250. Docker Compose

251. Docker Compose

252. Docker Compose

253. Docker Compose

254. Docker Compose

255. Docker Compose

256. Docker Compose

257. Docker Compose

258. Docker Compose

259. Docker Compose

260. Docker Compose

261. Docker Compose

262. Docker Compose

263. Docker Compose

264. Docker Compose

265. Docker Compose

266. Docker Compose

267. Docker Compose

268. Docker Compose

269. Docker Compose

270. Docker Compose

271. Docker Compose

272. Docker Compose

273. Docker Compose

274. Docker Compose

275. Docker Compose

276. Docker Compose

277. Docker Compose

278. Docker Compose

279. Docker Compose

280. Docker Compose

281. Docker Compose

282. Docker Compose

283. Docker Compose

284. Docker Compose

285. Docker Compose

286. Docker Compose

287. Docker Compose

288. Docker Compose

289. Docker Compose

290. Docker Compose

291. Docker Compose

292. Docker Compose

<

Today we will learn :

1. What are images
2. How to pull image
3. How to run a container using an image
4. Basic Commands

TIPS & TRICKS

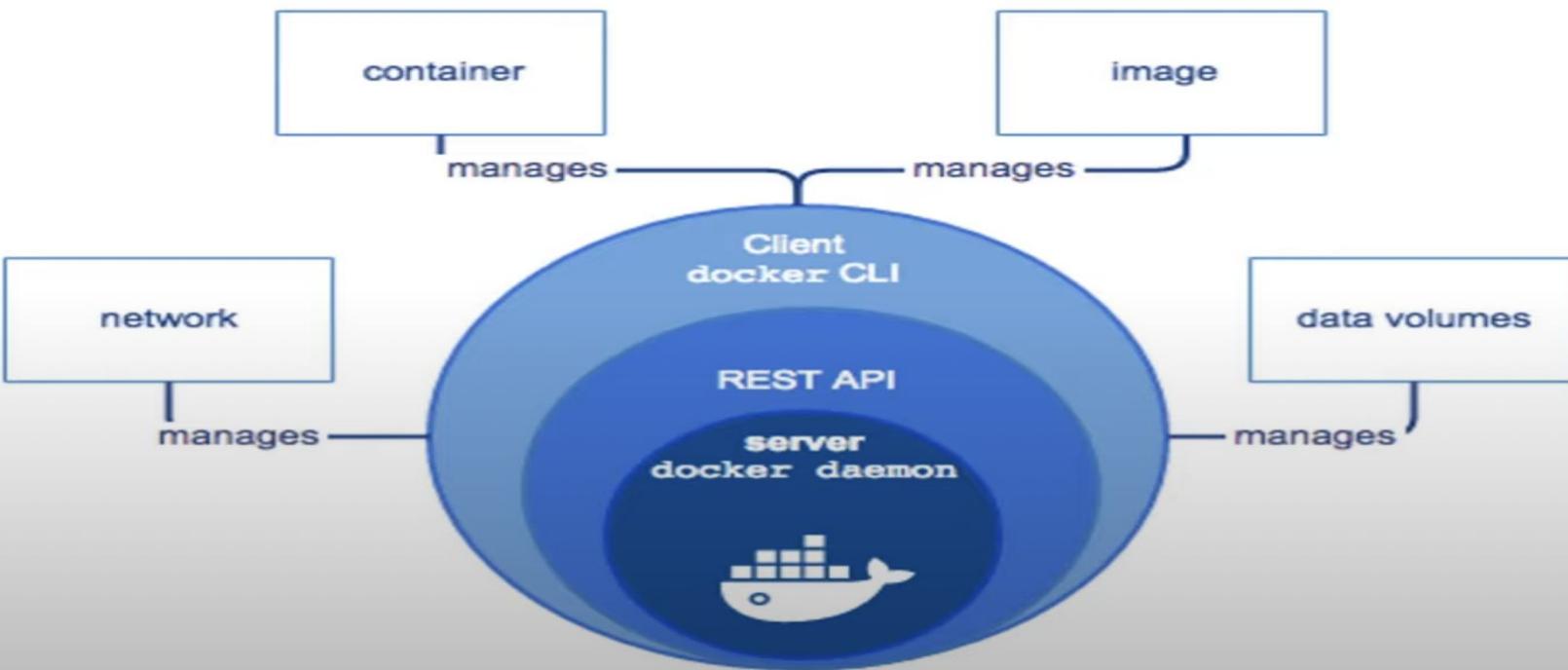
What are Images

Docker Images are templates used to create Docker containers
Container is a running instance of image

Where are Images Stored

Registries (e.g. docker hub)

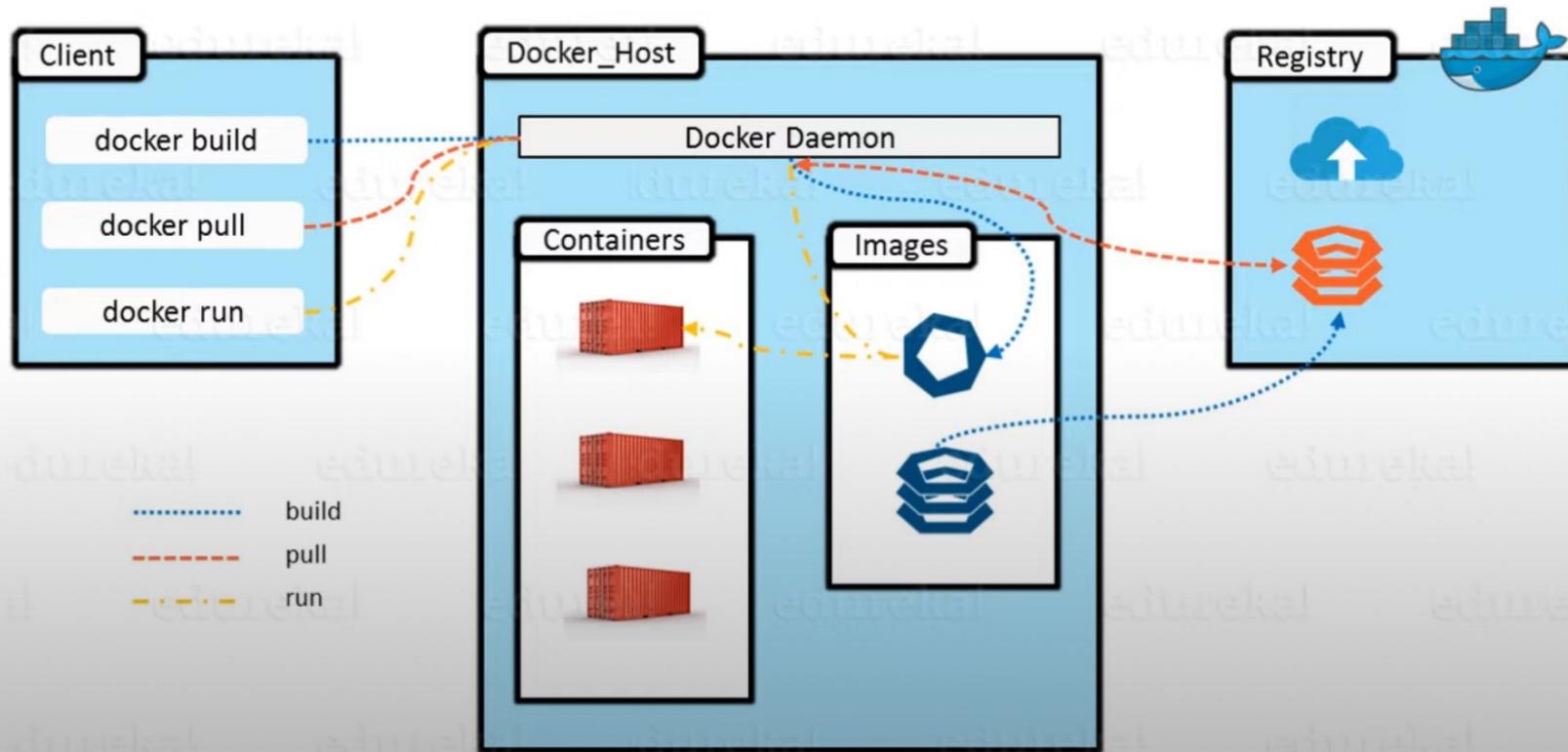
Docker Engine



1. Docker daemon which is used for the application interact with docker daemon
2. Docker cli is command line interface which is used for interacting with docker daemon
3. Docker client talk with docker daemon building ,distributing of docker containers
4. Client and daemon can run on the same system
- 5.

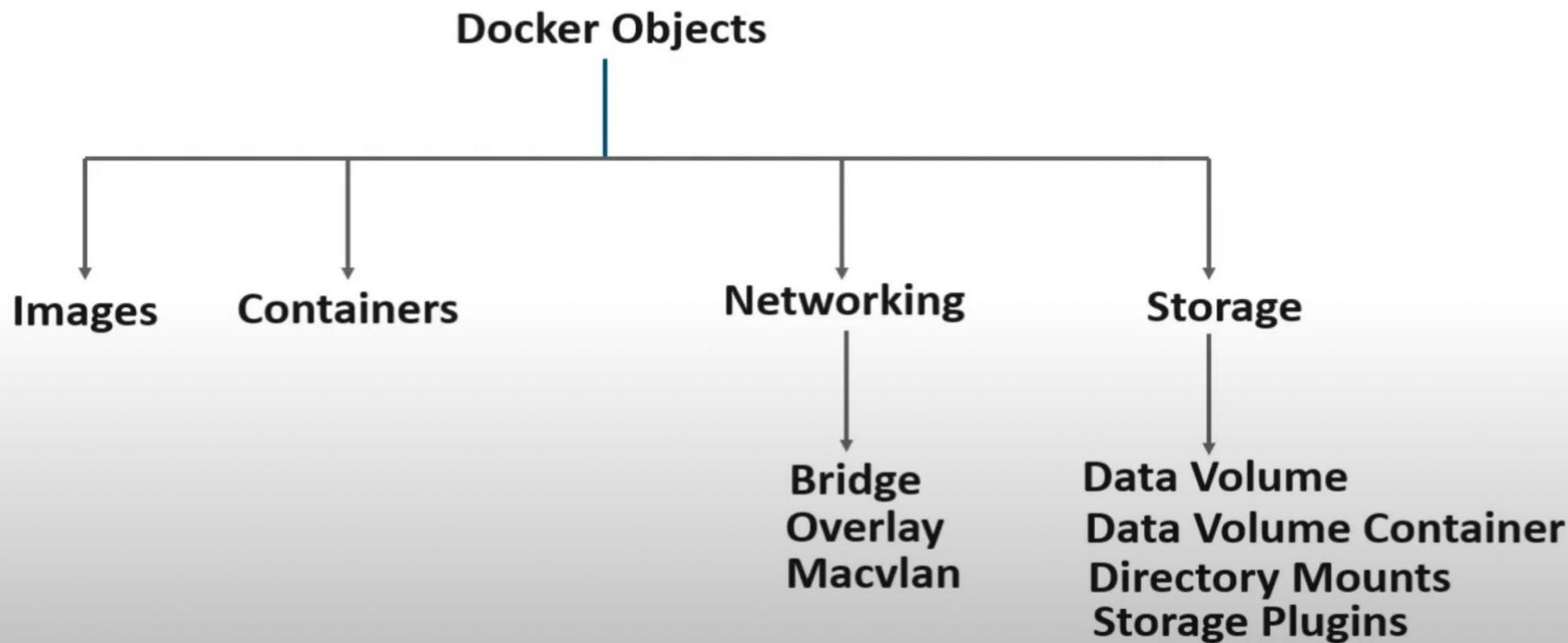
Docker Architecture

edureka!



1. Docker client can communicate with more than one daemon
2. Docker Host provide complete execute host
3. It contain network and storage,images
4. Daemon responsible for all container related actions
5. Docker daemon is pull and build image as requested by client

Docker Components



Docker Registry

Docker Registry

This page contains information about hosting your own registry using the [open source Docker Registry](#). For information about Docker Hub, which offers a hosted registry with additional features such as teams, organizations, web hooks, automated builds, etc, see [Docker Hub](#).

What it is

The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images. The Registry is open-source, under the permissive [Apache license](#). You can find the source code on [GitHub](#).

Why use it

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

- Containers are created they should be able to communicate with each other
-

Jenkins on Docker

5:49 AM Step by Step for...

Docker Volumes

Yesterday Step by Step f...

Docker Basic Command...

Yesterday Step by Step f...

How to create Dockerf...

Monday Docker Basic Co...

How to create Docker I...

Monday No additional text

Docker Containers

Monday What are Contai...

Docker Images
BEGINNERS

Today we will learn :

1. What are images
2. How to pull image
3. How to run a container using an image
4. Basic Commands

TIPS & TRICKS**What are Images**

Docker Images are templates used to create Docker containers
Container is a running instance of image

Where are Images Stored

Registries (e.g. docker hub)

1. File which contain what all things will be required to create container
2. Container is a running instance of the image
3. Images are stored in the docker hub
4. <https://github.com/docker/compose>

Docker Compose



Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see [the list of features](#).

Compose is great for development, testing, and staging environments, as well as CI workflows. You can learn more about each case in [Common Use Cases](#).

Using Compose is basically a three-step process.

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Lastly, run `docker-compose up` and Compose will start and run your entire app.

Docker Compose

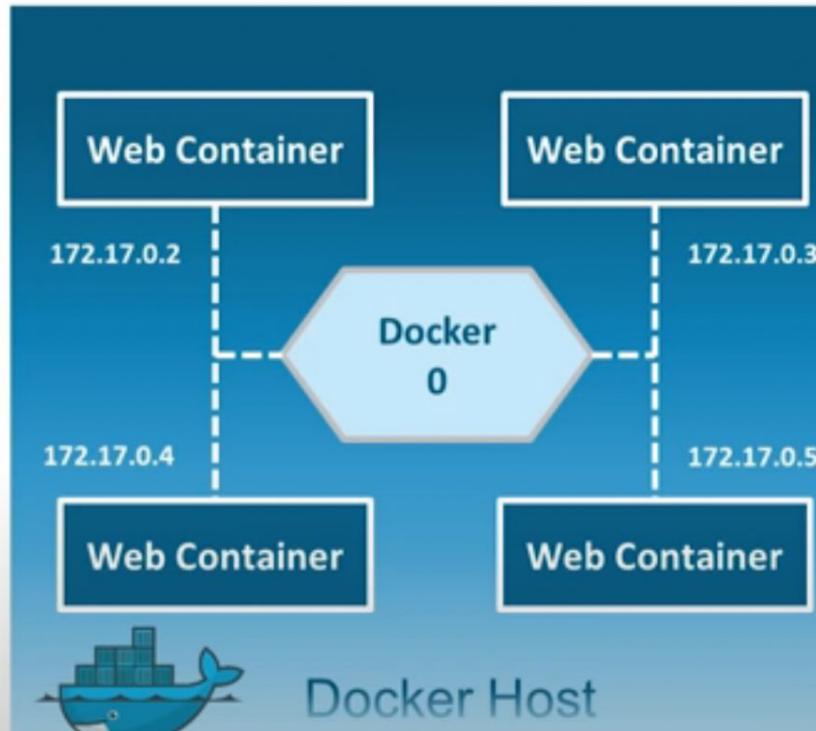
Docker Networking

Goals of Docker Networking



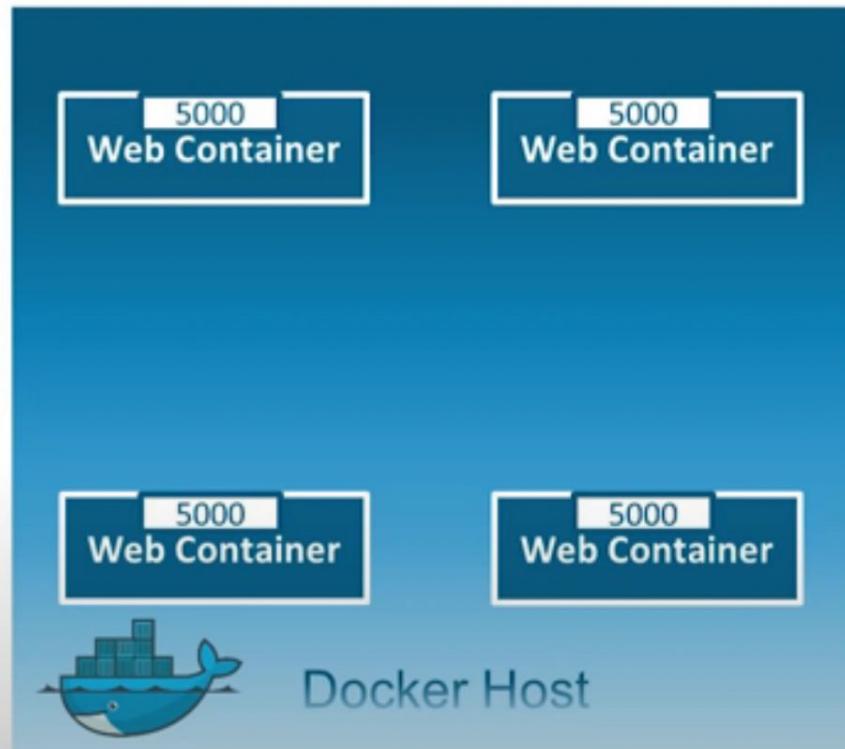
Network Drivers: Bridge

The default network driver. If you don't specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

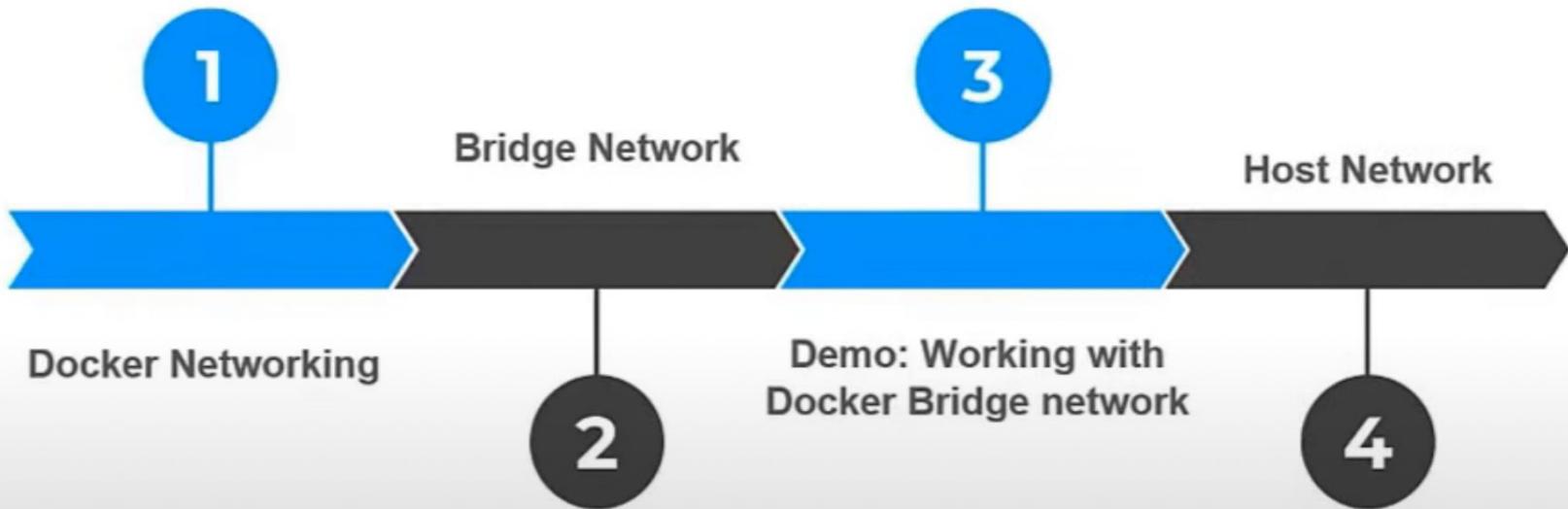


Network Drivers: Host

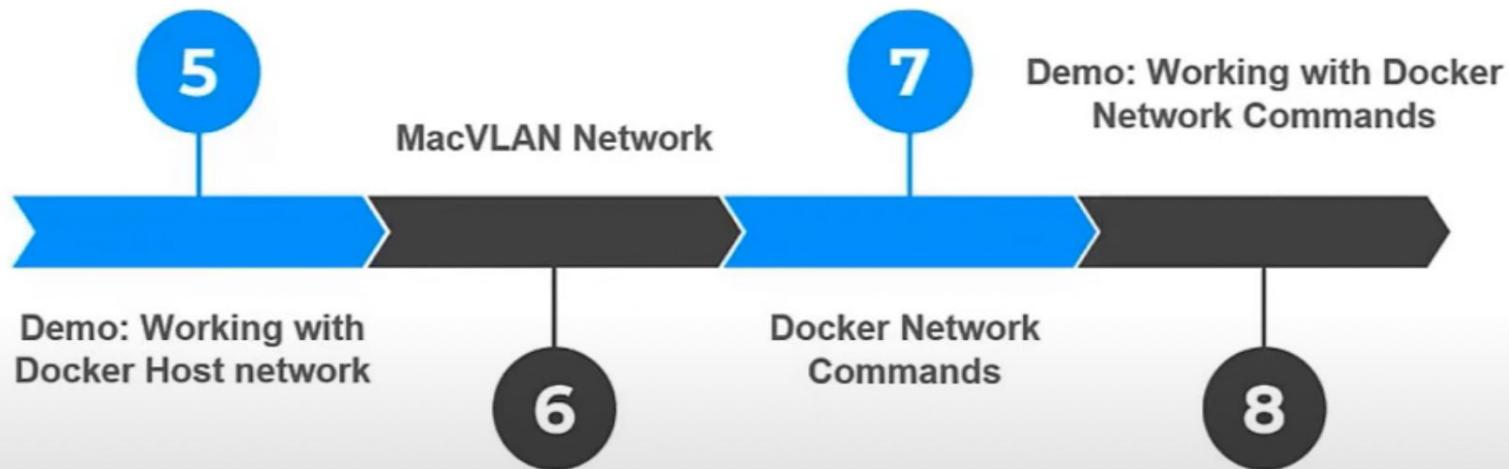
For standalone containers, removes network isolation between the container and the Docker host, and uses the host's networking directly.



Agenda



Agenda



```
sharathhanswadi@Chandras-MacBook-Pro getting-started % docker info
Client:
Context:    default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc., v0.8.2)
  compose: Docker Compose (Docker Inc., v2.5.0)
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
  scan: Docker Scan (Docker Inc., v0.17.0)

Server:
Containers: 5
Running: 3
Paused: 0
Stopped: 2
Images: 10
Server Version: 20.10.14
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
Logging Driver: json-file
Cgroup Driver: cgroupfs
Cgroup Version: 2
```

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
d9d67d745590	bridge	bridge	local
ec714e10e880	host	host	local
5ebf2db21d7d	minikube	bridge	local
b05bdc4ca458	none	null	local

- 1) Bridge
- 2) Host
- 3) Null
- 4) Macvlan
- 5) Overlay

1. Bridge will be always default network
2. Bridge is used when multiple docker containers are running on the same host machine
- 3.

What is Docker Networking?

Docker networking enables a user to link a Docker container to as many networks as he/she requires

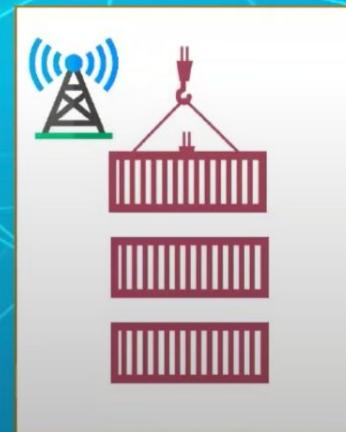


What is Docker Networking?

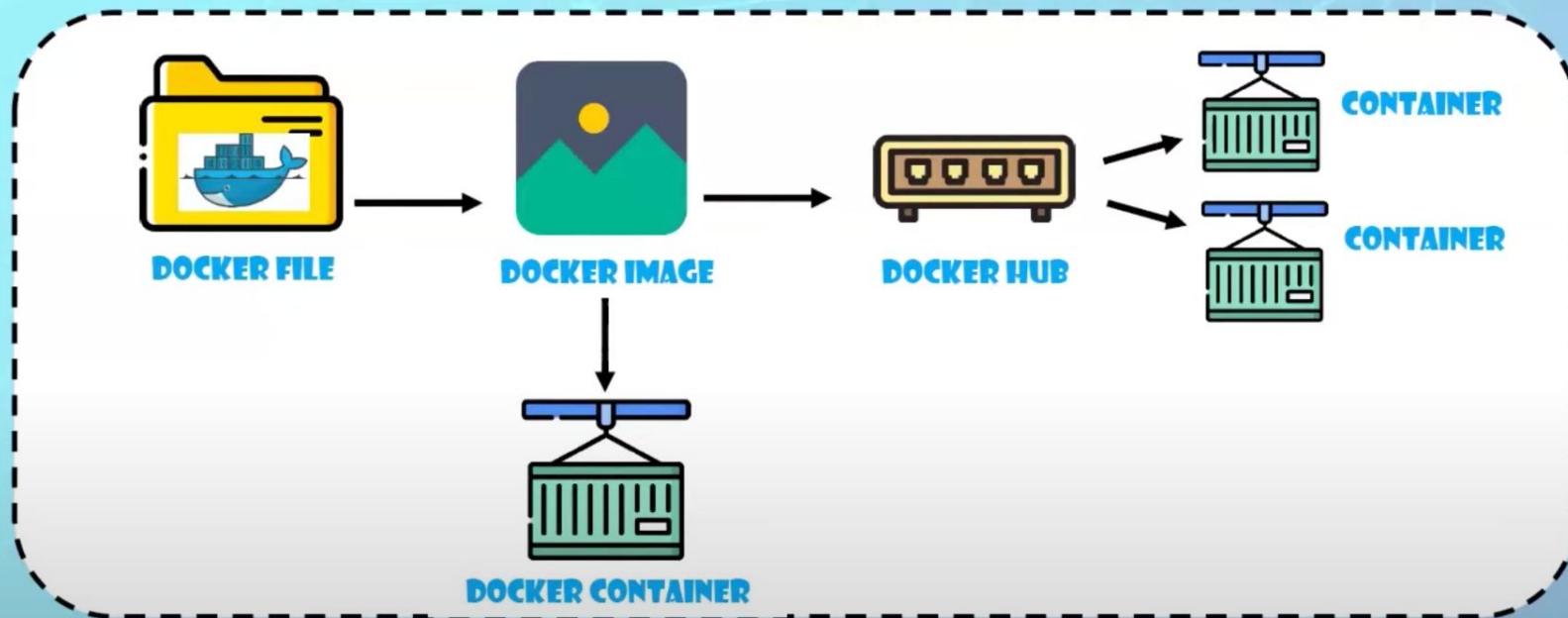
Docker networking enables a user to link a Docker container to as many networks as he/she requires

Docker Networks are used to provide complete isolation for Docker containers

Note: A user can add containers to more than one network



How Docker Networking Works?



Advantages of Docker Networking



Rapid Deployment



Portability



Better Efficiency



Faster Configuration

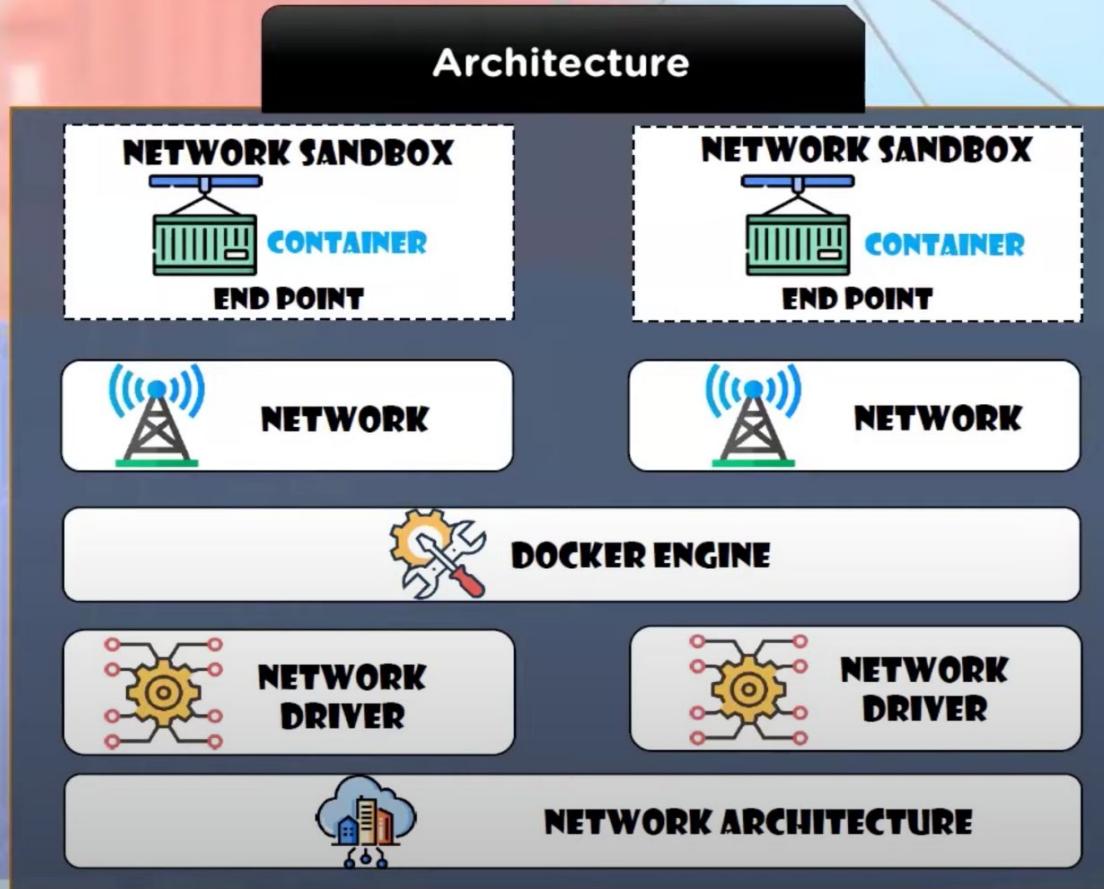


Scalability

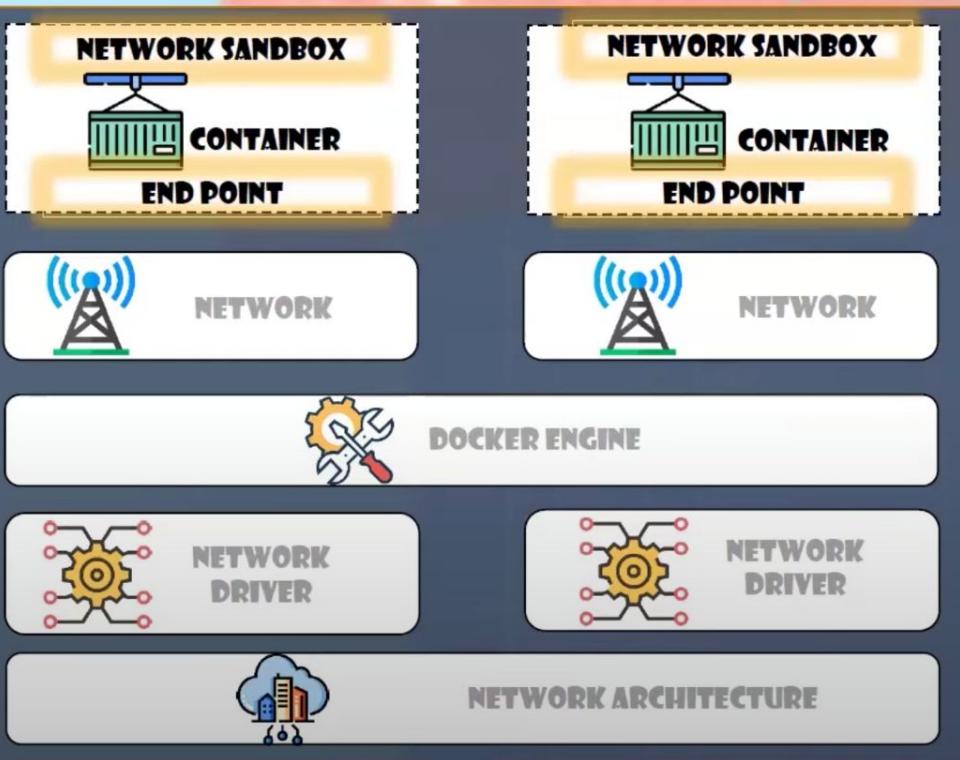


Security

Container Network Model



Container Network Model



- It is an isolated sandbox that holds the network configuration of containers
- Sandbox is created when a user requests to generate an endpoint on the network
- It can have several endpoints in a network, as it represents a container's network configuration such as IP-address, MAC-address, DNS etc.
- End point establishes the connectivity for container services (within a network) with other services

Network Drivers

The network drivers used in Docker are below:



BRIDGE



HOST



NONE



OVERLAY



MACVLAN



BRIDGE

- It is a private default network created on the host
- Containers linked to this network have an internal IP address through which they communicate with each other easily
- The Docker server (daemon) creates virtual ethernet bridge **docker0** that performs the operation by automatically delivering packets among various network interfaces



MACVLAN



BRIDGE



- It is a public network
- It utilizes the host's IP address and TCP port space in order to display the services running inside the container

Network Drivers

- In this network driver, the Docker containers will neither have any access to external networks nor will it be able to communicate with other containers



NONE



OVERLAY



MACVLAN

Network Drivers

- This is utilized for creating an internal private network to the Docker nodes in the Docker swarm cluster



BRI



OVERLAY

Demo - Networking with standalone containers using the default bridge network

```
root@ip-172-31-75-99:~# docker version
Client:
 Version:          19.03.8
 API version:      1.40
 Go version:       go1.13.8
 Git commit:       afacb8b7f0
 Built:            Tue Jun 23 22:26:12 2020
 OS/Arch:          linux/amd64
 Experimental:    false
```

```
Server:
Engine:
 Version:          19.03.8
 API version:      1.40 (minimum version 1.12)
 Go version:       go1.13.8
 Git commit:       afacb8b7f0
 Built:            Thu Jun 18 08:26:54 2020
 OS/Arch:          linux/amd64
 Experimental:    false
```

```
containerd:
 Version:          1.3.3-0ubuntu2
 GitCommit:
```

```
runc:
 Version:          spec: 1.0.1-dev
 GitCommit:
```

```
docker-init:
 Version:          0.18.0
 GitCommit:
```

```
root@ip-172-31-75-99:~# docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
e65c4305d6a1   bridge    bridge      local
729ac5c85118   host      host       local
8a3bfa625405   none      null       local
root@ip-172-31-75-99:~#
```

```
root@ip-172-31-75-99:~# docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
e65c4305d6a1	bridge	bridge	local
729ac5c85118	host	host	local
8a3bfa625405	none	null	local

```
root@ip-172-31-75-99:~# docker run -dit --name alpine1 alpine ash
```

```
Unable to find image 'alpine:latest' locally
```

```
latest: Pulling from library/alpine
```

```
df20fa9351a1: Pull complete
```

```
Digest: sha256:185518070891758909c9f839cf4ca393ee977ac378609f700f60a771a2dfe32
```

```
Status: Downloaded newer image for alpine:latest
```

```
c07ebd170163365e62f008eb957cc8267ca01824d9b9f977007654936e6153bd
```

```
root@ip-172-31-75-99:~#
```

```
root@ip-172-31-75-99:~# docker run -dit --name alpine2 alpine ash
```

```
4ac24fac22f714f44dad8d8f96066840479898a5dc1ca4e78a68c239198085e8
```

```
root@ip-172-31-75-99:~#
```

IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
alpine	"ash"	10 seconds ago	Up 9 seconds		alpine2
alpine	"ash"	21 seconds ago	Up 21 seconds		alpine1

```
· docker network inspect bridge
```

```

"EnableIPv6": false,
"IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
        {
            "Subnet": "172.17.0.0/16"
        }
    ]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {
    "4ac24fac22f714f44dad8d8f96066840479898a5dc1ca4e78a68c239198085e8": {
        "Name": "alpine2",
        "EndpointID": "f8c015a0c13ff69fc90013c3e31a608f620af99303056a0cf0d4d1d6bb2e859d",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
    },
    "c07ebd170163365e62f008eb957cc8267ca01824d9b9f977007654936e6153bd": {
        "Name": "alpine1",
        "EndpointID": "bdcf411f60d6fc8212fed717a16413f114dd0b0fa2f15c9dd757d9ae93daaed",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    }
},
"Options": {
    "com.docker.network.bridge.default_bridge": "true",
    "com.docker.network.bridge.enable_icc": "true",
    "com.docker.network.bridge.enable_ip_masquerade": "true",
    "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
    "com.docker.network.bridge.name": "docker0",
    "com.docker.network.driver.mtu": "1500"
},
"Labels": {}

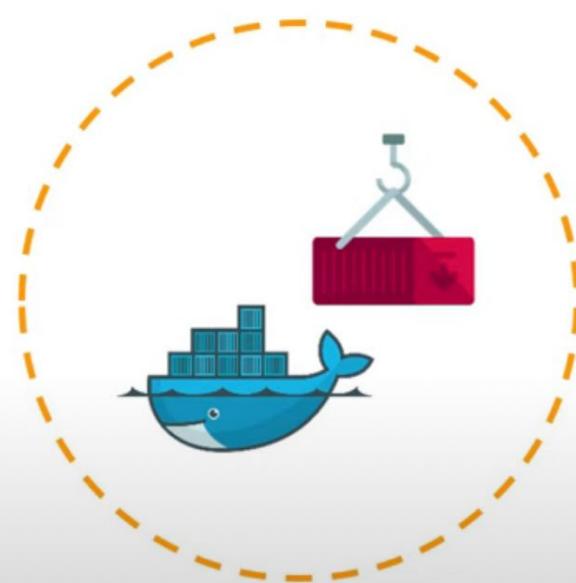
```

p-172-31-75-99:~# docker network ls

K	ID	NAME	DRIVER	SCOPE
	05d6a1	bridge	bridge	local

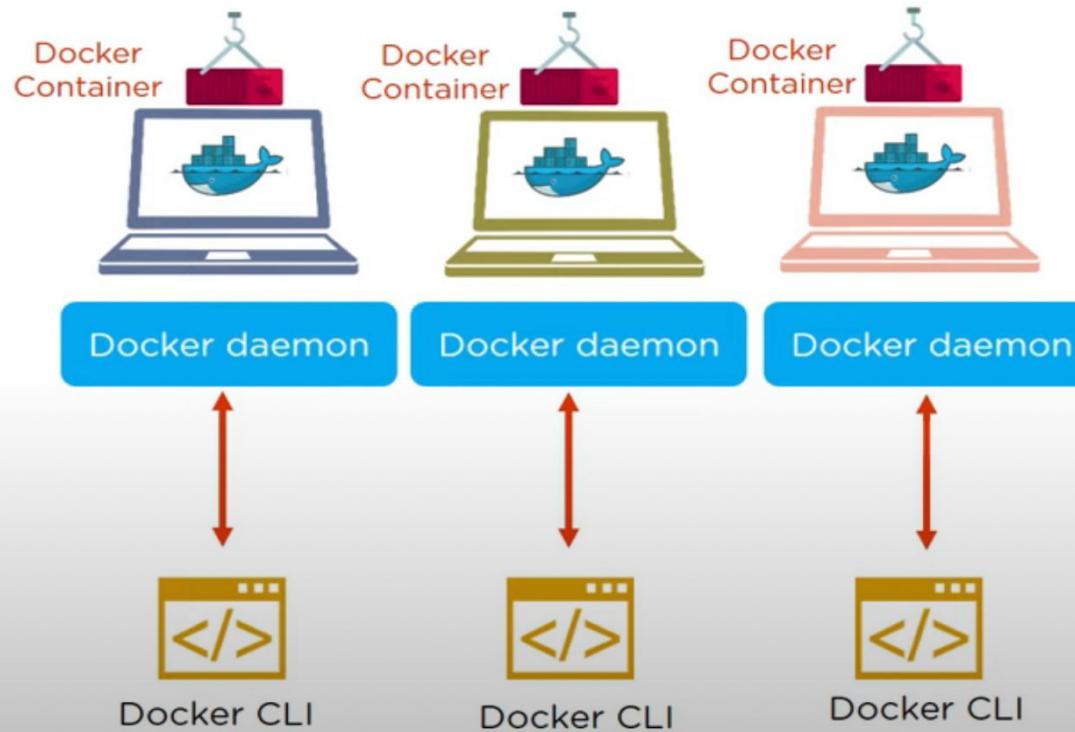
What is Docker Swarm?

- Docker Swarm is a service which allows users to create and manage a cluster of Docker nodes and schedule containers



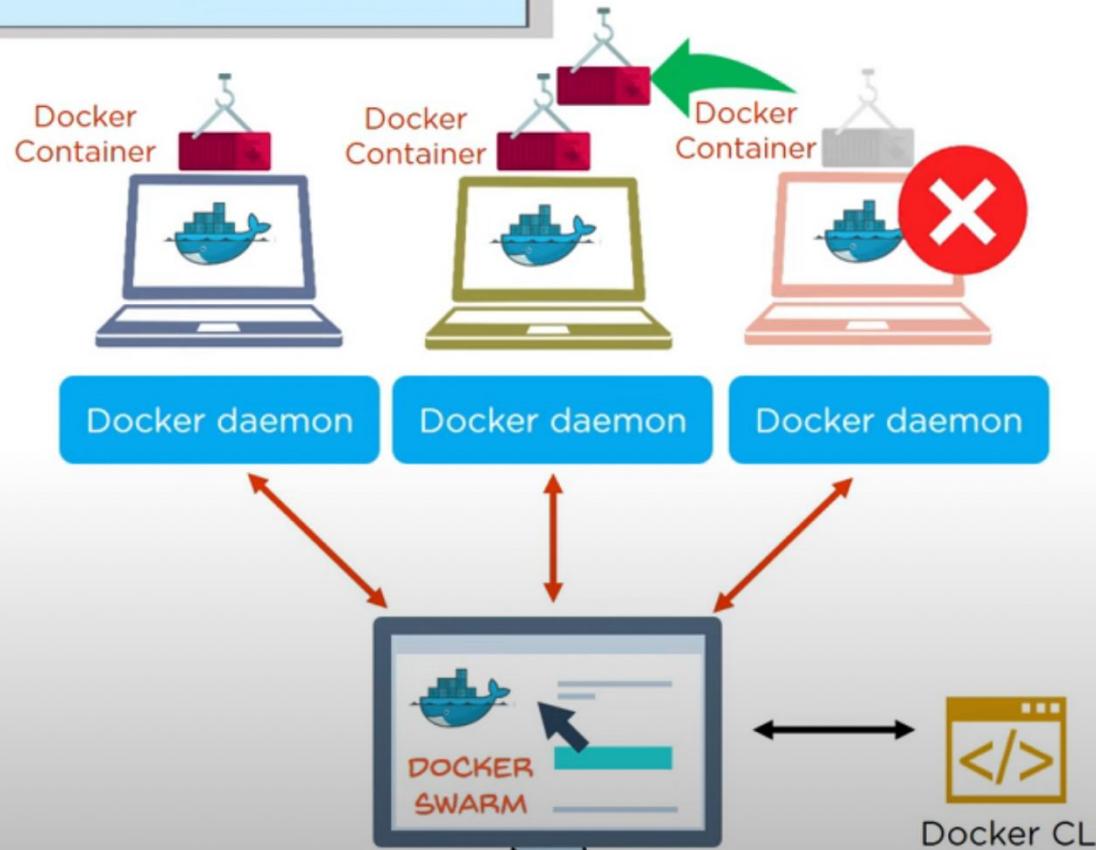
What is Docker Swarm?

With Docker



What is Docker Swarm?

With Docker Swarm on fault tolerance

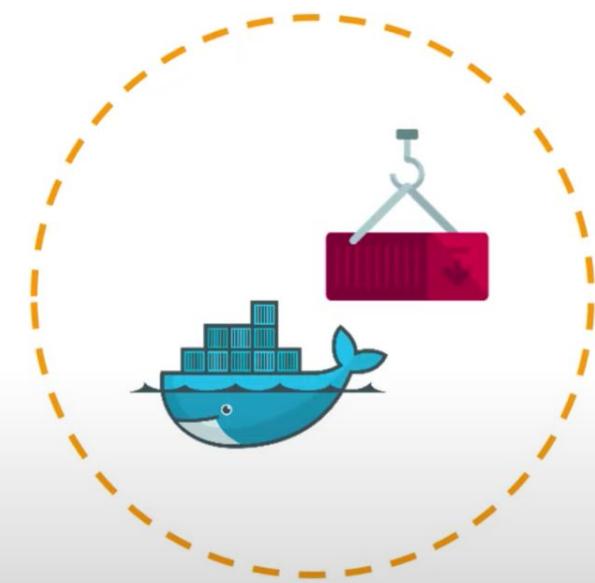


DOCKER SWARM CAN RESCHEDULE CONTAINERS ON NODE FAILURES



What is Docker Swarm?

- Docker Swarm is a service which allows users to create and manage a cluster of Docker nodes and schedule containers



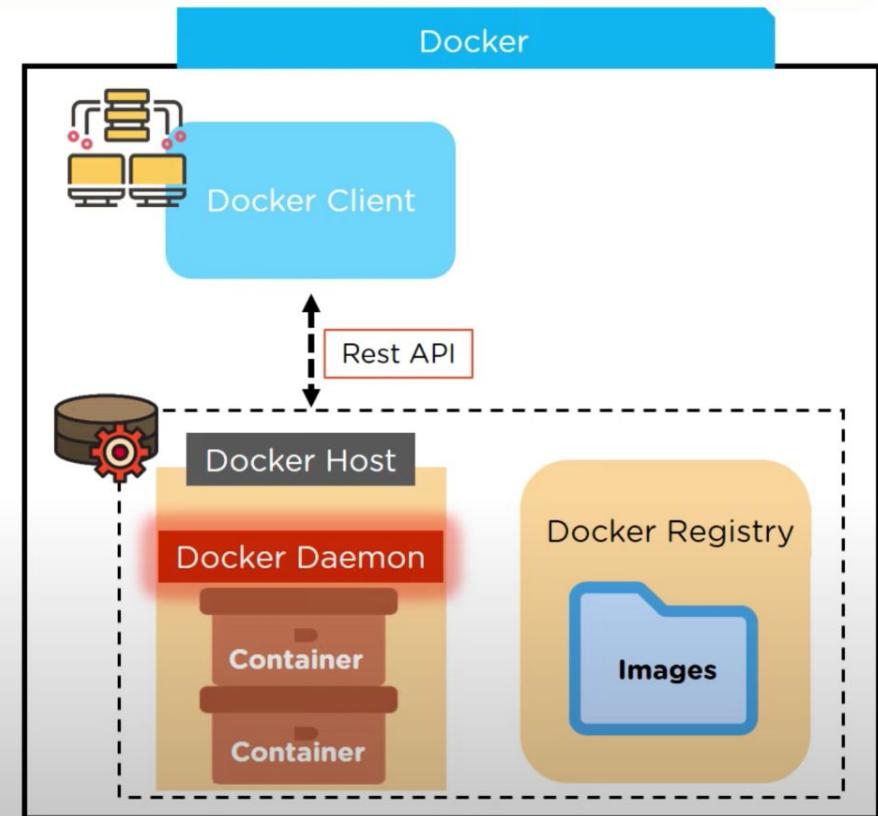
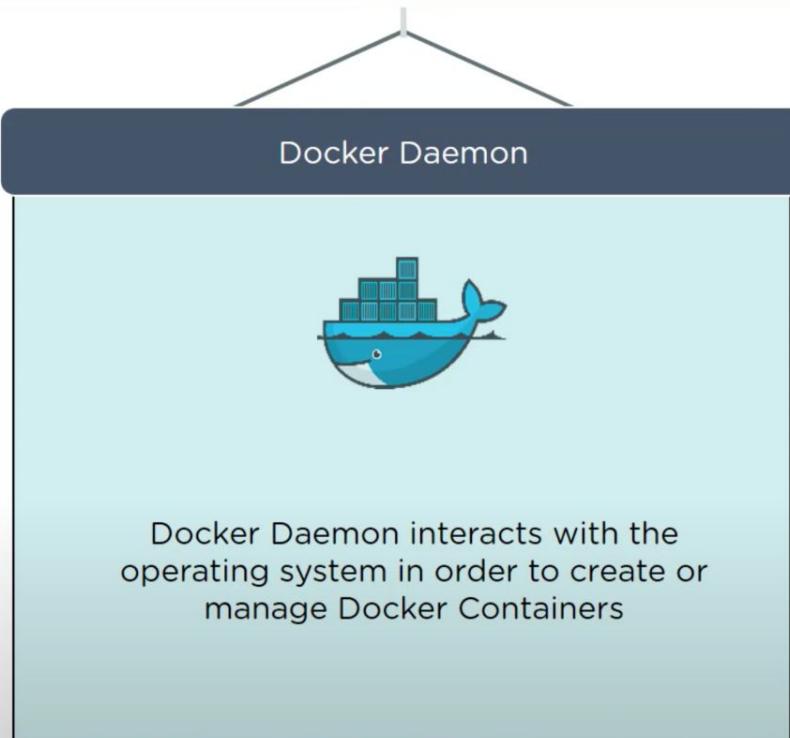
What is Docker Swarm?



- Docker Swarm is a service which allows users to create and manage a cluster of Docker nodes and schedule containers
- Each node of a Docker Swarm is a Docker daemon and all Docker daemons interact using the Docker API
- Here, services can be deployed and accessed by nodes of same cluster



What is Docker Swarm?



Features of Docker Swarm



Decentralized
access

High security

Auto load
balancing

High scalability

Roll-back a task

Docker Swarm



- In Swarm, containers are launched using services
- A service is a group of containers of the same image
- Services enables to scale your application
- Before you can deploy a service in Docker Swarm, you must have at least one node deployed
- There are two types of nodes in Docker Swarm

Manager node



Worker node



Docker Swarm



- In Swarm, containers are launched using services
- A service is a group of containers of the same image
- Services enables to scale your application
- Before you can deploy a service in Docker Swarm, you must have at least one node deployed
- There are two types of nodes in Docker Swarm

Manager node



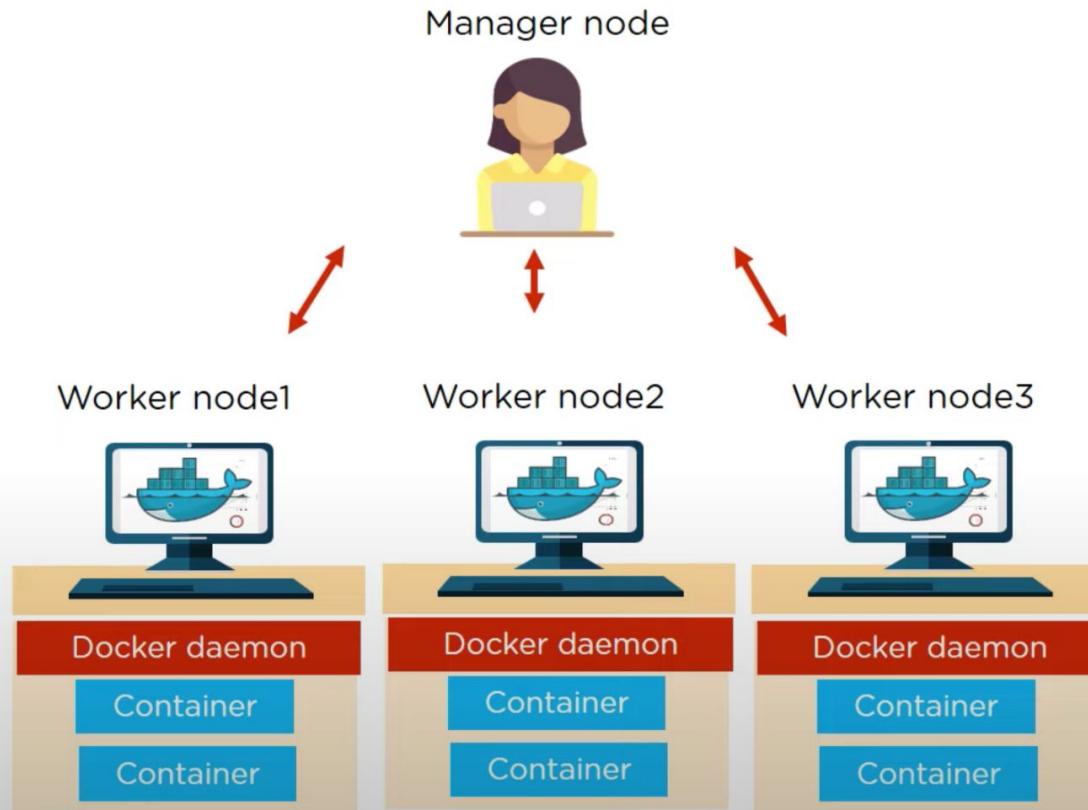
Manager node
maintains cluster
management tasks

Worker node



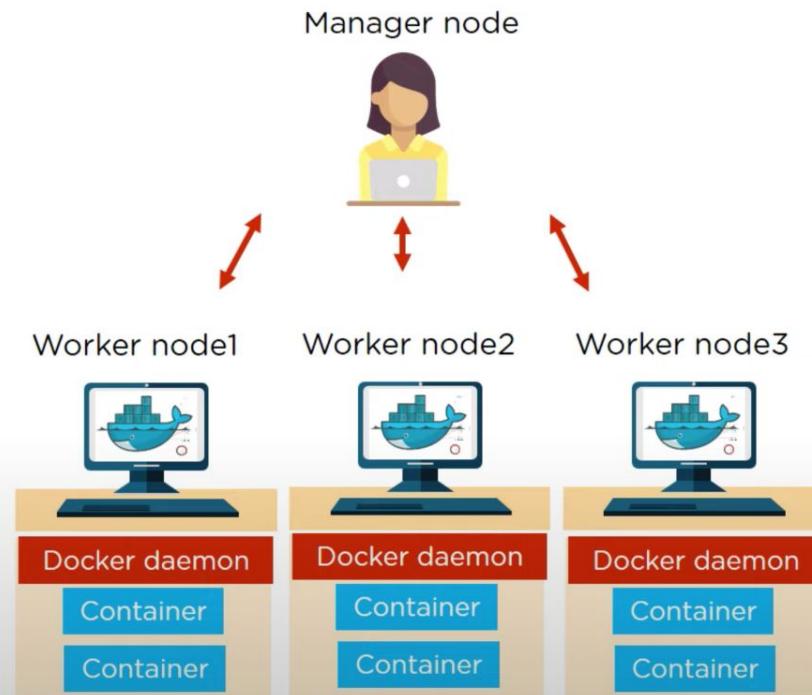
Worker nodes receive
and execute tasks from
manager node

Architecture of Docker Swarm



How does Docker Swarm work?

- Manager node knows the status of all the worker nodes in a cluster
- Worker nodes accept tasks sent from manager node
- Every worker node has an agent, which reports on the state of the node's tasks to the manager



How does Docker Swarm work?

- Manager node knows the status of all the worker nodes in a cluster
- Worker nodes accept tasks sent from manager node
- Every worker node has as an agent, which reports on the state of the node's tasks to the manager

THIS WAY, THE
MANAGER NODE CAN
MAINTAIN THE
DESIRED STATE OF
THE CLUSTER



Worker node1 Worker node2

Worker node3



Docker daemon

Docker daemon

Docker daemon

Container

Container

Container

Container

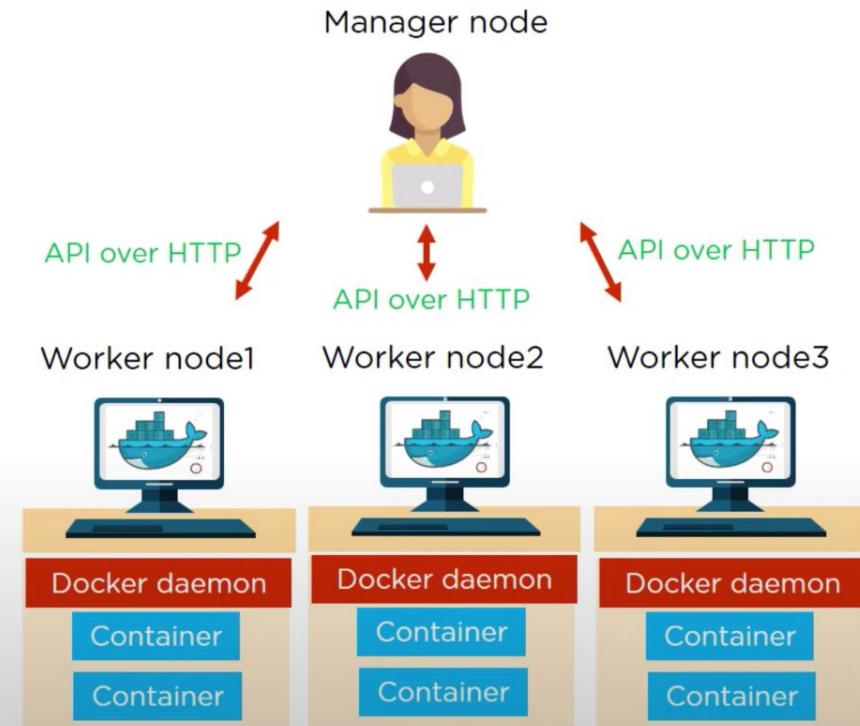
Container

Container

How does Docker Swarm work?



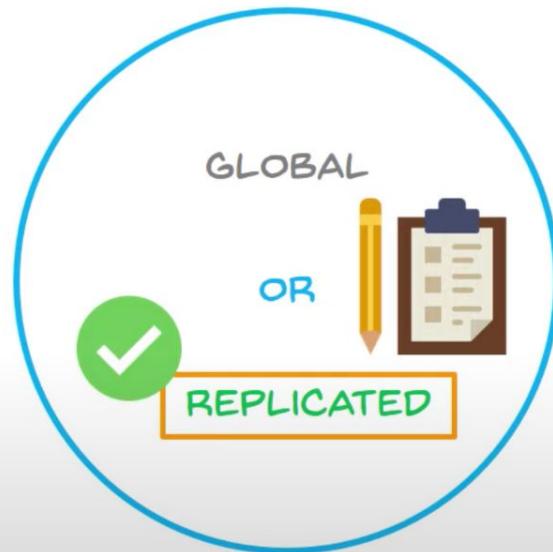
- Manager node knows the status of all the worker nodes in a cluster
- Worker nodes accept tasks sent from manager node
- Every worker node has an agent, which reports on the state of the node's tasks to the manager
- The worker nodes communicate with the manager node using API over HTTP



How does Docker Swarm work?



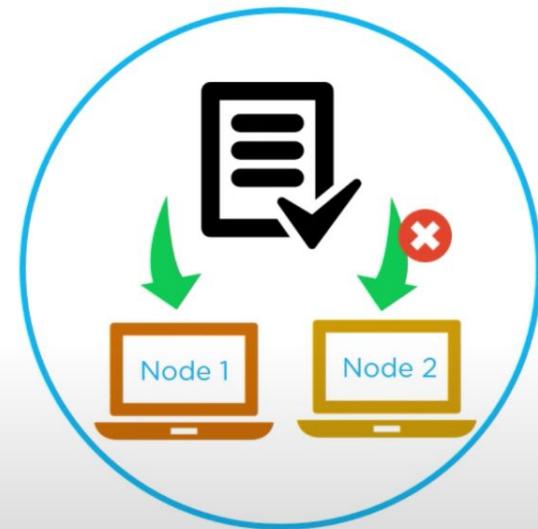
- In Docker Swarm, services can be deployed and accessed by any node of same cluster
- While creating a service, a user has to specify which container image to use
- Here, a service is either global or replicated
- A global service will run on every Swarm node
- In a replicated service, the manager node distributes tasks to worker nodes



How does Docker Swarm work?



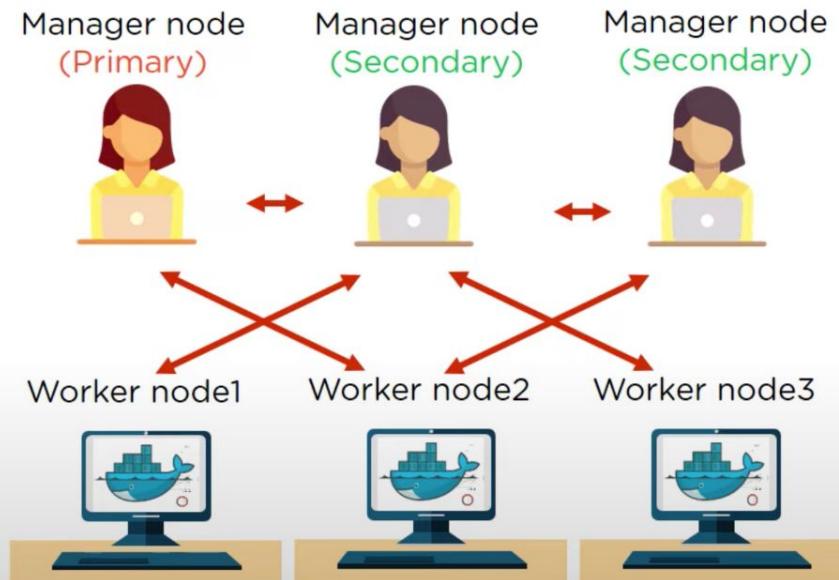
- A service is a description of a task or the state, whereas a task does the work
- Docker enables a user to create services, which can start tasks
- When a task is assigned to a node, it cannot be assigned to another node



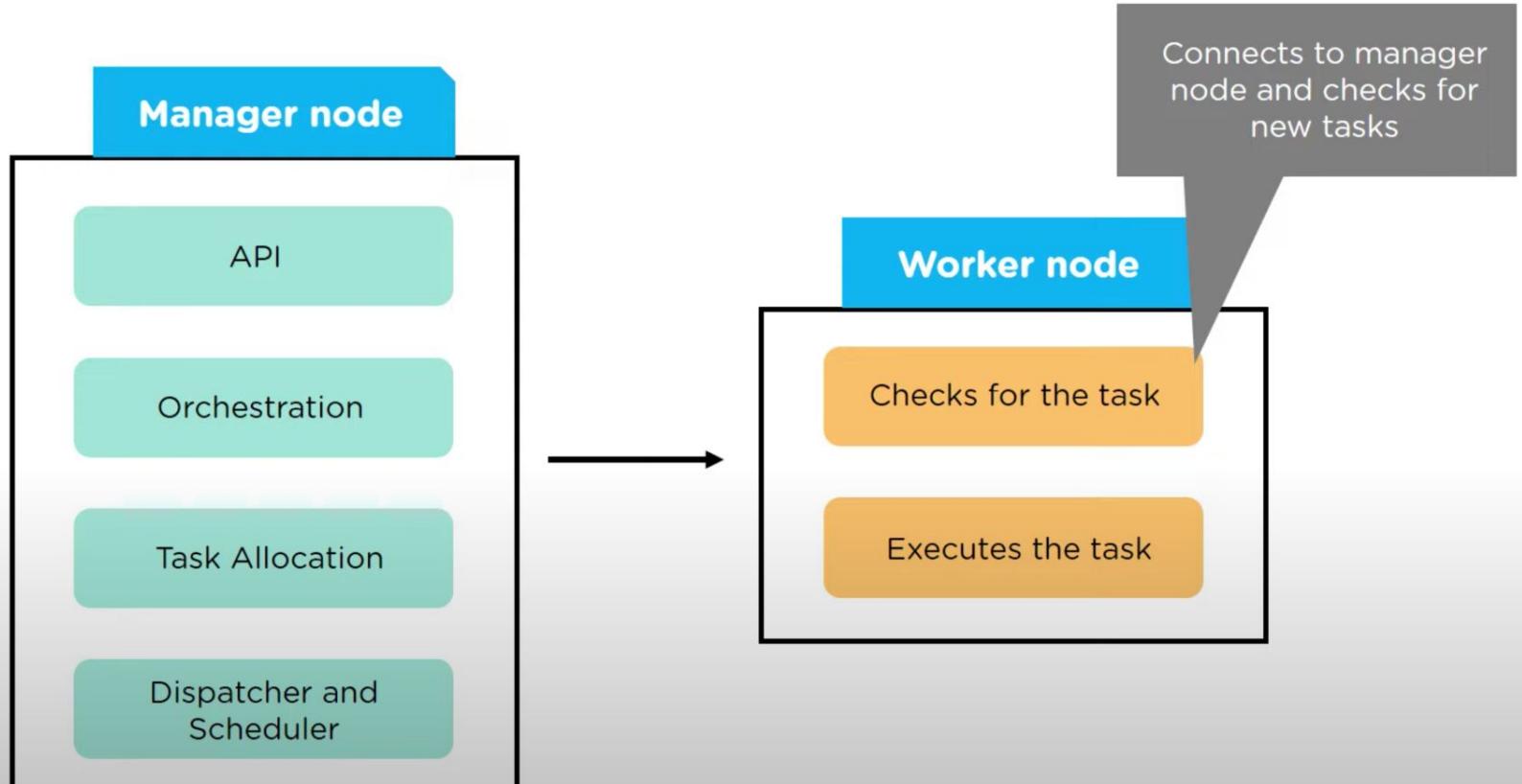
How does Docker Swarm work?

Did you Know?

It is possible to have multiple manager nodes on Swarm, but there will be only one primary manager node, which gets elected by the other Manager nodes



Recap



```
simpliworker2@manager-virtualbox:~$ sudo docker swarm init --advertise-addr 192.168.2.151
[sudo] password for simpliworker2:
Swarm initialized: current node (5awua4fx1pd3pu24s7nda4s2s) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-5g257rwdimmk84r4s3d7a29y3f9mmzbx6984fdcfozkfjy81vk-bvmb8bp5qcv7ak0kjaej4wjrp 192.168.2.151:2
377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
simpliworker2@manager-virtualbox:~$
```

```
simpliworker2@manager-virtualbox:~$ sudo docker swarm init --advertise-addr 192.168.2.151
[sudo] password for simpliworker2:
Swarm initialized: current node (5awua4fx1pd3pu24s7nda4s2s) is now a manager.
```

To add a worker to this swarm, run the following command:

```
377 docker swarm join --token SWMTKN-1-5g257rwdimmk84r4s3d7a29y3f9mmzbx6984fdcfozkfjy81vk-bvmb8bp5qcv7ak0kjaej4wjrp 192.168.2.151:2
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
simpliworker2@manager-virtualbox:~$ █
```

```
simpliworker2@manager-virtualbox:~$ sudo docker swarm init --advertise-addr 192.168.2.151
[sudo] password for simpliworker2:
Swarm initialized: current node (5awua4fx1pd3pu24s7nda4s2s) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-5g257rwdimmk84r4s3d7a29y3f9mmzbx6984fdcfozkfjy81vk-bvmb8bp5qcv7ak0kjaej4wjrp 192.168.2.151:377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
simpliworker2@manager-virtualbox:~$ sudo docker swarm node ls
```

Usage: docker swarm COMMAND

Manage Swarm

Options:

```
Commands:
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm
```

Run 'docker swarm COMMAND --help' for more information on a command.

```
simpliworker2@manager-virtualbox:~$ cl
```

```
sudo docker service create --name helloworld alpine ping docker.com
```

```
simpliworker2@manager-virtualbox:~$ sudo docker node ls
[sudo] password for simpliworker2:
ID                  HOSTNAME          STATUS        AVAILABILITY      MANAGER STATUS
5awua4fx1pd3pu24s7nda4s2s *    manager-virtualbox  Ready       Active           Leader
o9j2ljm0efmrh9m8j5ib339yk     worker-virtualbox  Ready       Active
simpliworker2@manager-virtualbox:~$ sudo docker service create --name helloworld alpine ping docker.com
ti0li83aculuijx19wvk42nkj
overall progress: 1 out of 1 tasks
1/1: running    [=====>]
verify: Waiting 4 seconds to verify that tasks are stable...
```

```
sudo docker service ls
ID      NAME      MODE      REPLICAS  IMAGE      PORTS
3ki6e2o5sug9  helloworld  replicated  1/1      alpine:latest
```

File Edit View Search Terminal Help

simpliworker2@manager-virtualbox:~\$ sudo docker node ls
[sudo] password for simpliworker2: █

I

File Edit View Search Terminal Help

simpliworker2@worker-virtualbox:~\$ sudo docker swarm leave --force
Node left the swarm.
simpliworker2@worker-virtualbox:~\$

```
sudo docker service create --name helloworld alpine ping docker.com  
docker service scale my_web=3
```

File Edit View Search Terminal Help

```
simpliworker2@manager-virtualbox:~$ sudo docker node ls
[sudo] password for simpliworker2:
ID                      HOSTNAME      STATUS      AVAILABILITY      MANAGER STATUS
awua4fx1pd3pu24s7nda4s2s *  manager-virtualbox  Ready      Active          Leader
9j2ljm0efmrh9m8j5ib339yk   worker-virtualbox  Ready      Active

simpliworker2@manager-virtualbox:~$ sudo docker service create --name helloworld alpine ping docker.com
i0li83aculuijx19wvk42nkj
overall progress: 1 out of 1 tasks
1/1: running  [=====
verify: Service converged

simpliworker2@manager-virtualbox:~$ sudo docker service ls
ID          NAME      MODE      REPLICAS      IMAGE      PORTS
i0li83aculu  helloworld  replicated  1/1          alpine:latest

simpliworker2@manager-virtualbox:~$ sudo docker ps
CONTAINER ID        IMAGE           COMMAND      CREATED      STATUS      PORTS      NAMES
34a4ed0e60f        alpine:latest    "ping docker.com"  About a minute ago  Up About a minute
1.zedorydklfotvkt24uijjrl08

simpliworker2@manager-virtualbox:~$ █
```

