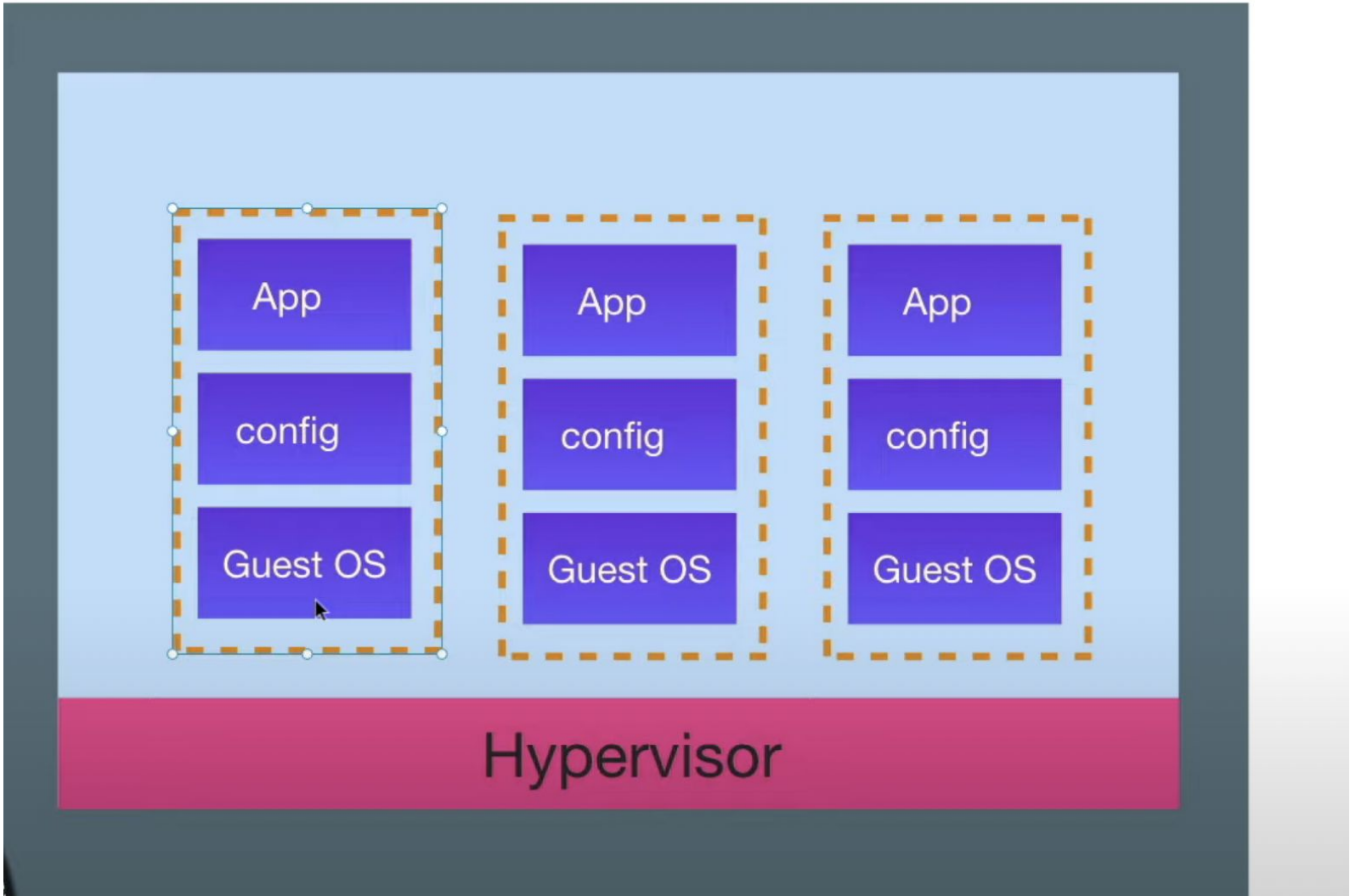
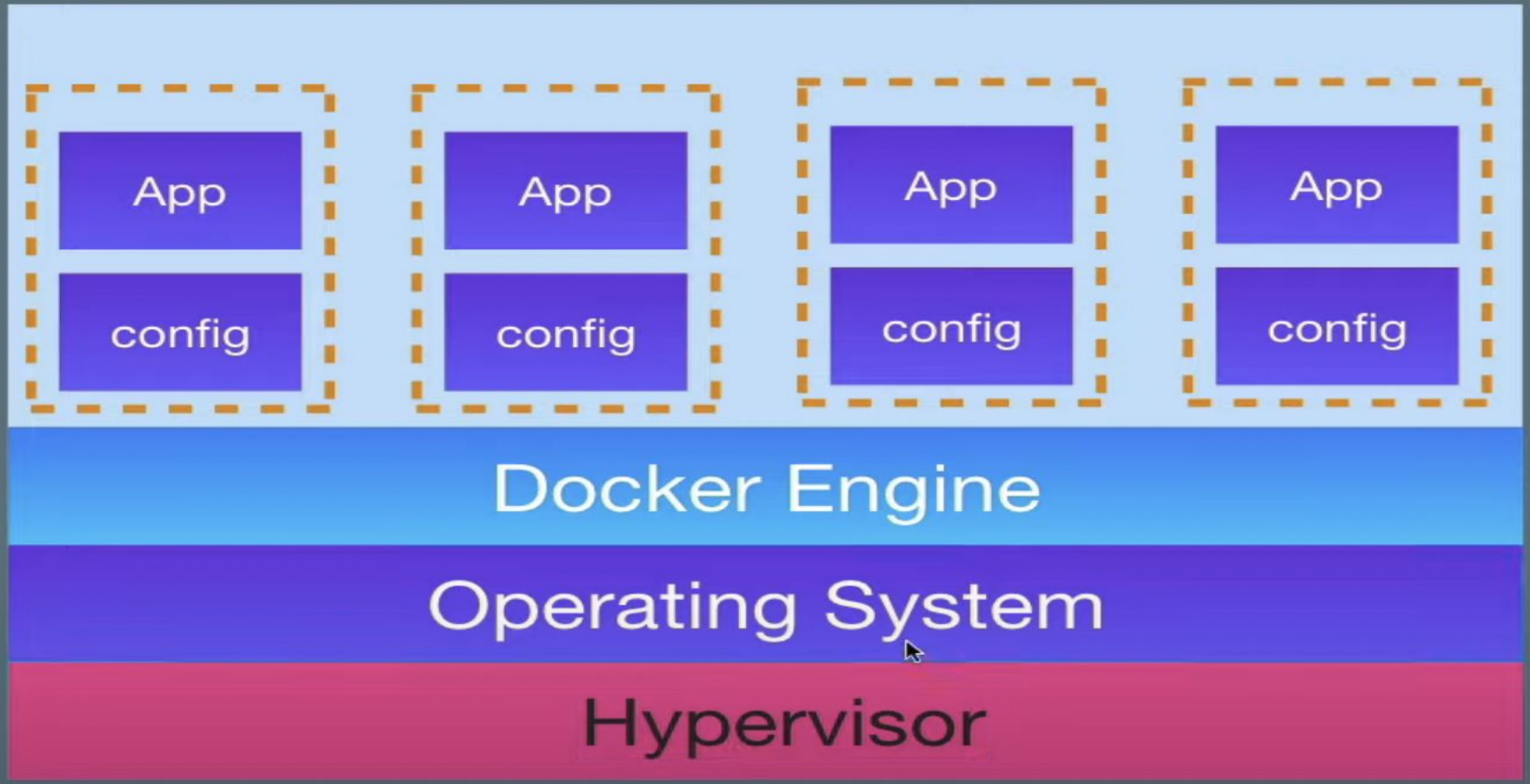
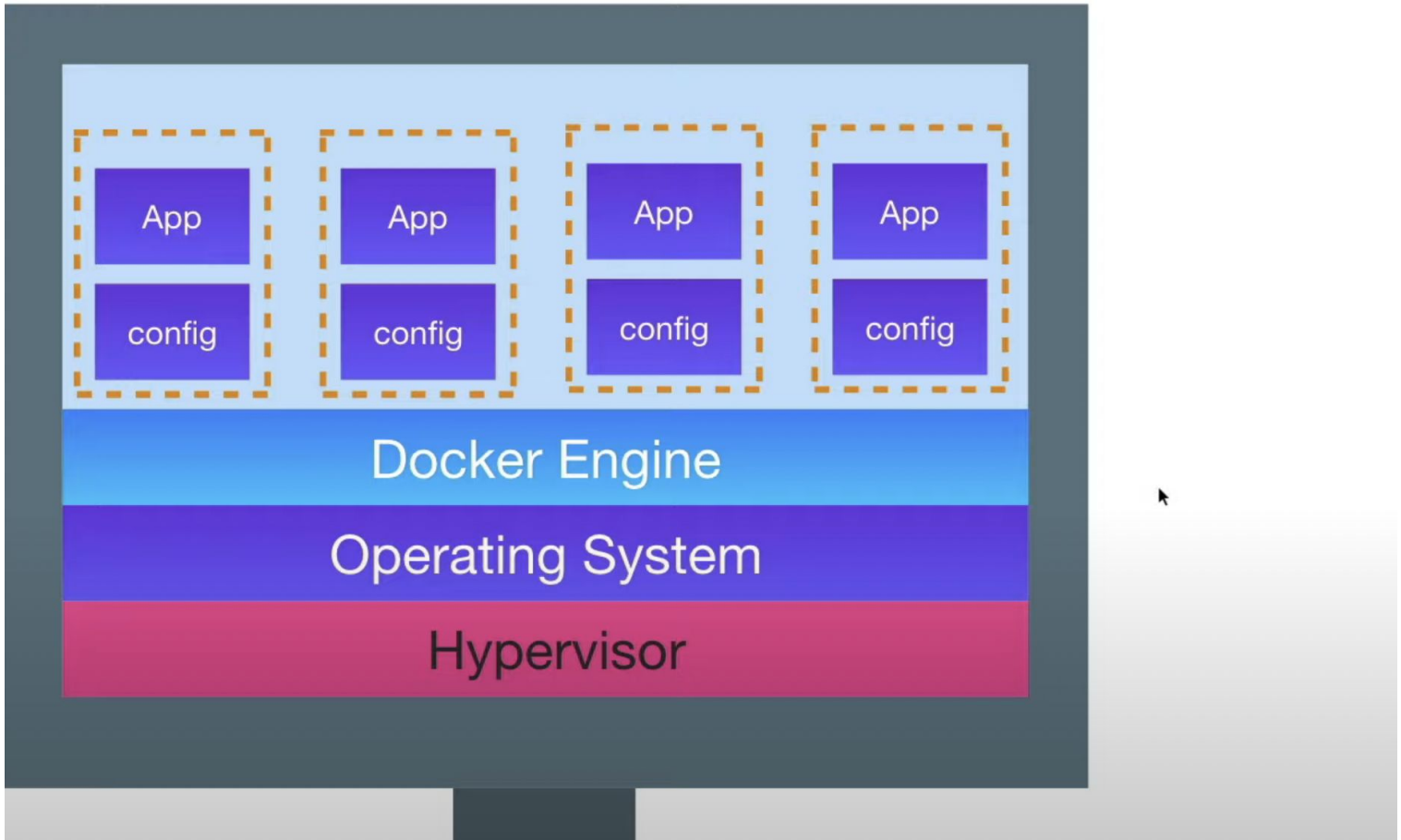
Containers vs VMs


App
config
Guest OS
App
config
Guest OS
App
config
Guest OS
Hypervisor

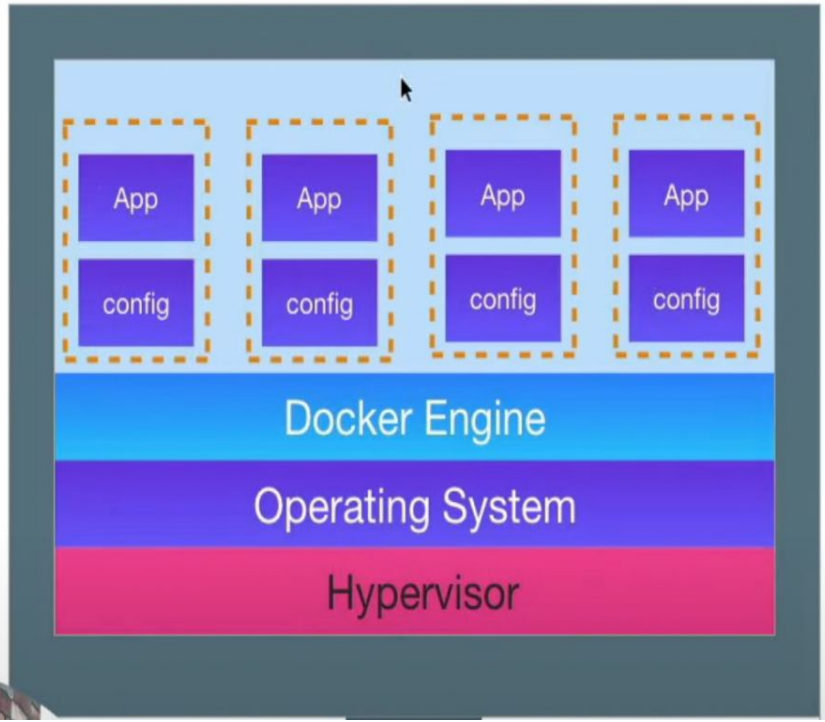
1) Its resource hungry and consume lot of resources
2) The guest os can be ubuntu its wastage of many resources
3)


App
config
App
config
App
config
App
config
Docker Engine
Operating System
Hypervisor

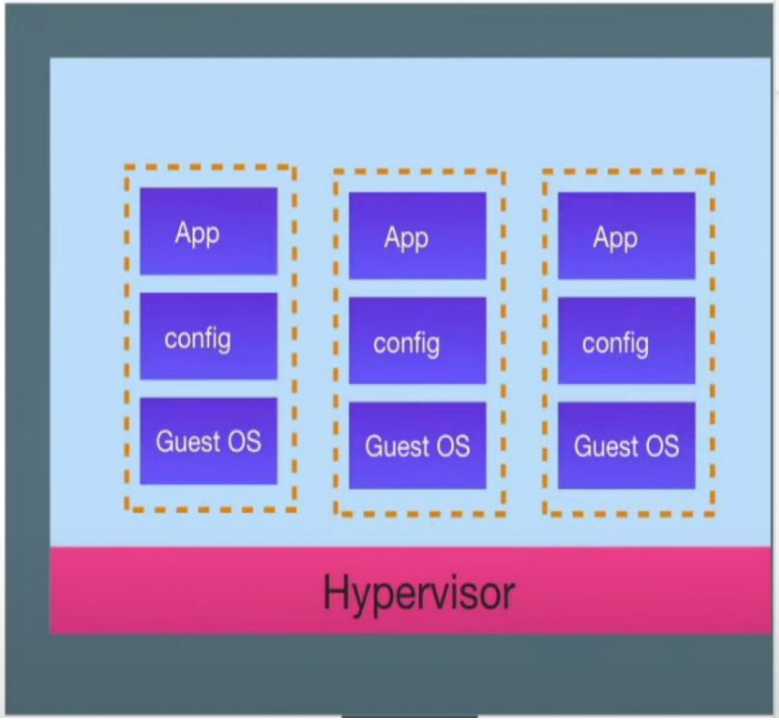
Container


App
config
App
config
App
config
App
config
Docker Engine
Operating System
Hypervisor

Container



VM



App Dependencies

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

Multiple Versions

5.1

6.13

7.4

5.7

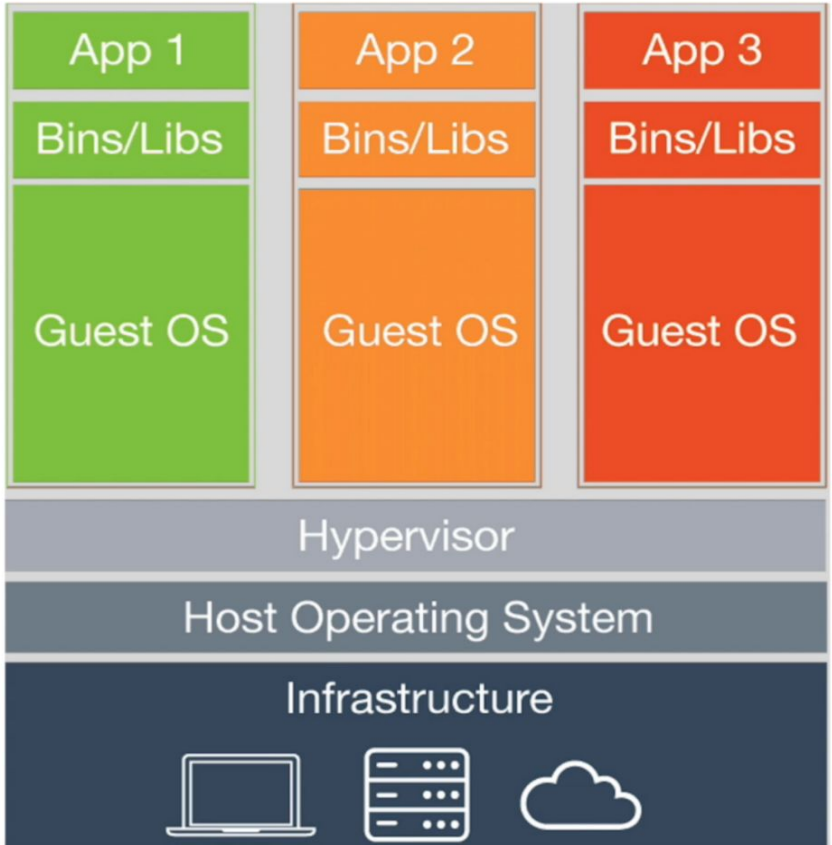
8.2

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

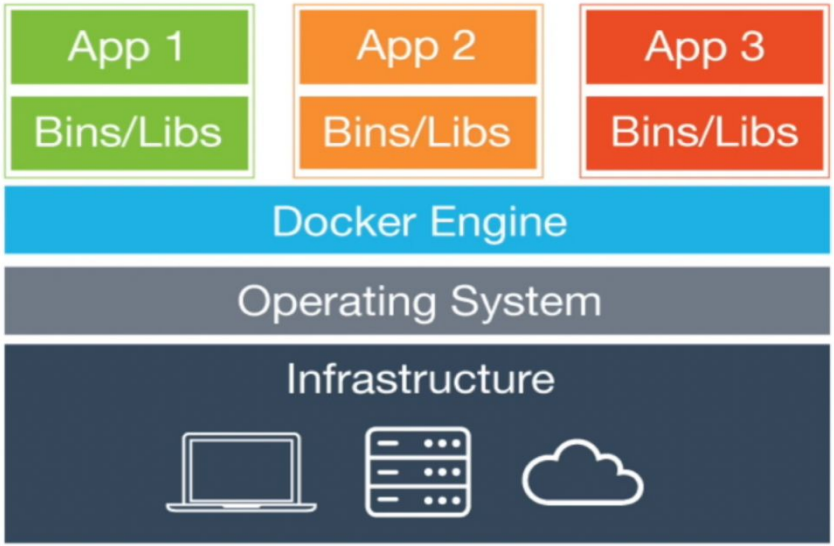
Guest OS / Virtual Machine

Hypervisor

Host OS

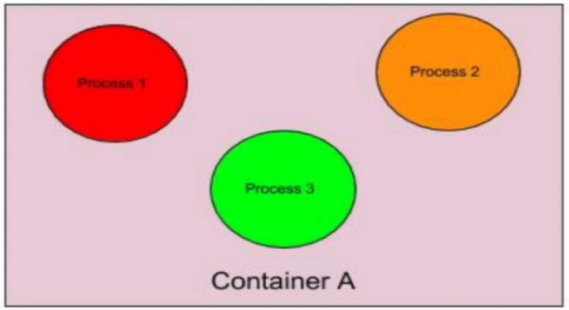
Physical Machine

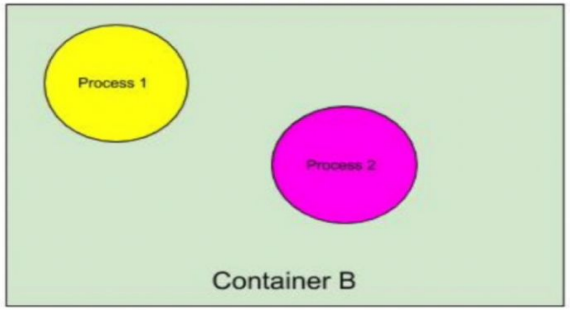
Containers

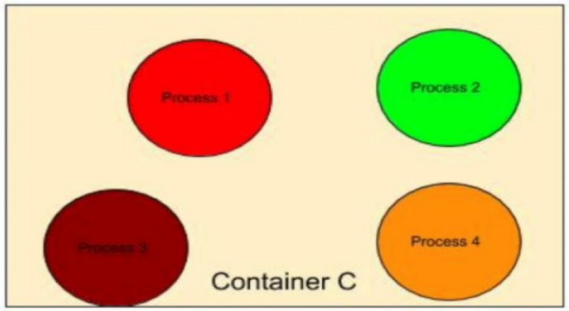


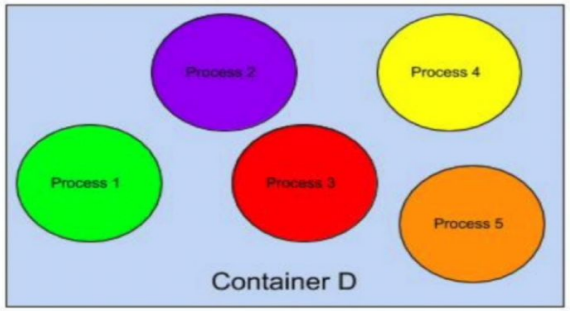
- OS - Kernel - libs/bins
- Container = App + Deps
- **Container Hypervisors**:
    A. Docker Engine,
    B. Core OS Rocket,
    C. Canonical LXC/LXD

Containers

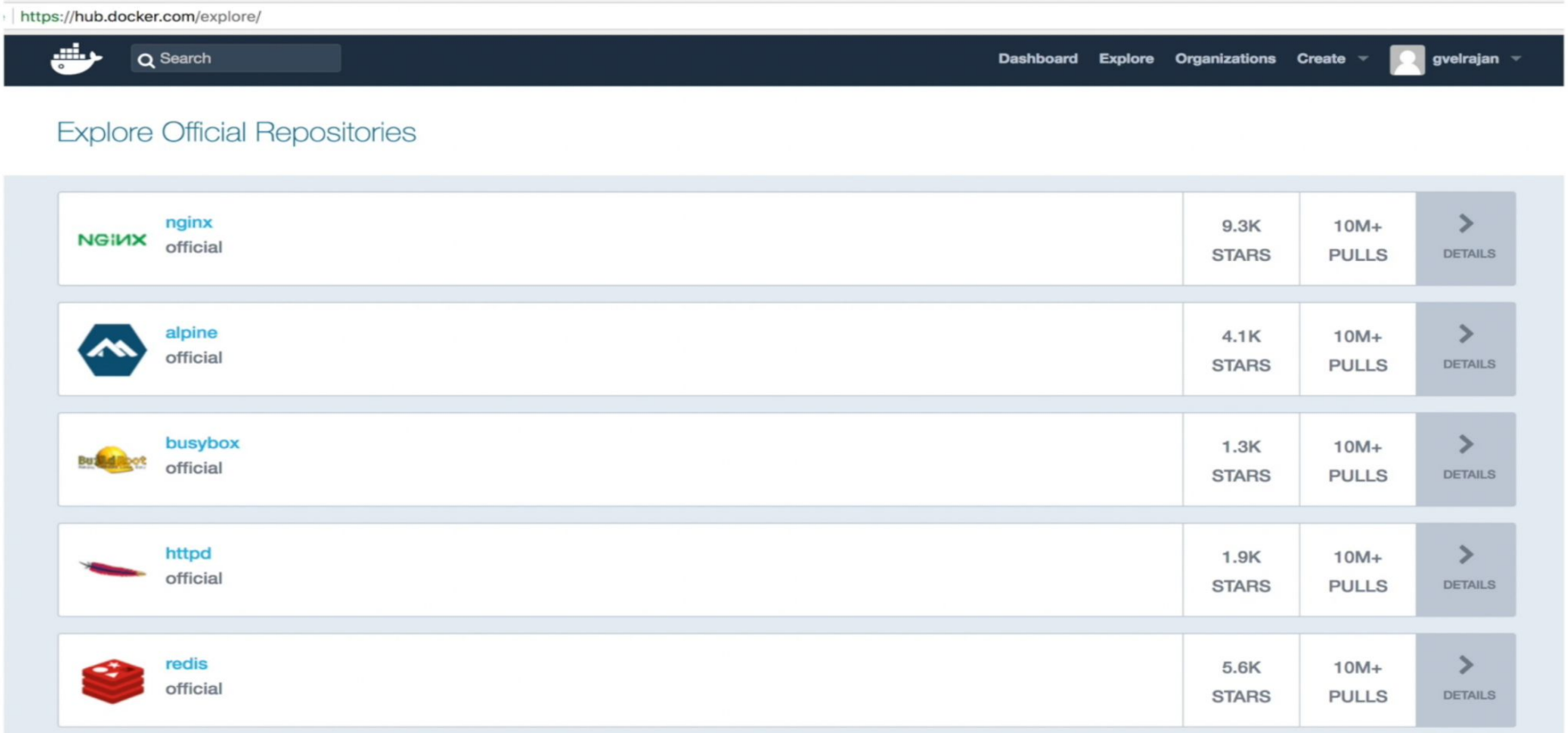

Process 1
Process 2
Process 3
Container A


Process 1
Process 2
Container B


Process 1
Process 2
Process 3
Process 4
Container C

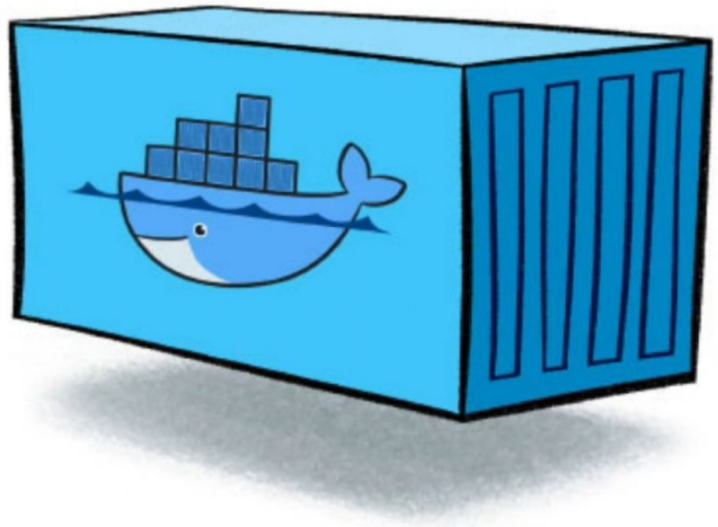

Process 2
Process 4
Process 1
Process 3
Process 5
Container D

Docker Hub


https://hub.docker.com/explore/
Search
Dashboard
Explore
Organizations
Create
gvelrajan
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

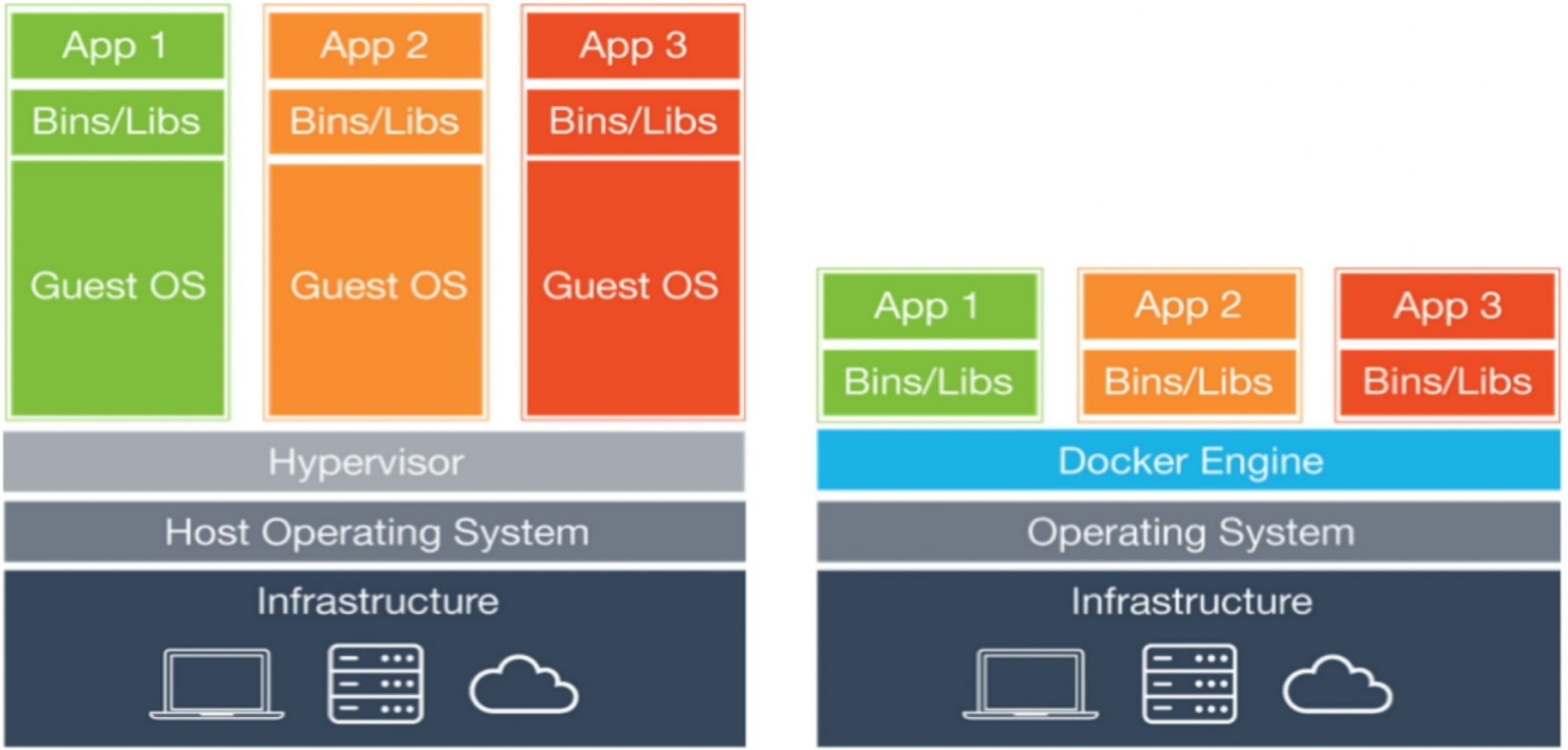
Docker Benefits

- 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

```
Docker Container Basics:
========================

docker pull <image name>
docker pull nginx

docker run -d -p 80:80 nginx

docker run -it -p 80:80 nginx

docker container logs <id>

docker exec <container name or id > <command to
docker exec afbg ls

docker exec -it <container id> /bin/bash

docker container stop  <id>

docker container start  <id>
```

Line 11, Column 27

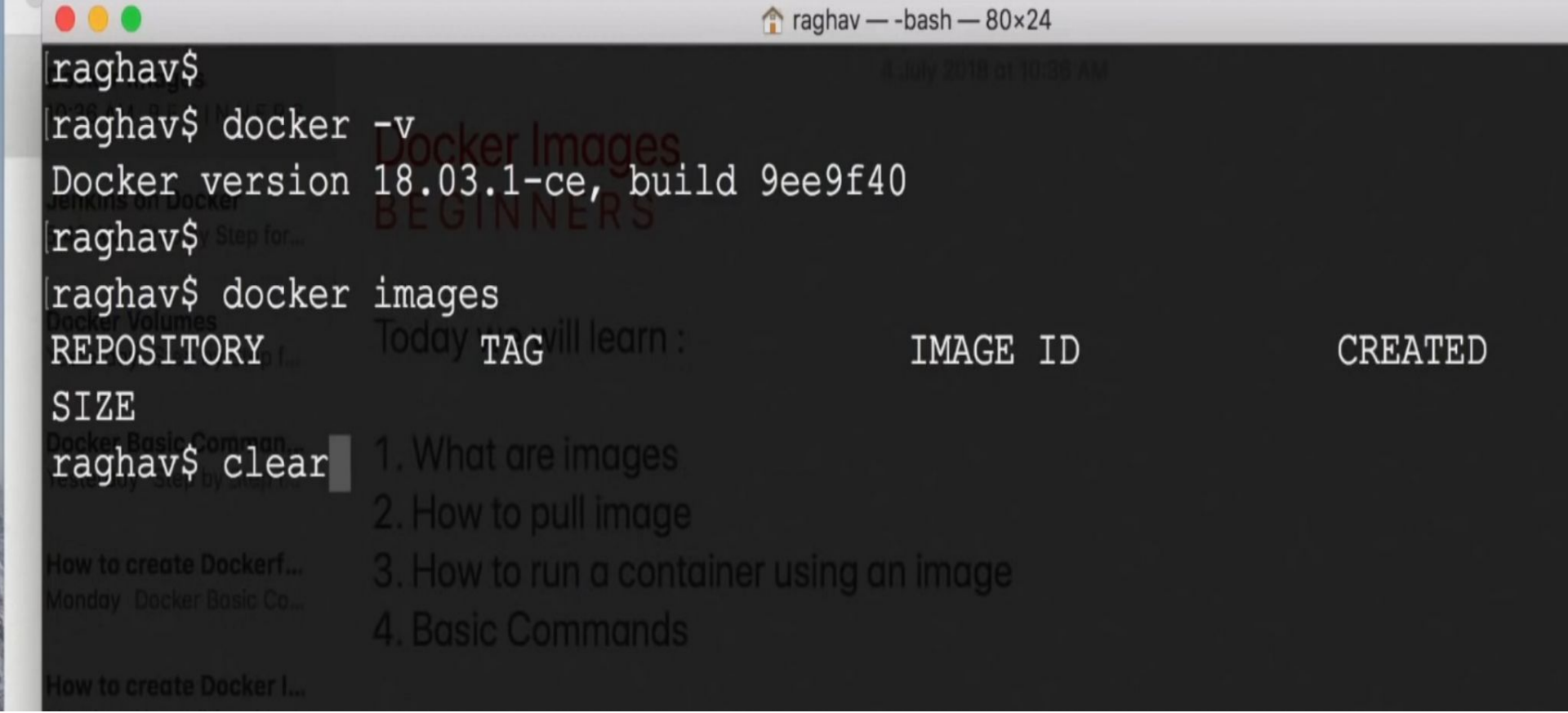
https://ssh.cloud.google.com/projects/docker-demo-218413/zones/asia-south1-c/instances/docker?authuser=

```
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:
```

Line 13, Column 33

```
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:~$
```


raghav — -bash — 80×24
raghav$
raghav$ docker -v
Docker version 18.03.1-ce, build 9ee9f40
raghav$
raghav$ docker images
REPOSITORY TAG IMAGE ID CREATED
SIZE
raghav$ clear
Docker Images
BEGINNERS
Today you will learn :
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]]

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
raghav$
```

List images

```
Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
raghav$
raghav$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6b98dfc16071: Pull complete
4001a1209541: Pull complete
6319fc68c576: Pull complete
b24603670dc3: Pull complete
97f170c87c6f: Pull complete
Digest: sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Status: Downloaded newer image for ubuntu:latest
```

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

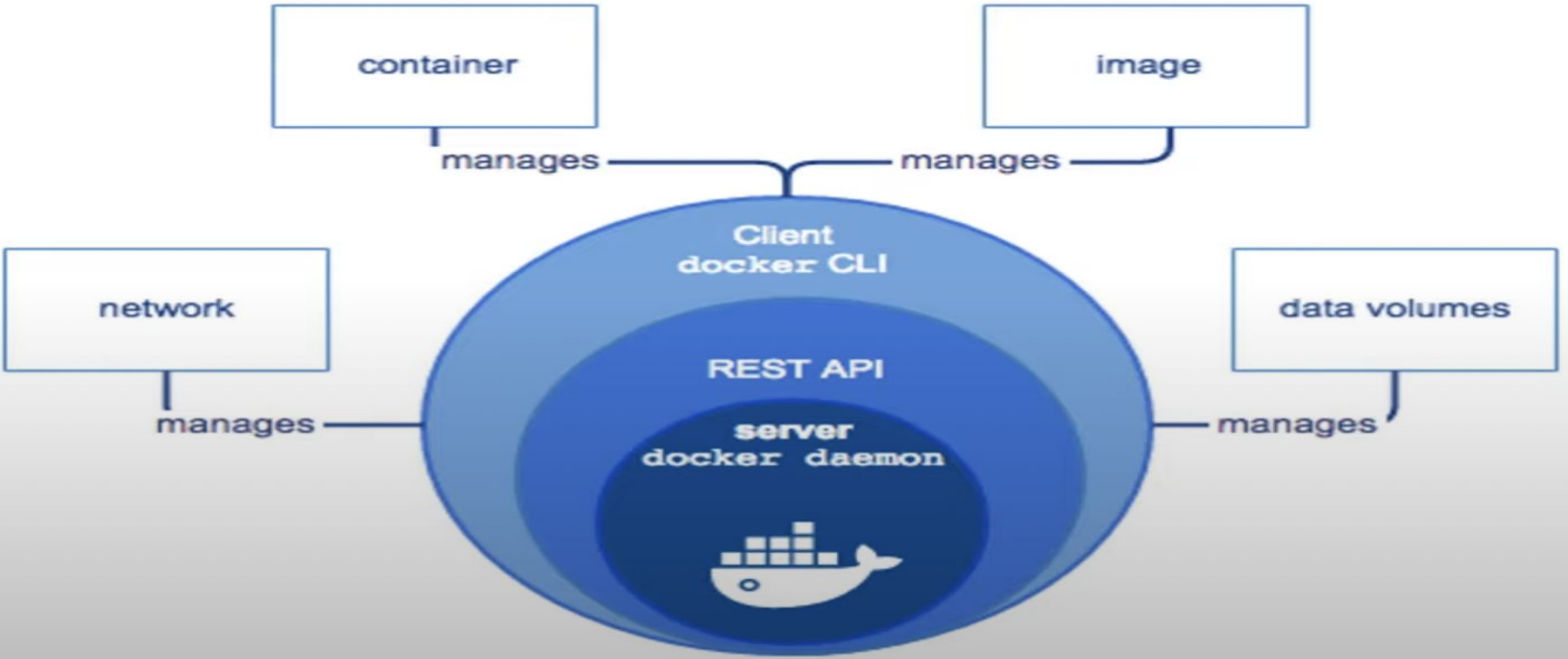
What are Images

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

Where are Images Stored

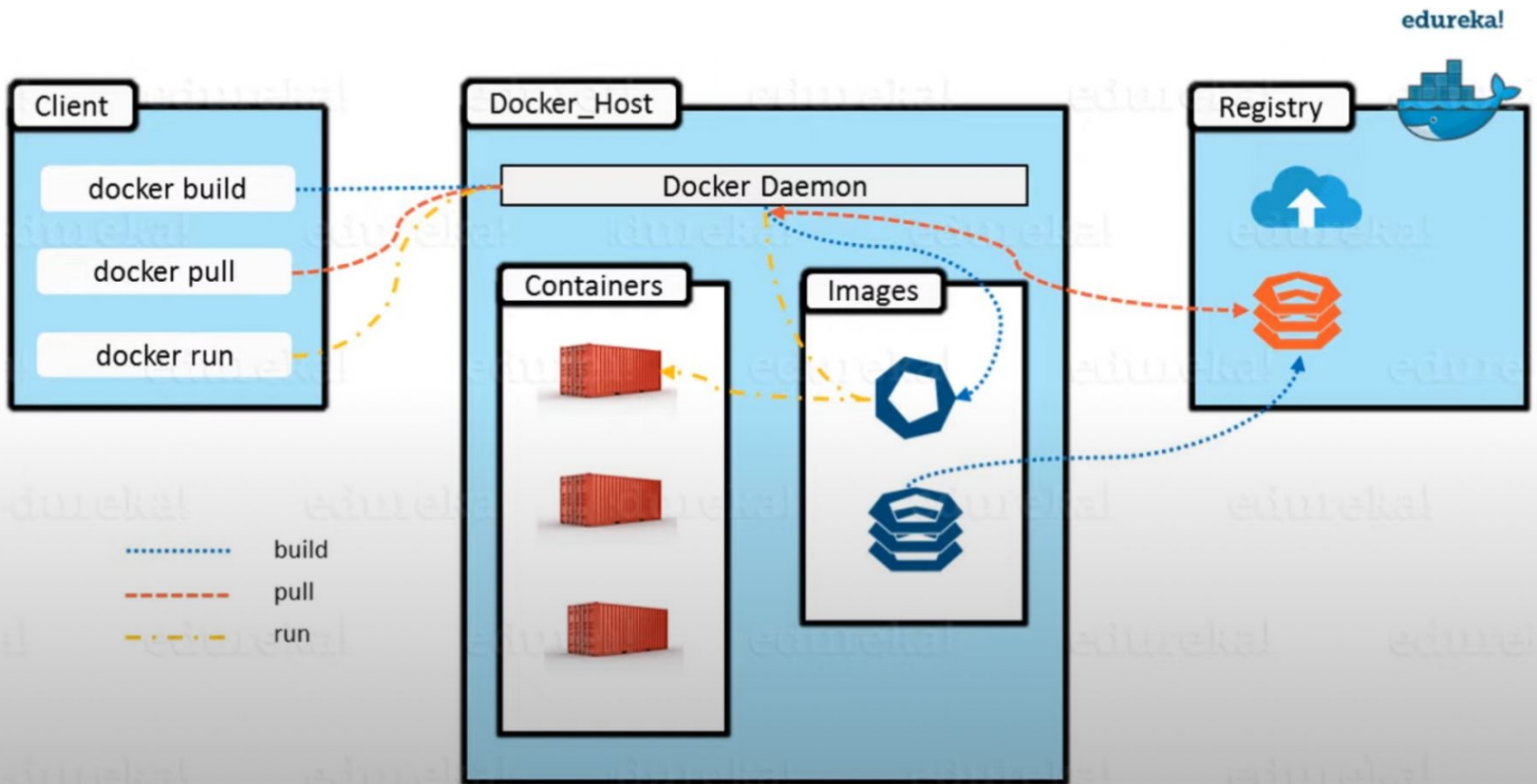
Registries (e.g. docker hub)

Docker Engine



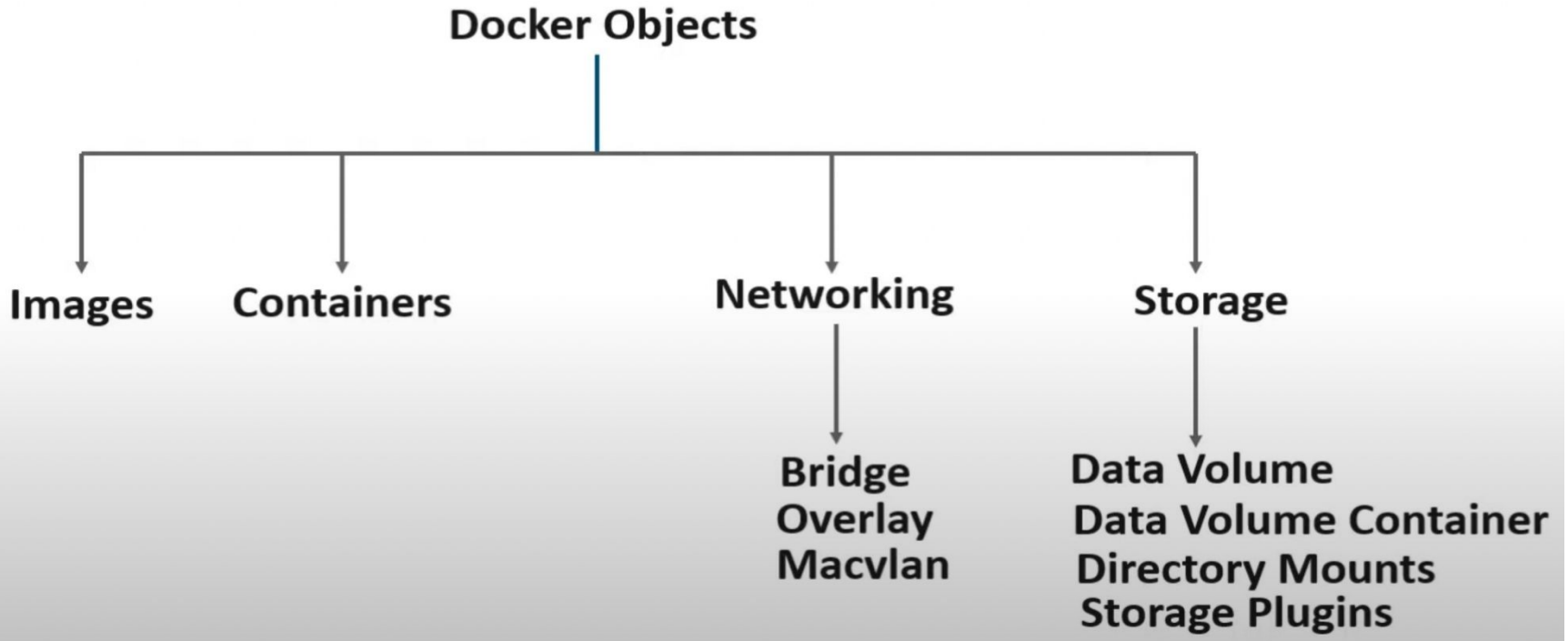
1. Docker daemon which is used for the application interact with docker daemon
2. Docker cli is command line interface which is used for interacting with docker daemon
3. Docker client talk with docker daemon building ,distributing of docker containers
4. Client and daemon can run on the same system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

Docker Architecture



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
-

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.

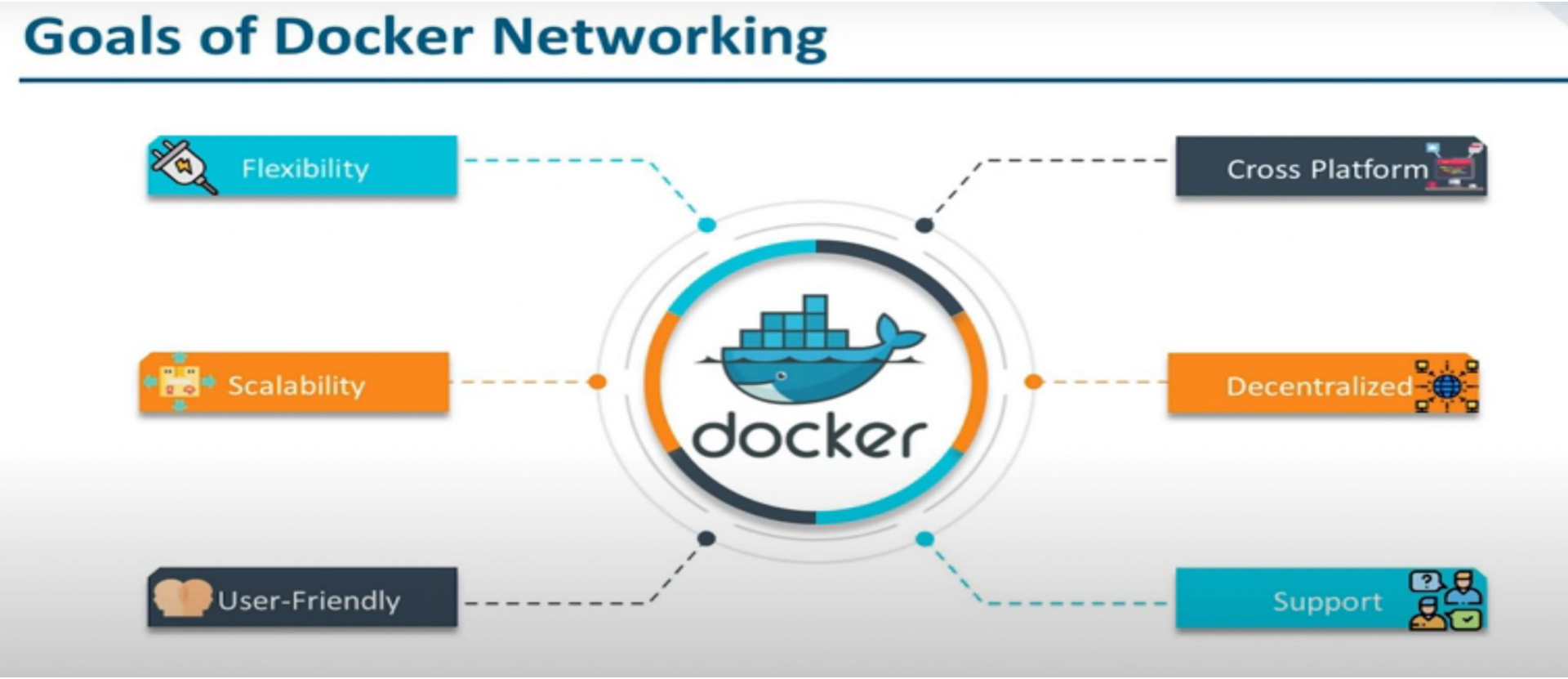
Compose is great for development, testing, and staging environments, as well as CI workflows. You can learn more about each case in Common Use Cases.

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.

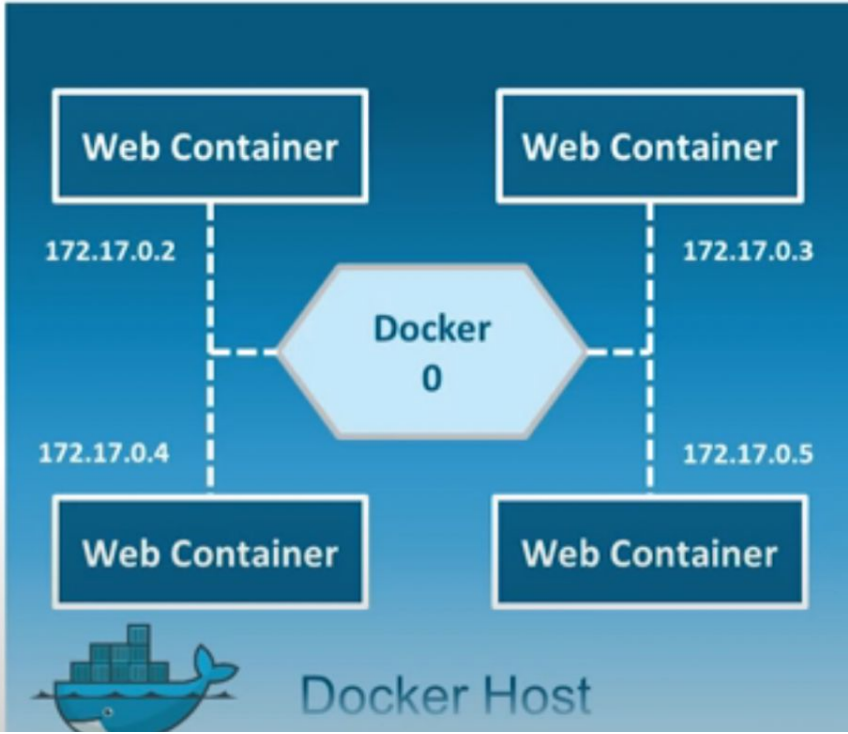
Docker Compose

Docker Networking



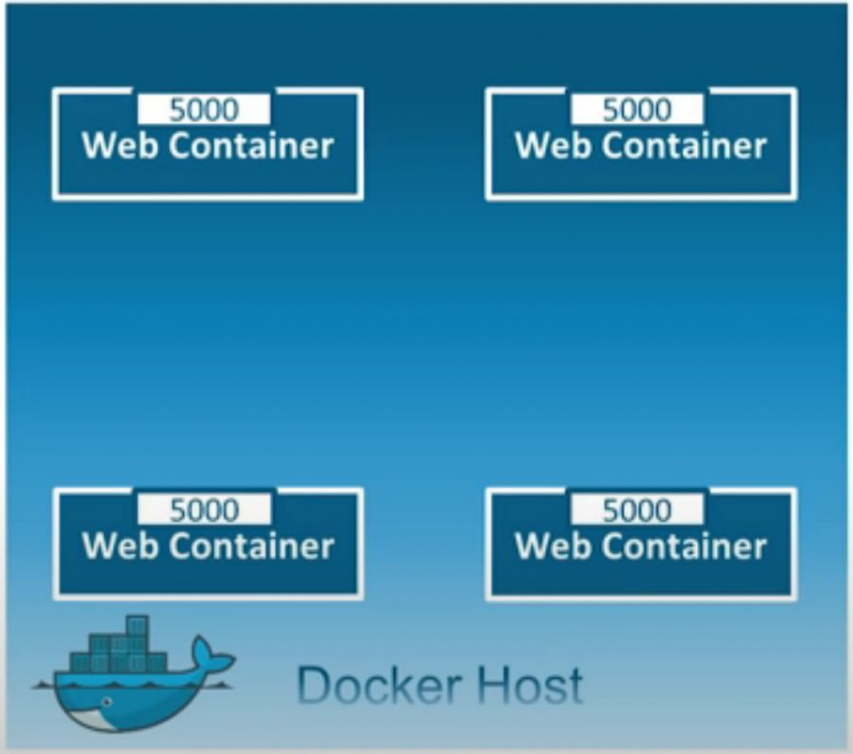
Network Drivers: Bridge

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.

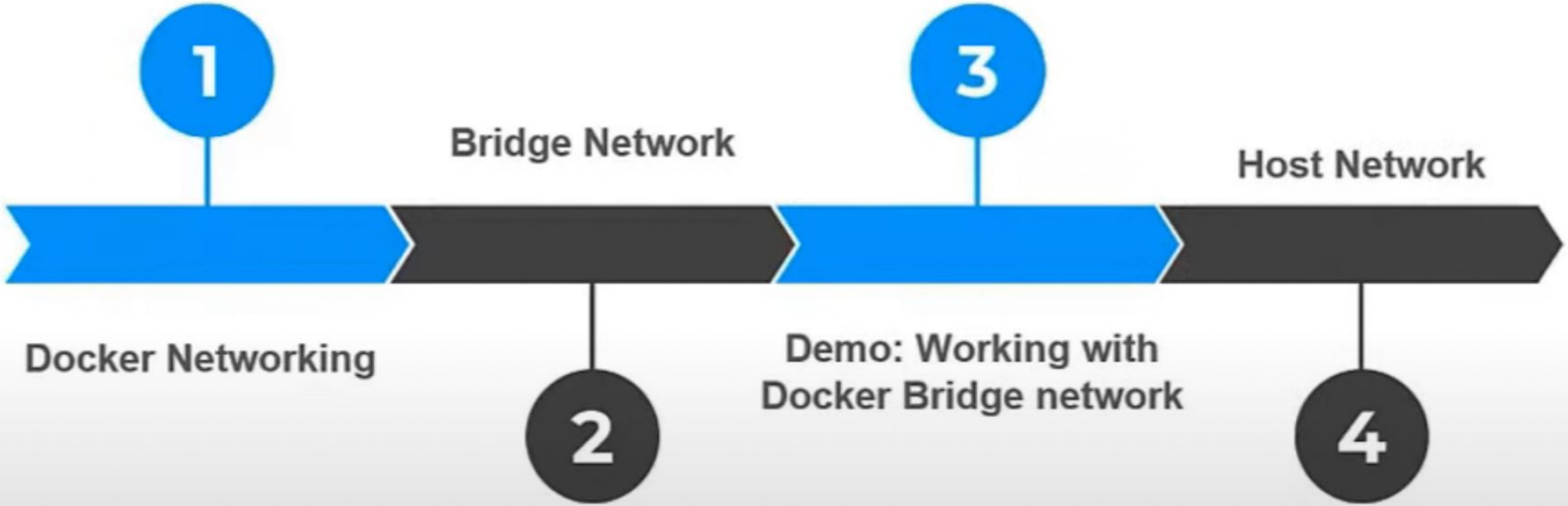


Network Drivers: Host

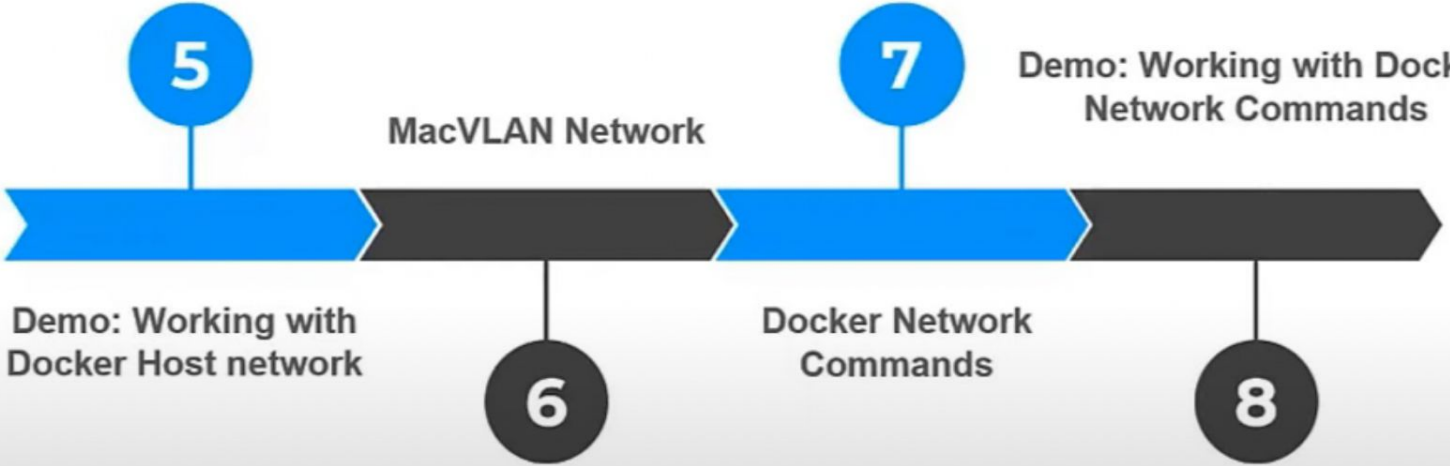
For standalone containers, removes network isolation between the container and the Docker host, and uses the host's networking directly.



Agenda


1
Docker Networking
2
Bridge Network
3
Demo: Working with
Docker Bridge network
4
Host Network

Agenda


5
Demo: Working with Docker Host network
MacVLAN Network
6
7
Docker Network Commands
Demo: Working with Docker Network Commands
8

```
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?

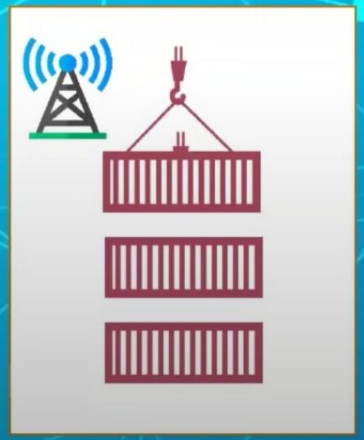
Docker networking enables a user to link a Docker container to as many networks as he/she requires

What is Docker Networking?

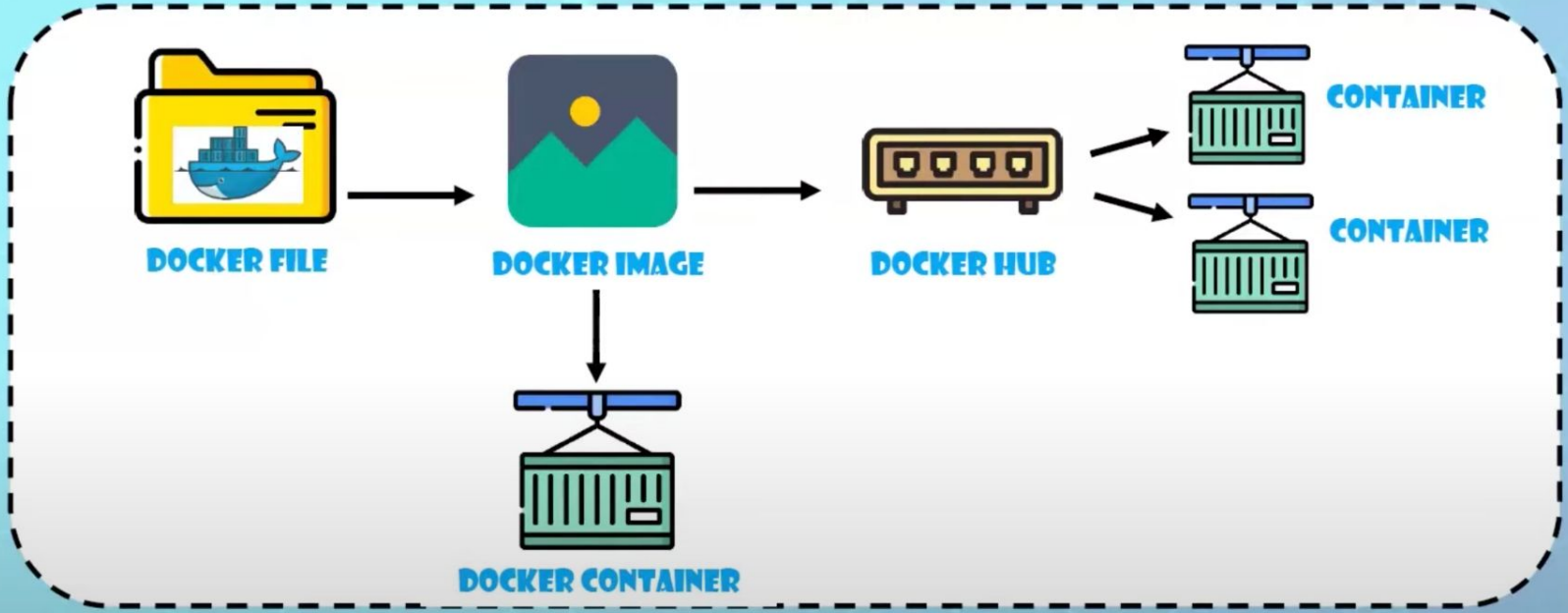
Docker networking enables a user to link a Docker container to as many networks as he/she requires

Docker Networks are used to provide complete isolation for Docker containers

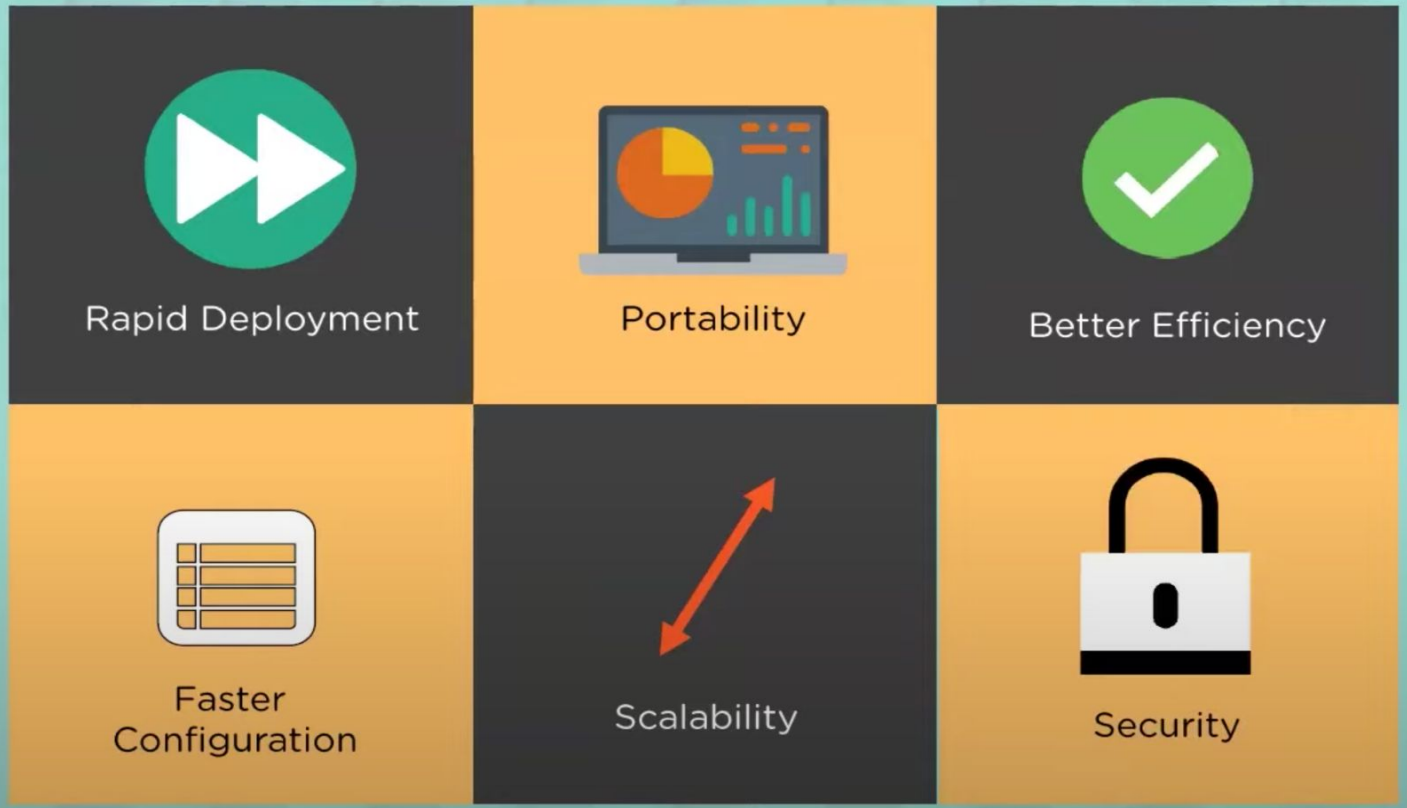
Note: A user can add containers to more than one network



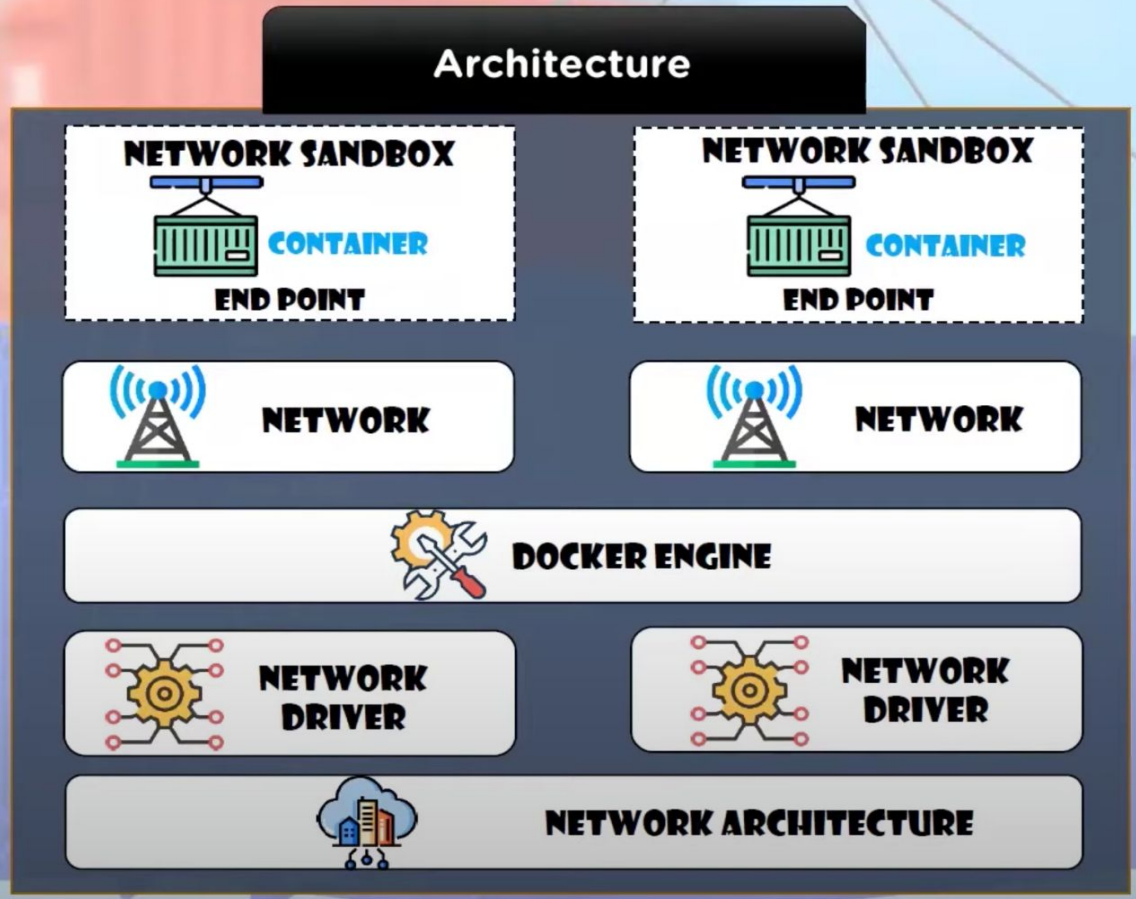
How Docker Networking Works?



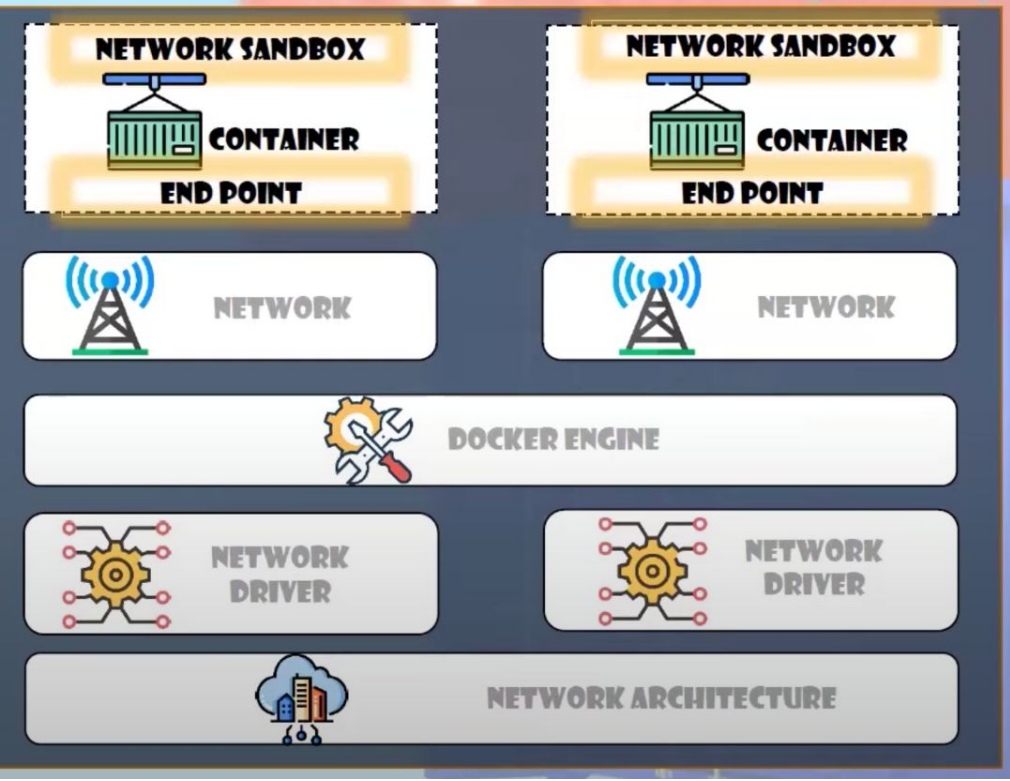
Advantages of Docker Networking



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

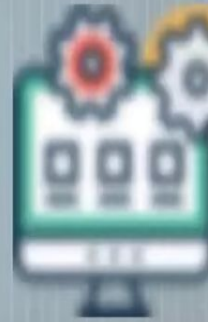
NONE

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

HOST

- 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

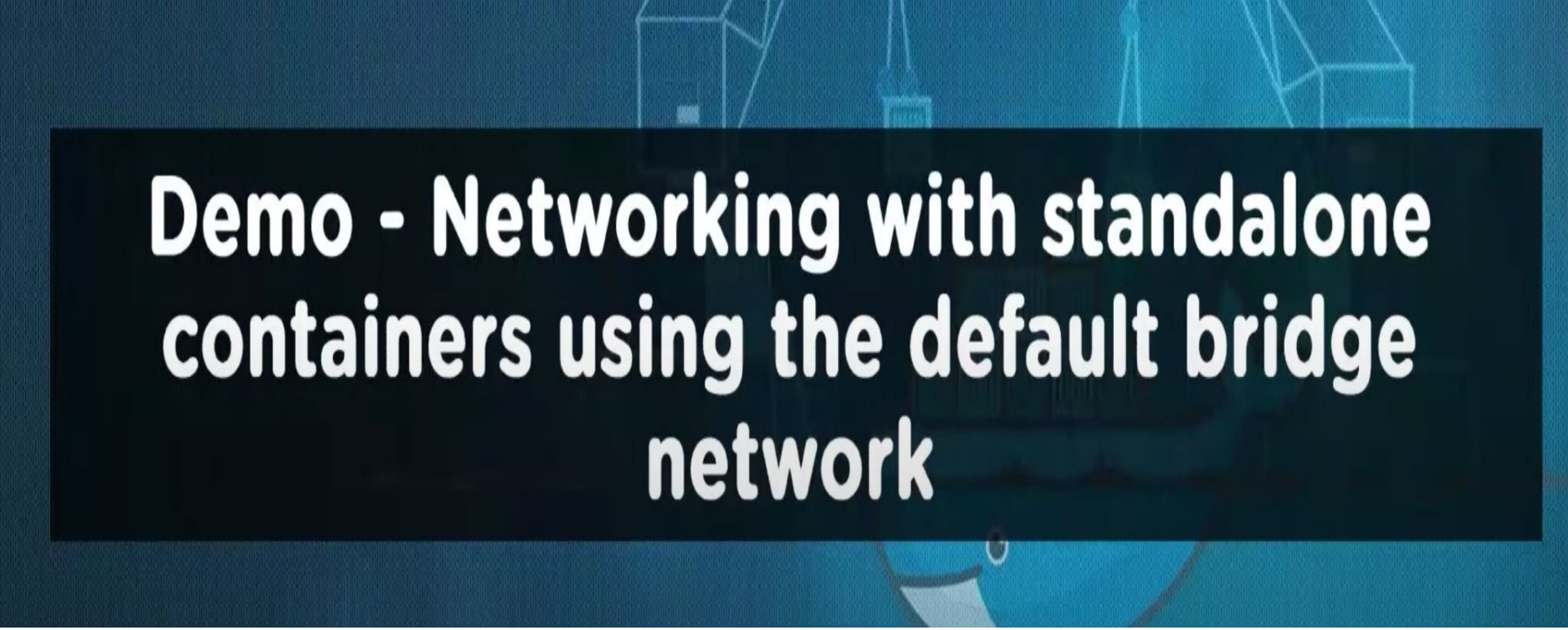
MACVLAN

Network Drivers

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

BRID

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:~#
```

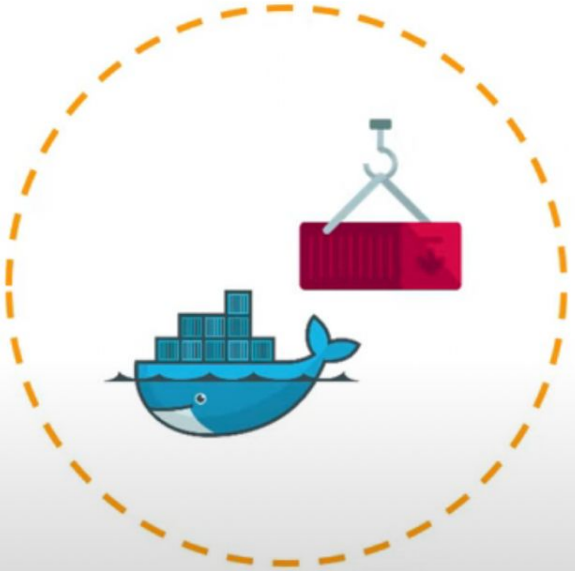
```
root@ip-172-31-75-99:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e65c4305d6a1        bridge              bridge              local
729ac5c85118        host                host                local
8a3bfa625405        none                null            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

```
 docker ps
GE                  COMMAND             CREATED             STATUS              PORTS               NAMES
ine                 "ash"               10 seconds ago      Up 9 seconds                            alpine2
ine                 "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": "bdcf411f60d6fc8212fede717a16413f114dd0b0fa2f15c9dd757d9ae93daaed",
         "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
```

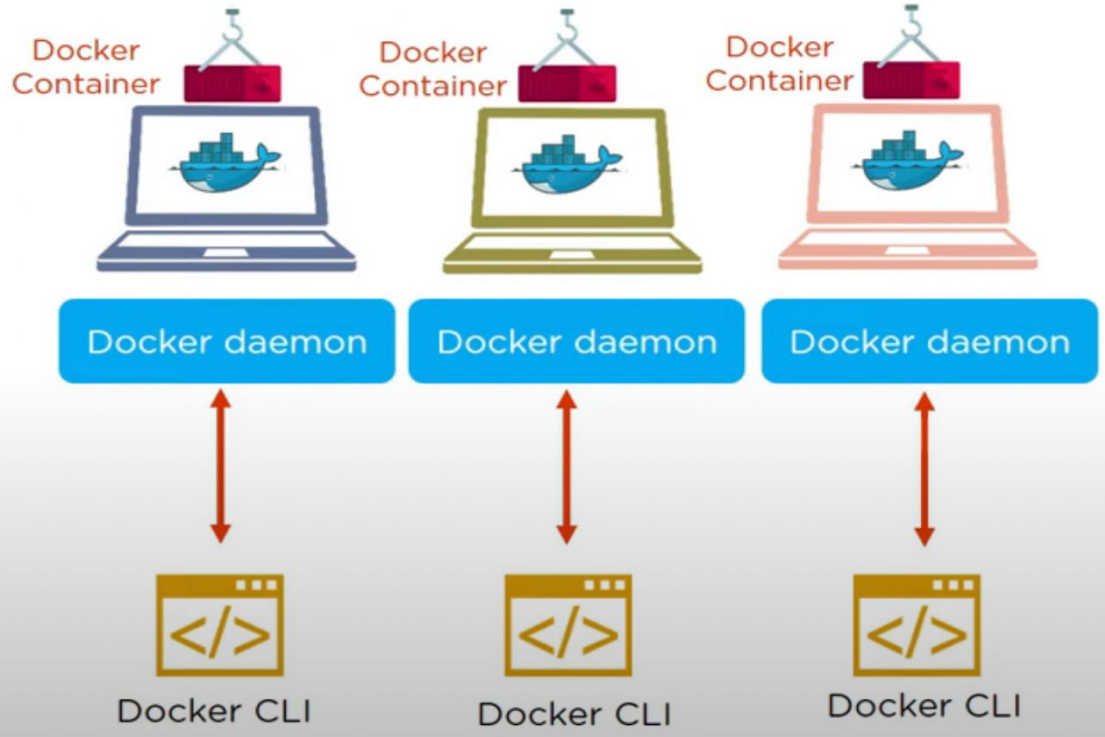
What is Docker Swarm?

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



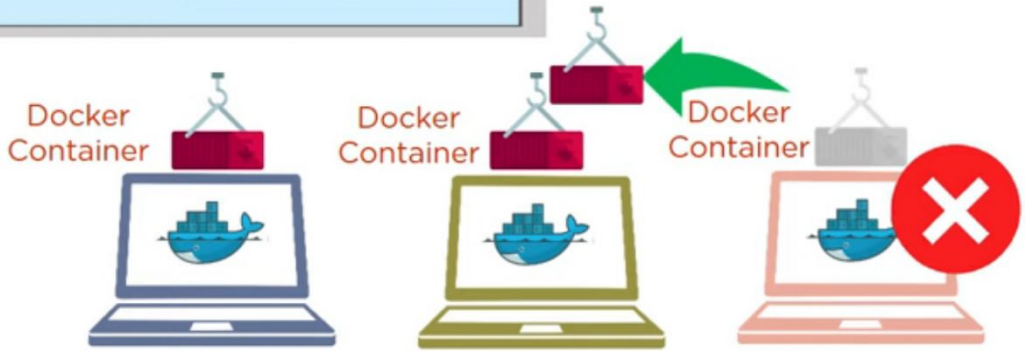
What is Docker Swarm?

With Docker

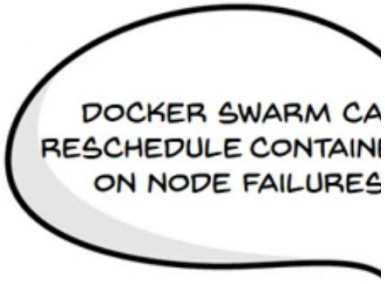


What is Docker Swarm?

With Docker Swarm on fault tolerance

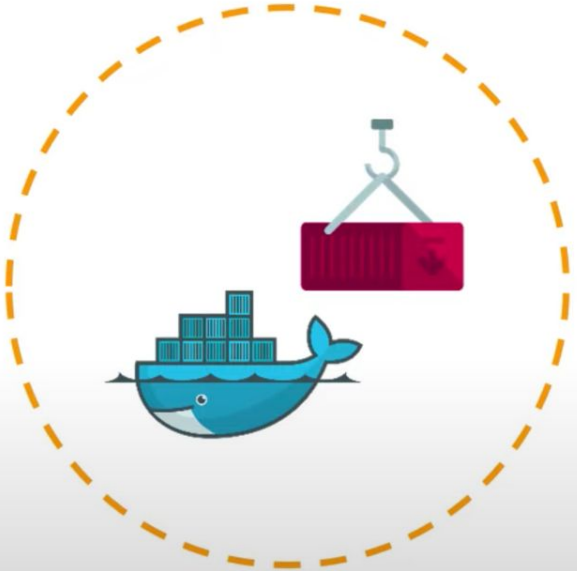


Docker daemon | Docker daemon | Docker daemon



What is Docker Swarm?

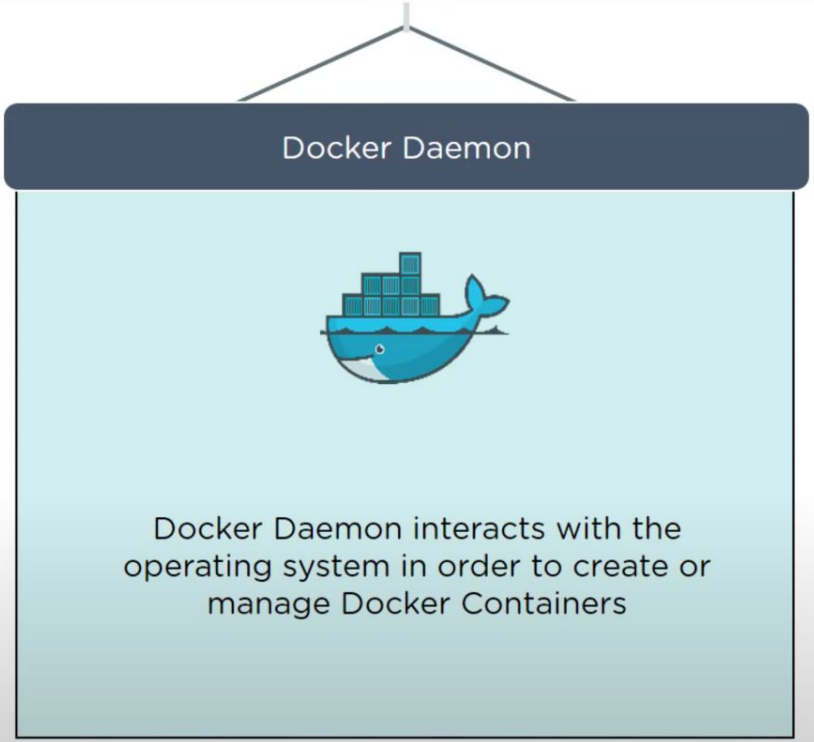
- Docker Swarm is a service which allows users to create and manage a cluster of Docker nodes and schedule containers

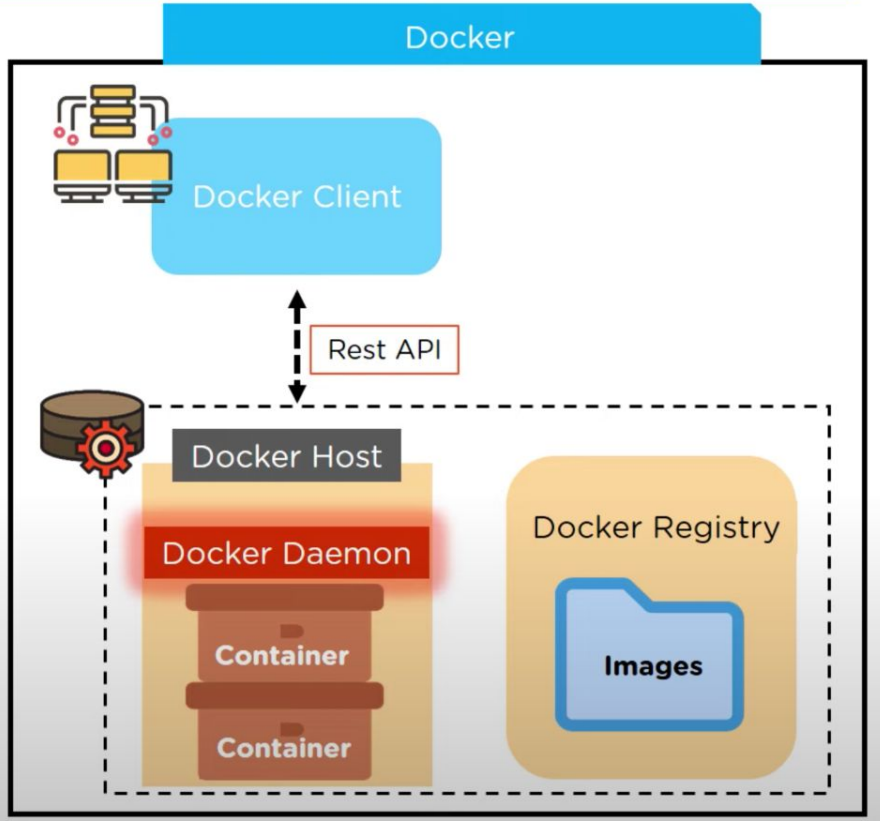


What is Docker Swarm?

- 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

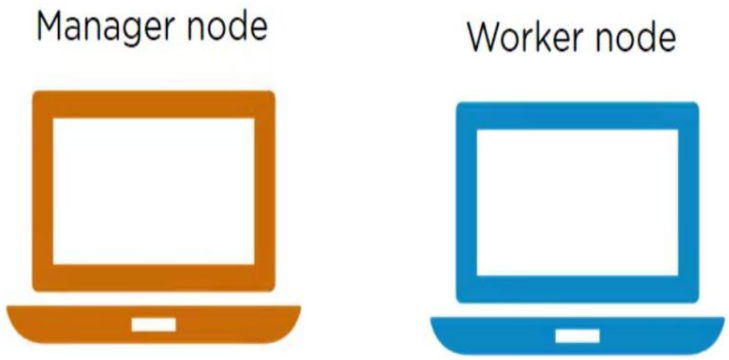
Auto load balancing

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



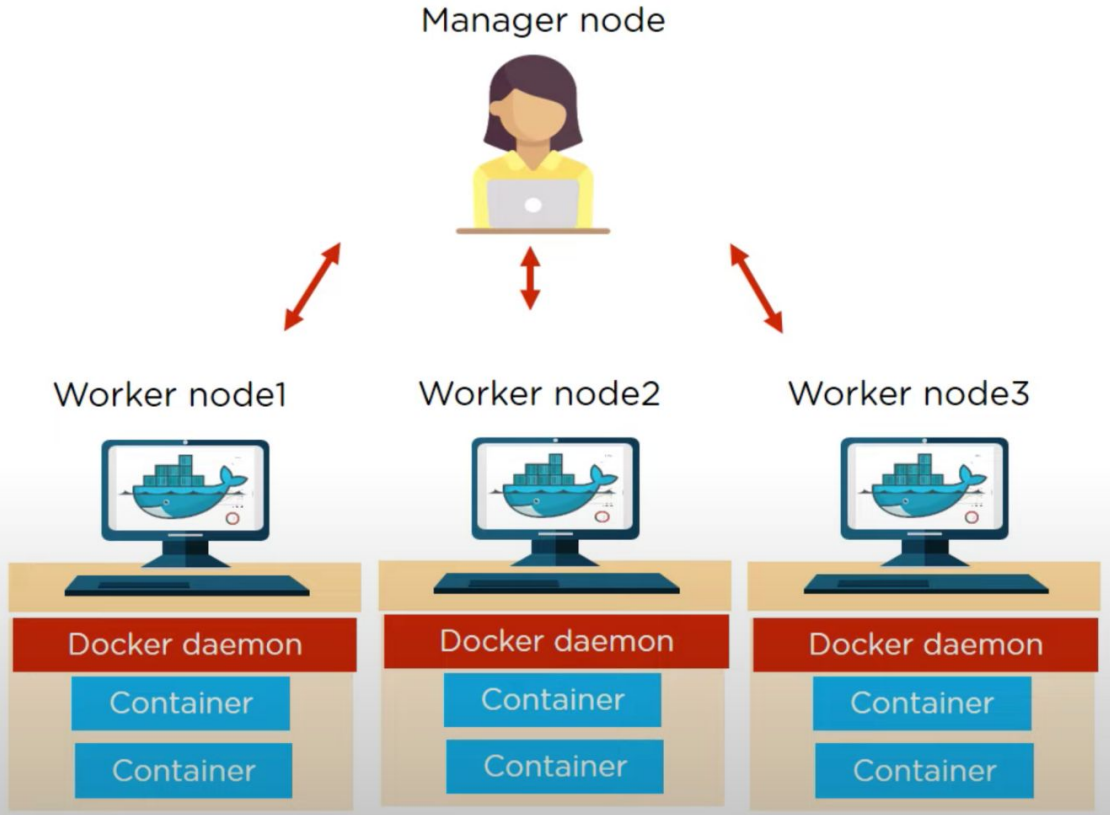
Docker Swarm

- 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 maintains cluster management tasks

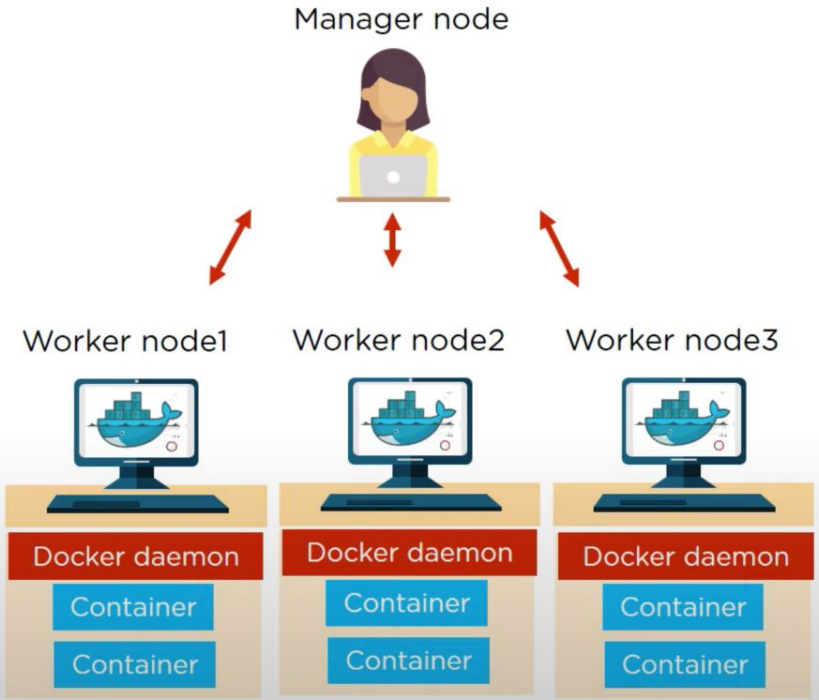
Worker nodes receive and execute tasks from manager node

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 as an agent, which reports on the state of the node's tasks to the manager

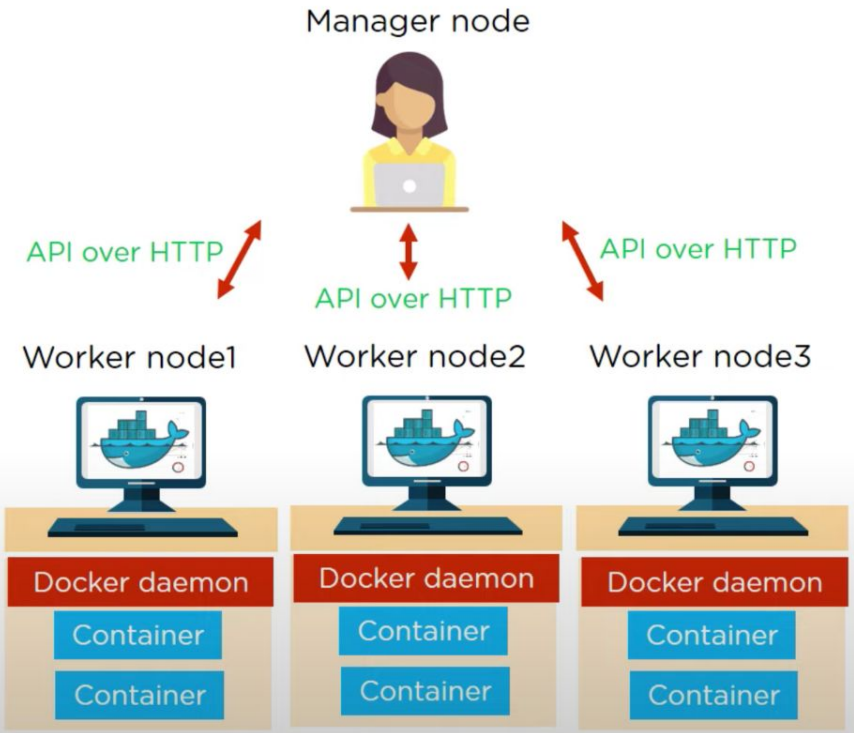


How does Docker Swarm work?

- 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

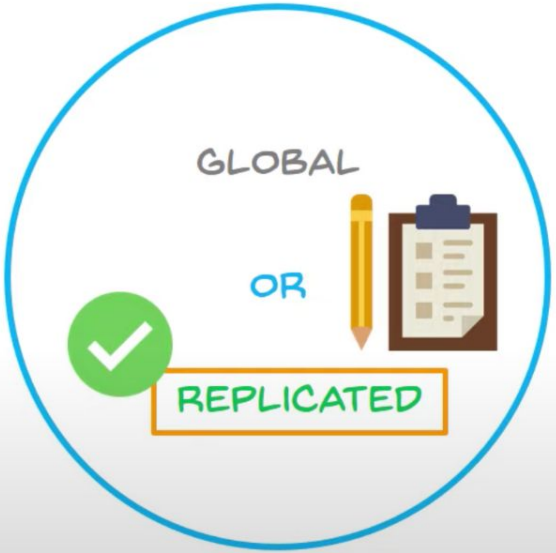
How does Docker Swarm work?

- 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
- The worker nodes communicate with the manager node using API over HTTP



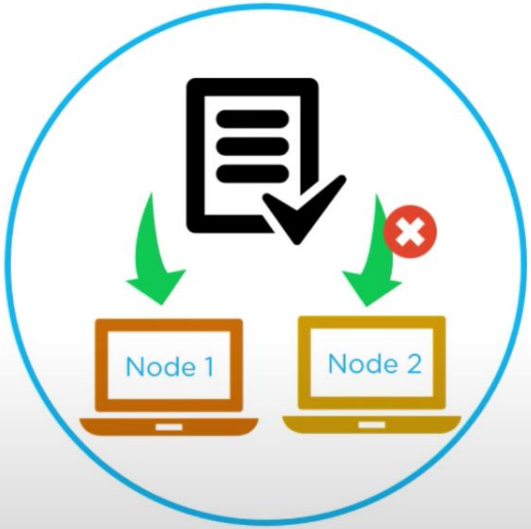
How does Docker Swarm work?

- 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?

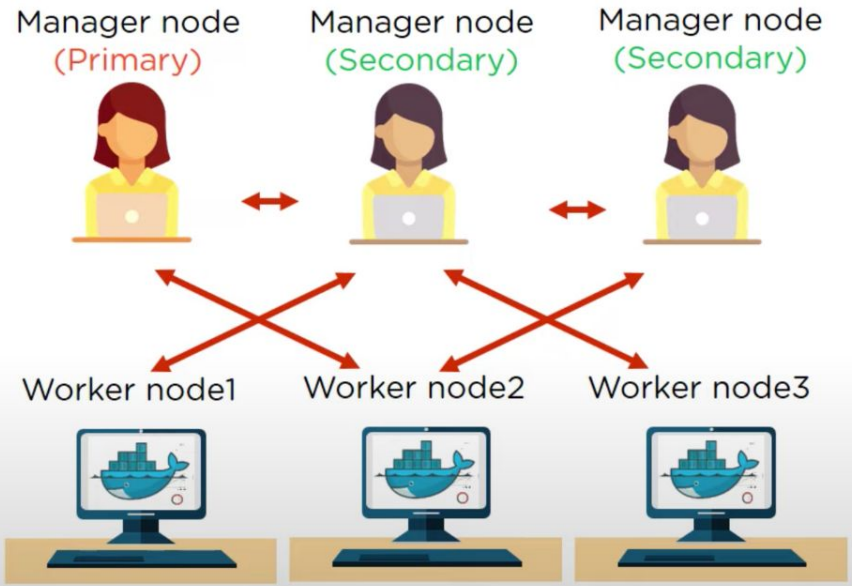
- A service is a description of a task or the state, whereas a task does the work
- Docker enables a user to create services, which can start tasks
- When a task is assigned to a node, it cannot be assigned to another node



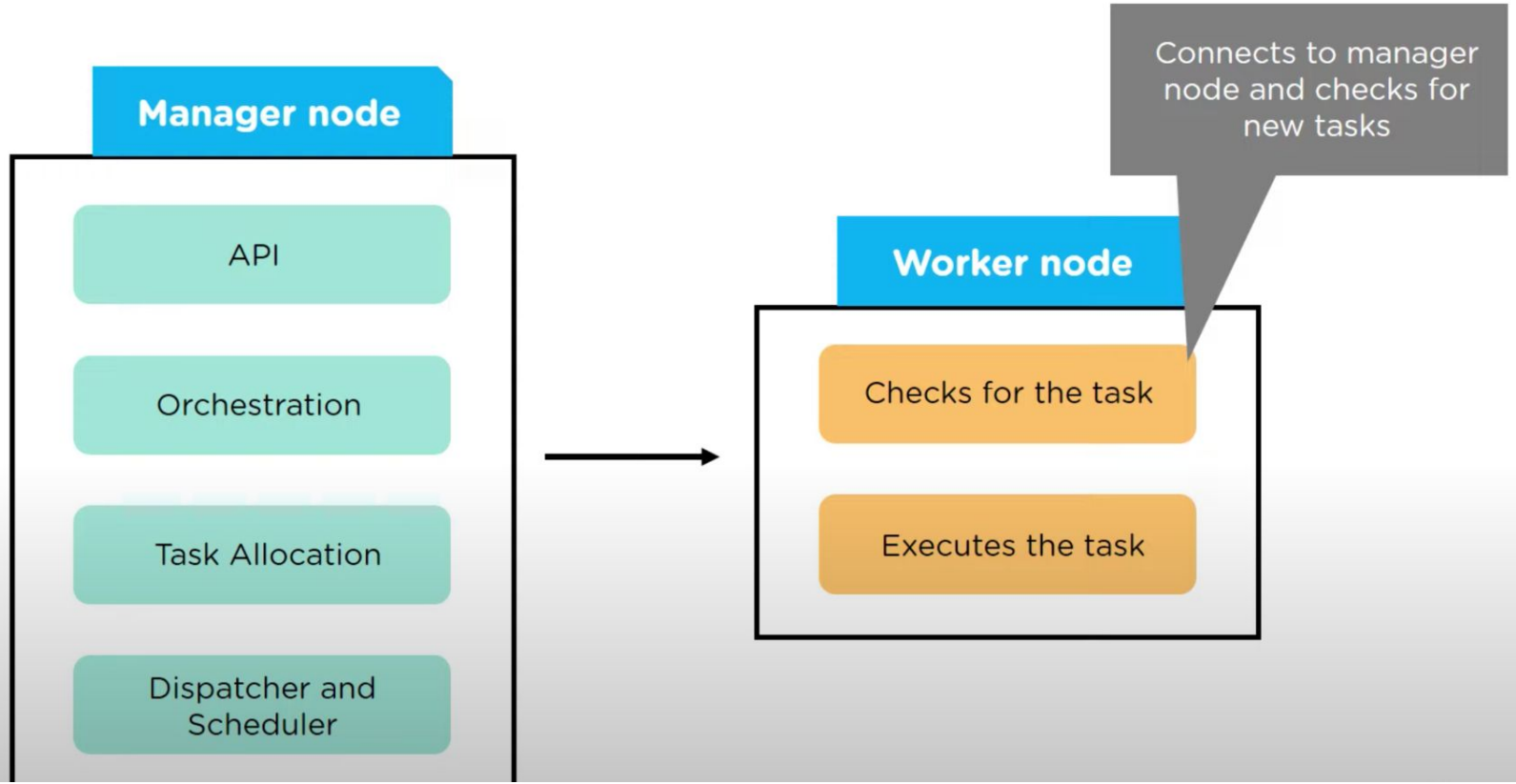
How does Docker Swarm work?

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

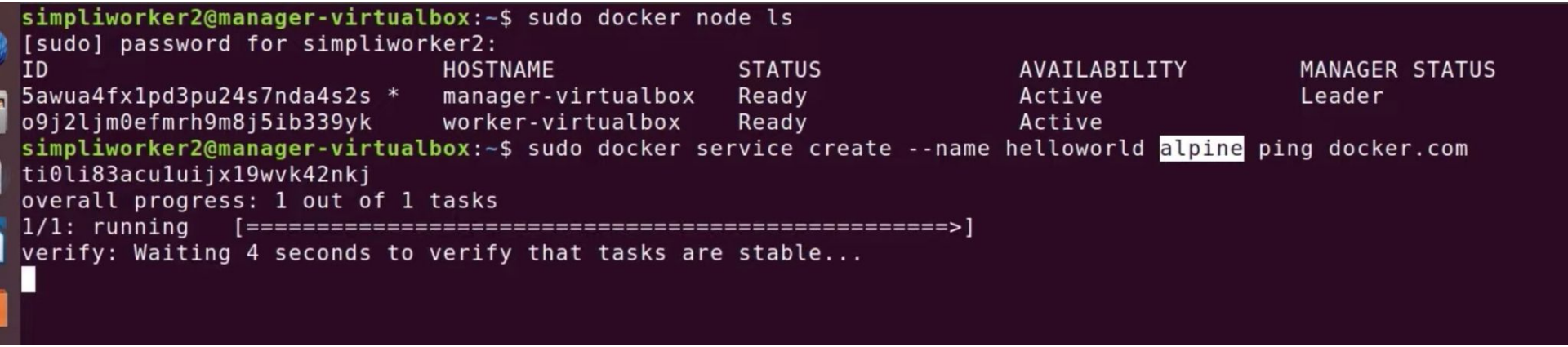

Manager node
API
Orchestration
Task Allocation
Dispatcher and Scheduler
Worker node
Checks for the task
Executes the task
Connects to manager node and checks for new tasks

```
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:~$
```

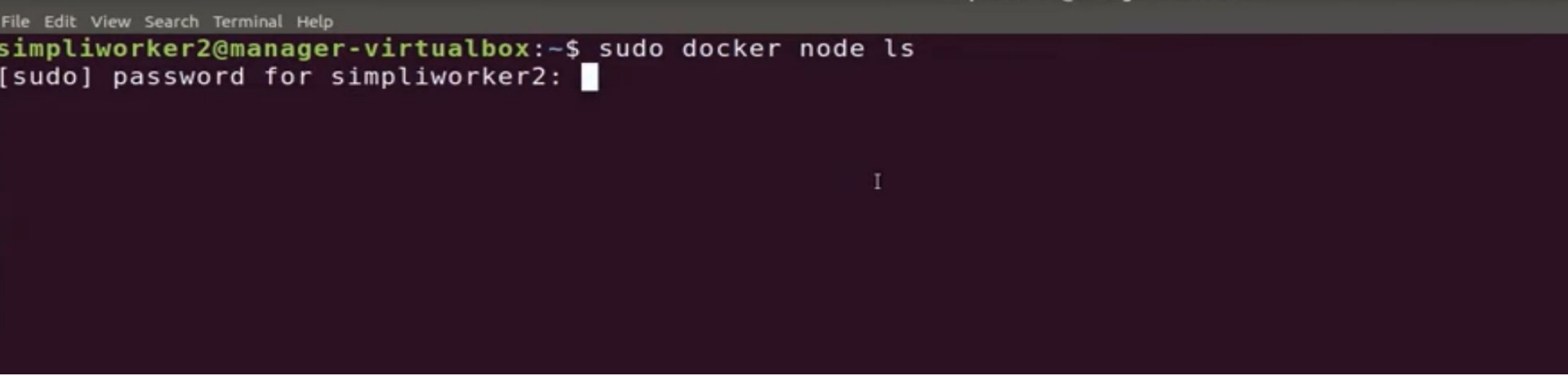
```
simpliworker2@manager-virtualbox:~$ sudo docker swarm init --advertise-addr 192.168.2.151
[sudo] password for simpliworker2:
Swarm initialized: current node (5awua4fx1pd3pu24s7nda4s2s) is now a manager.

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

    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:

    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



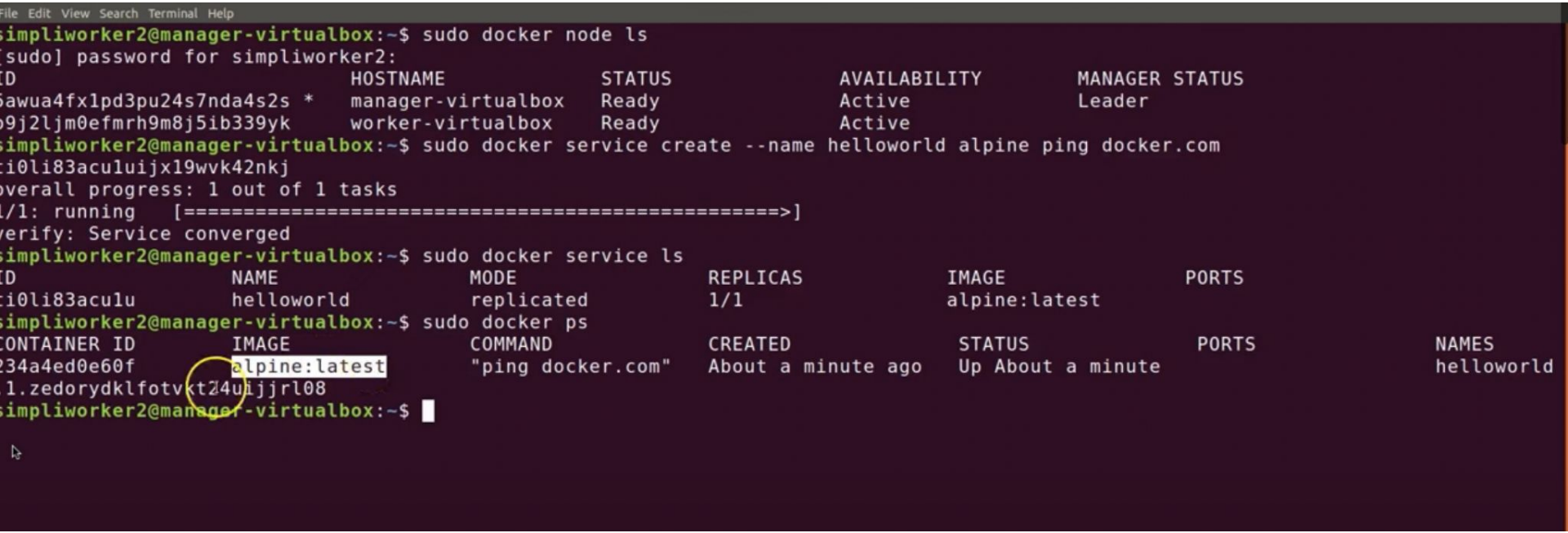
```
sudo docker service  ls
ID            NAME        MODE         REPLICAS   IMAGE          PORTS
3ki6e2o5sug9  helloworld  replicated   1/1        alpine:latest
```


File Edit View Search Terminal Help
simpliworker2@manager-virtualbox:~$ sudo docker node ls
[sudo] password for simpliworker2:


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
5awua4fx1pd3pu24s7nda4s2s * manager-virtualbox Ready Active Leader
o9j2ljm0efmrh9m8j5ib339yk worker-virtualbox Ready Active
simpliworker2@manager-virtualbox:~$ sudo docker service create --name helloworld alpine ping docker.com
ti0li83acu1uijx19wvk42nkj
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
ti0li83acu1u helloworld replicated 1/1 alpine:latest
simpliworker2@manager-virtualbox:~$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
234a4ed0e60f alpine:latest "ping docker.com" About a minute ago Up About a minute helloworld
.1.zedorydklfotvkt24uijjrl08
simpliworker2@manager-virtualbox:~$