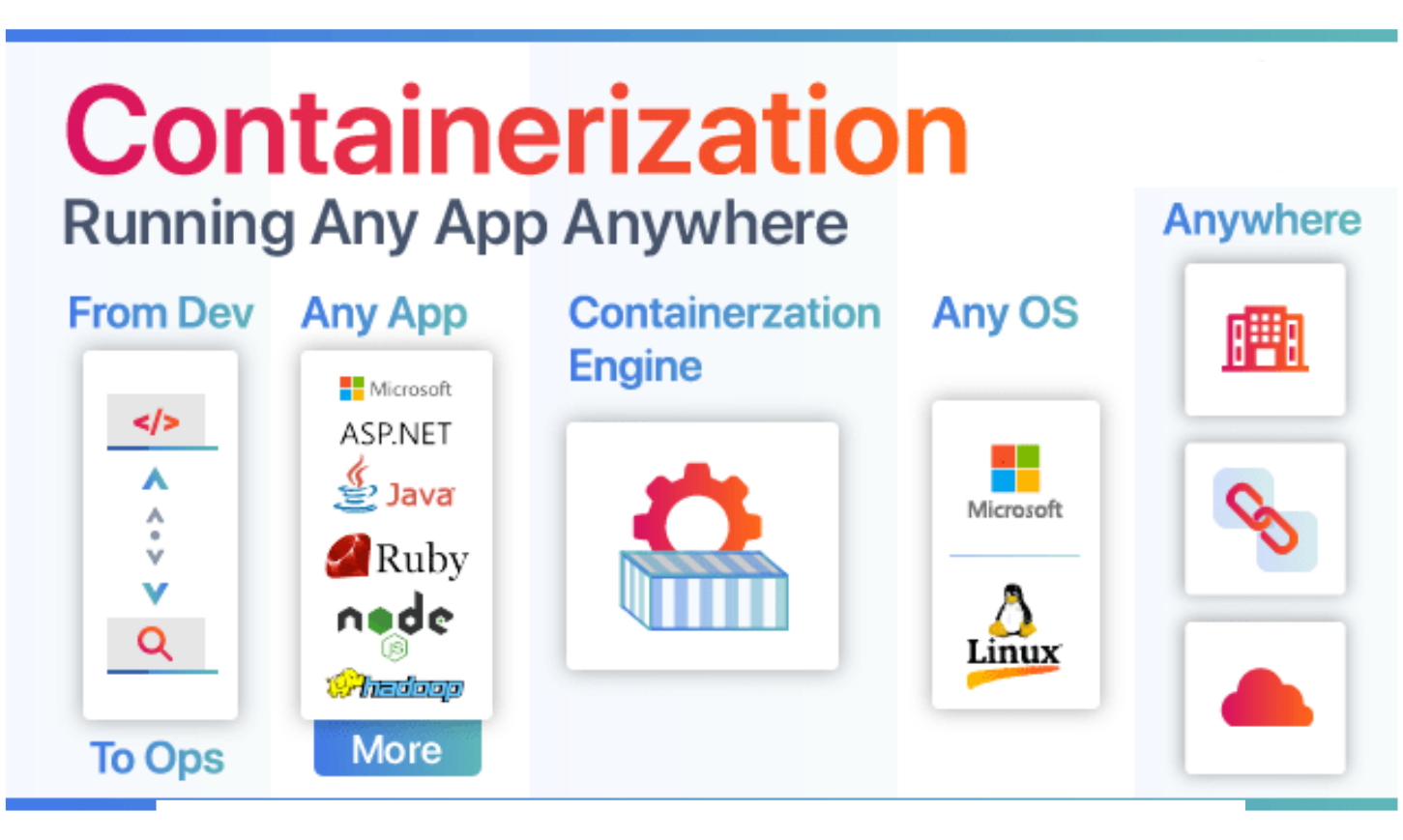
Docker Simplified: An Introductory Guide for Beginners

Welcome to the world of Docker! This document is tailored for beginners who are eager to explore Docker and its capabilities in containerization. Docker has revolutionized how applications are developed, shipped, and deployed, offering a lightweight and efficient solution to container management.
In this guide, I have endeavored to provide a clear and concise overview of Docker's fundamentals. From understanding containers to practical examples of Dockerizing applications, this document will equip you with the foundational knowledge needed to get started.
To enhance your learning experience, I have collected images and diagrams from various sources within the Docker community and educational platforms like YouTube. These visuals are used to illustrate concepts effectively and simplify complex topics.
Your feedback is valuable in improving this guide for future readers. If you have any questions, suggestions, or encounter any issues, please don't hesitate to text me. I would be happy to hear from you and assist in any way possible.

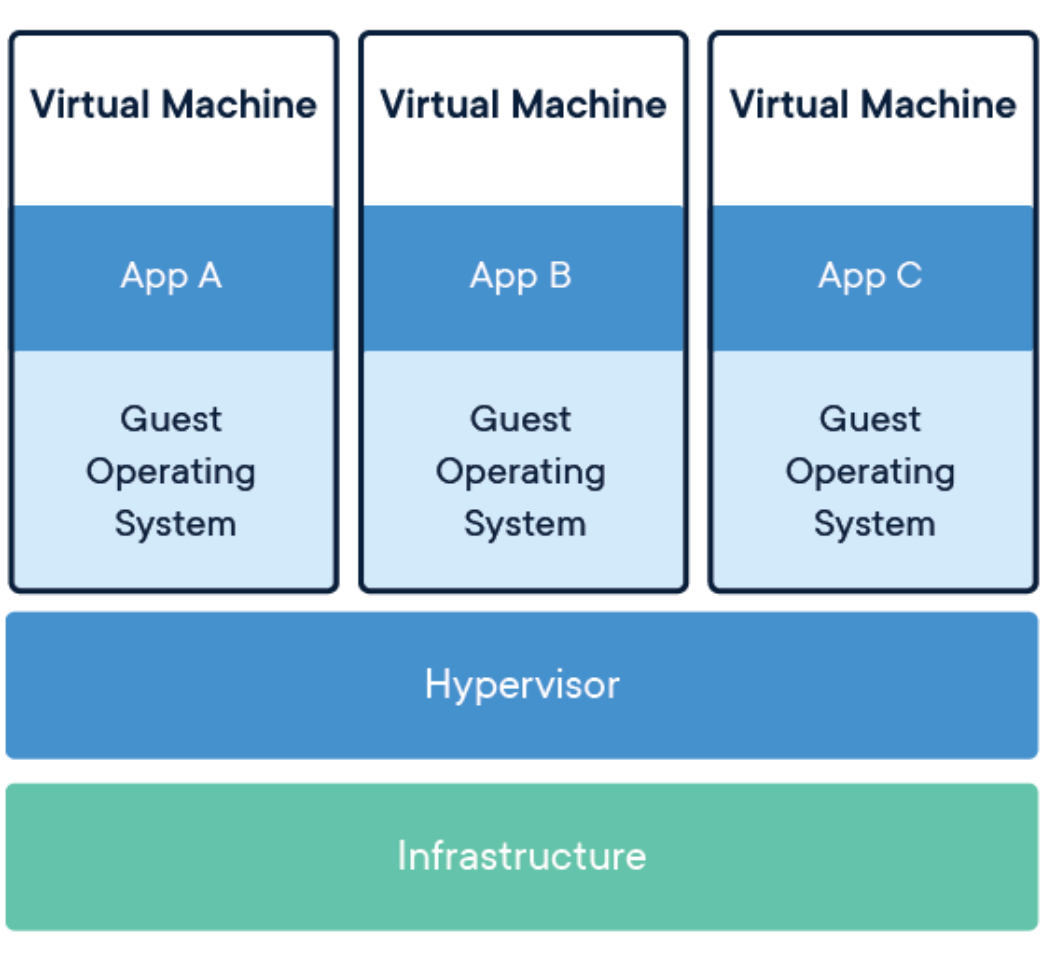
Let's dive into Docker and simplify containerization together!



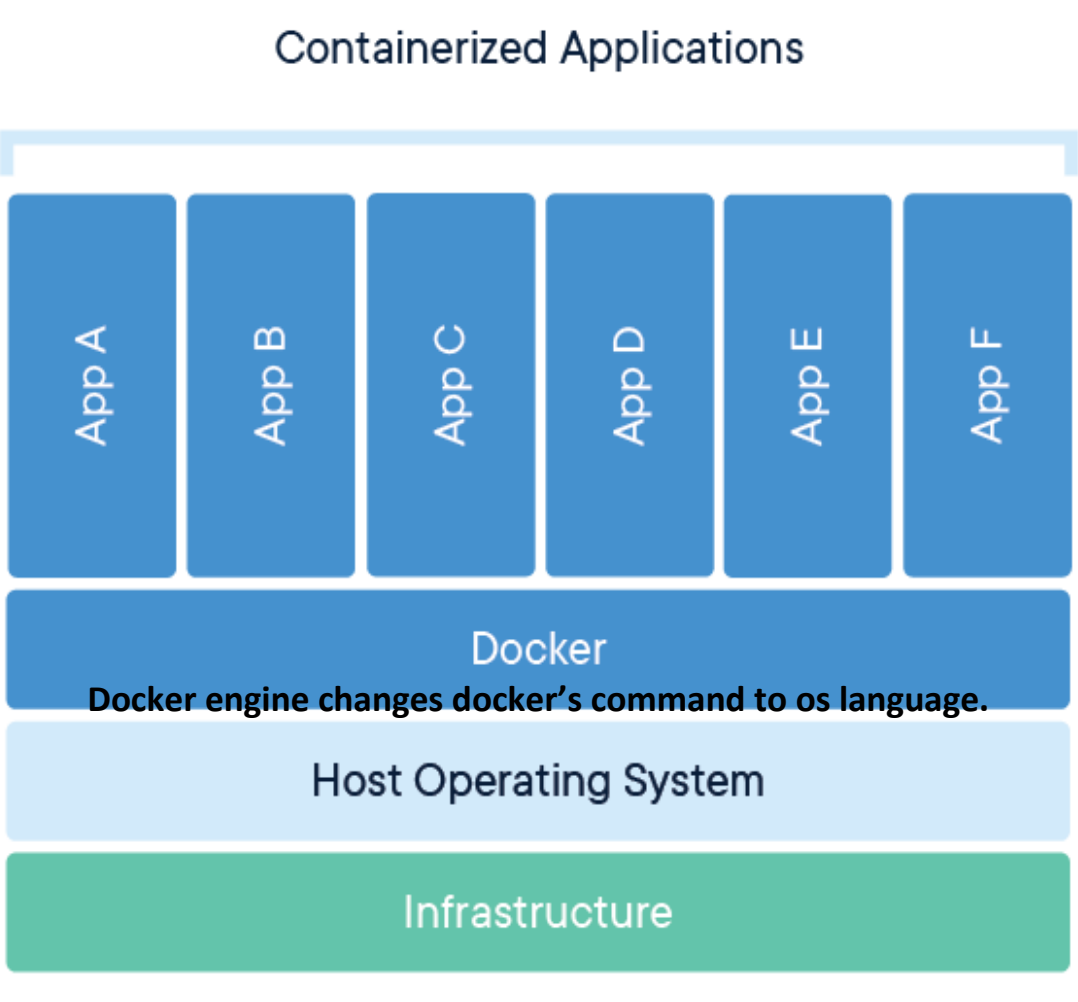
What is docker?

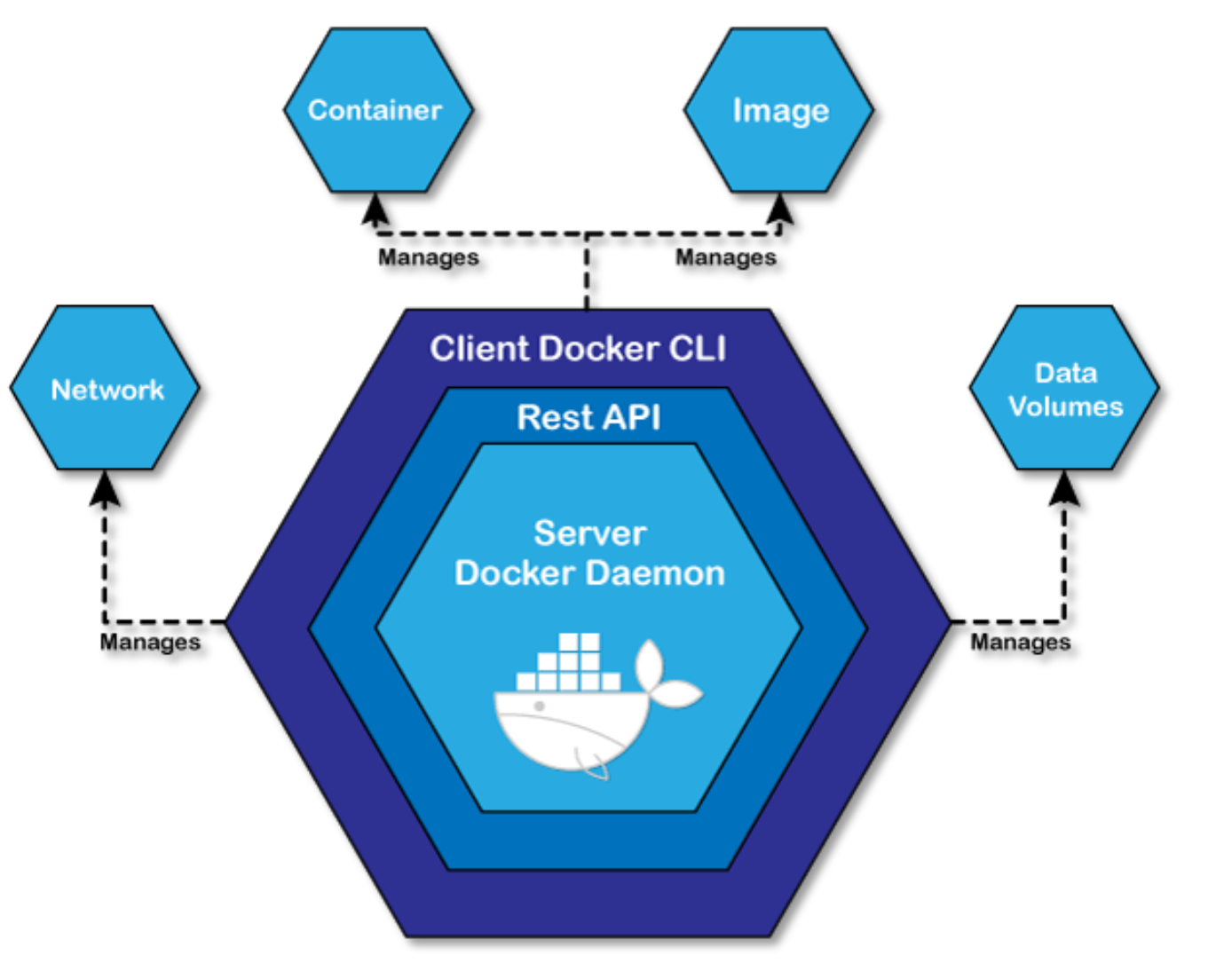
- Docker is an **open source containerization** platform.
- It provides the ability to run applications in an **isolated environment** (**Containers**).
- Containers are like **lightweight virtual machines** that can run directly on our host kernel **without need of a hypervisor**.
- Instead of virtualizing entire physical machine, containers virtualize **host OS only**.

Containers share **host OS kernel**, **binaries** & **libraries** as well.



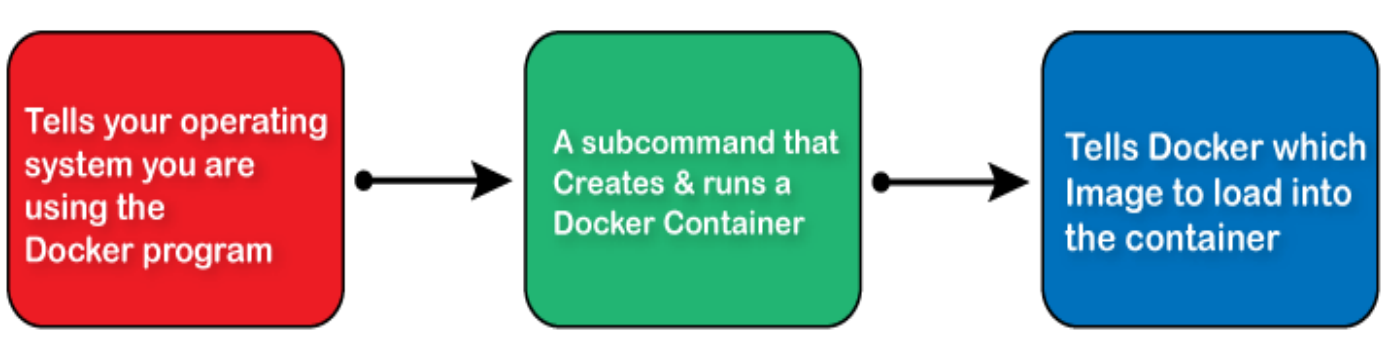
Containerized Applications



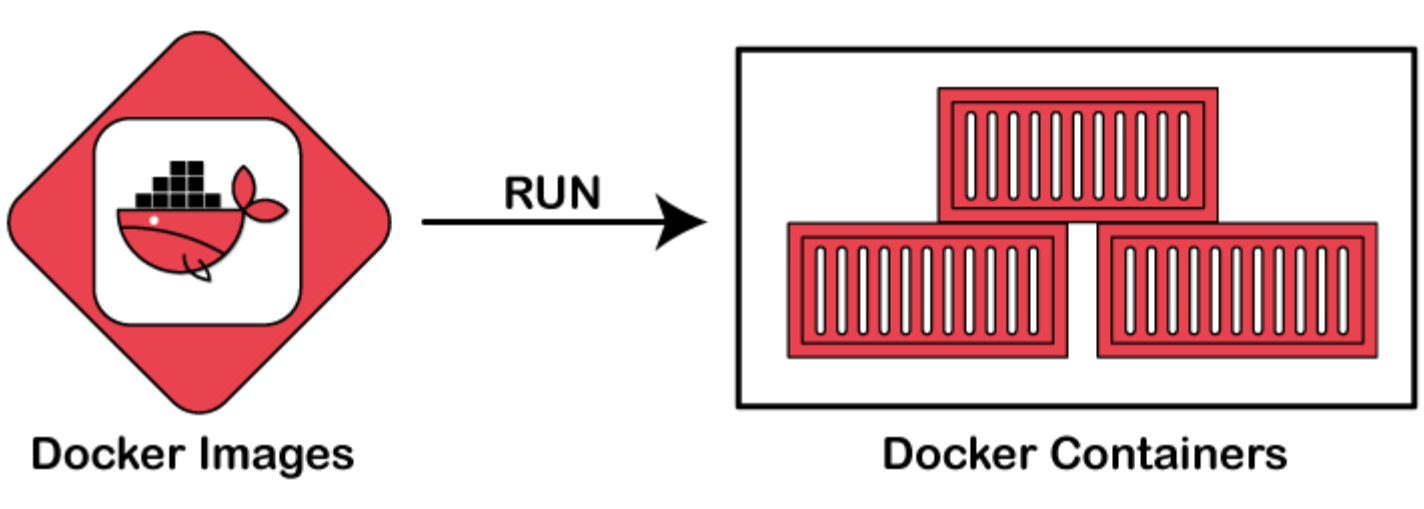


- A **Docker image** acts as a **set of instructions** to **build a Docker container**.
- An image is **comparable** to a **snapshot** in virtual machines.
- Images are created using **`docker build`** command.

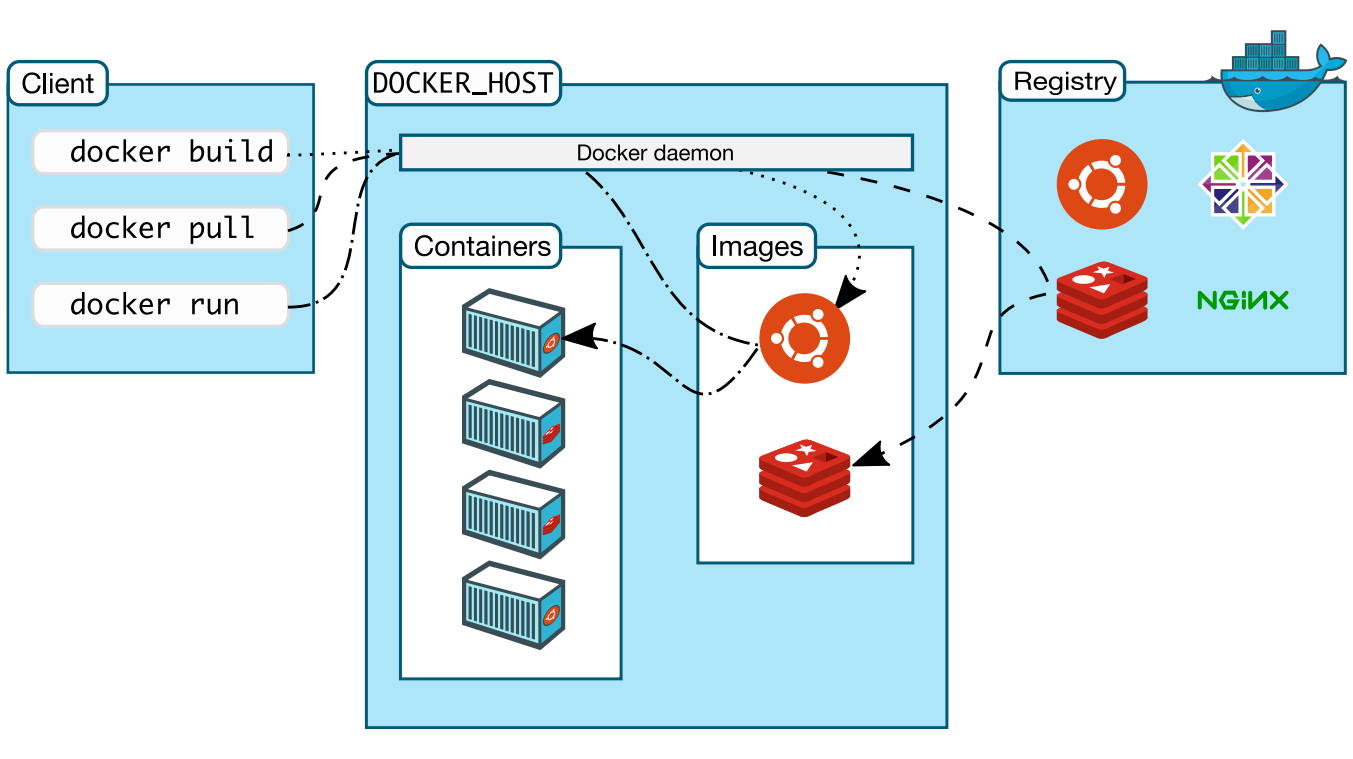
Docker allows you to create and share software using docker images.

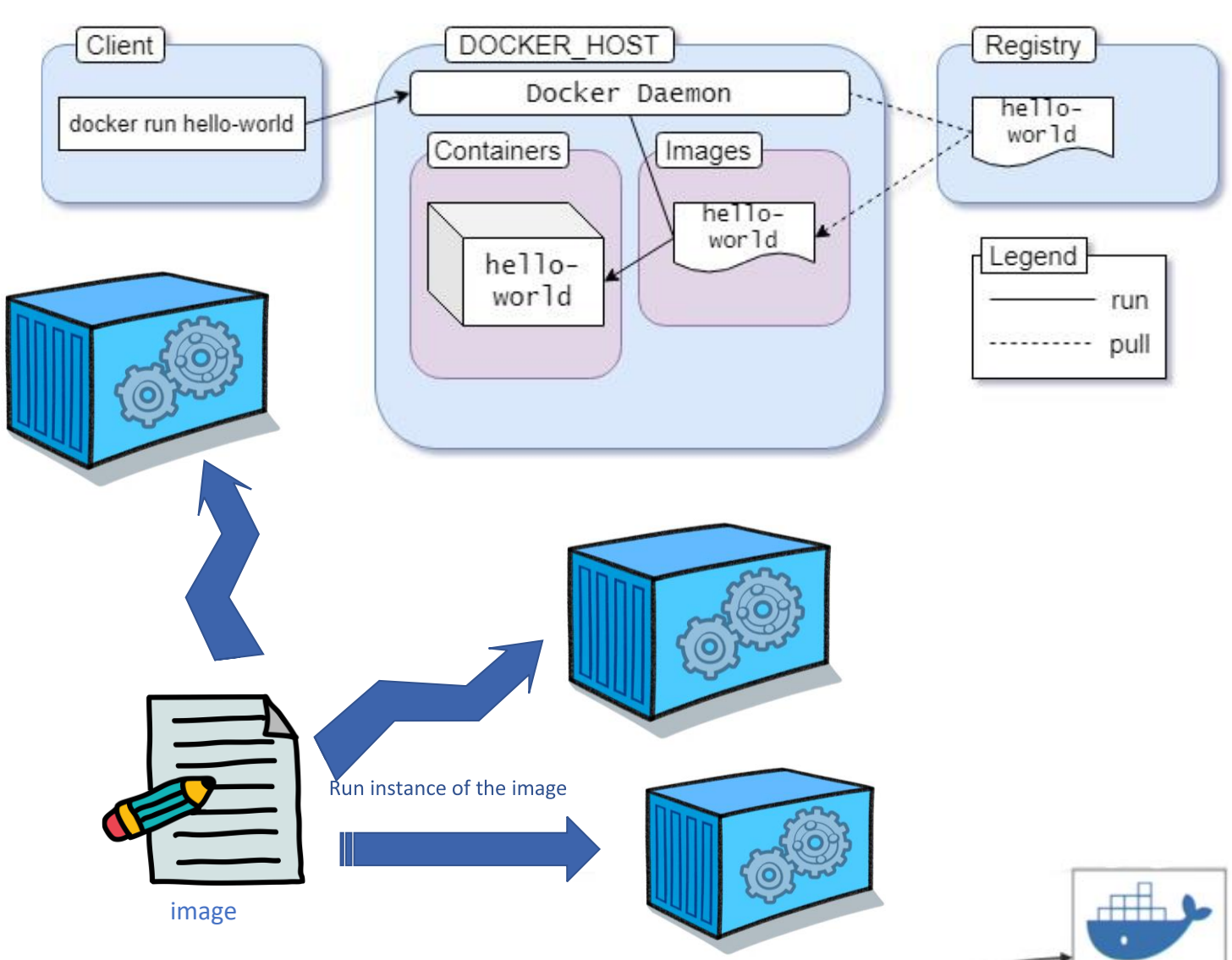


- Docker Container is a runnable instance of images.
- Through Docker API or CLI, we can create or delete a container.
- Containers are by default isolated from each other and their host machine.



The container is isolated environment, but for **networking** it has connection to the OS, so the container has a interface and ip address and it makes a specific **port** listen.
Data volume : bind node's disk to the container.



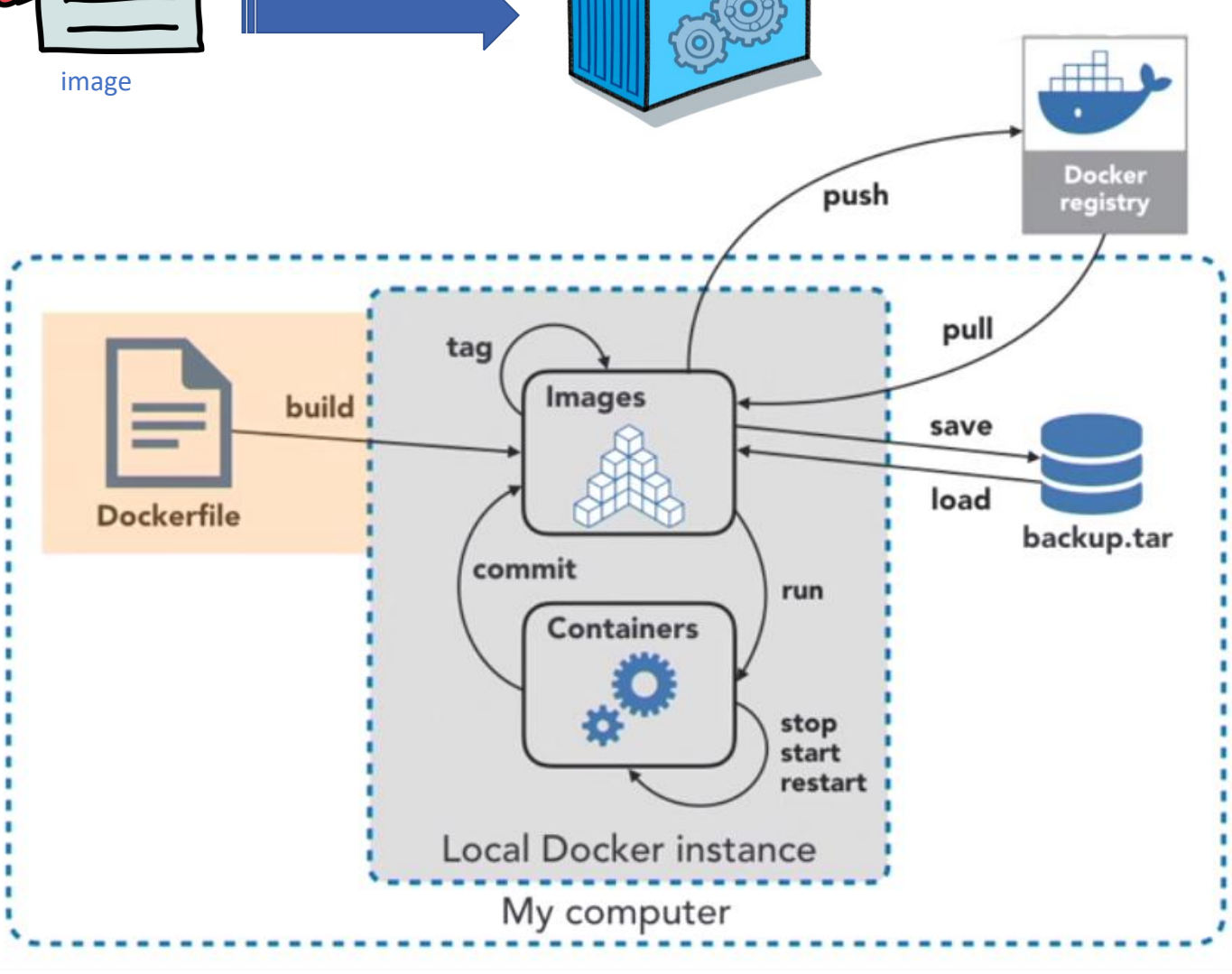


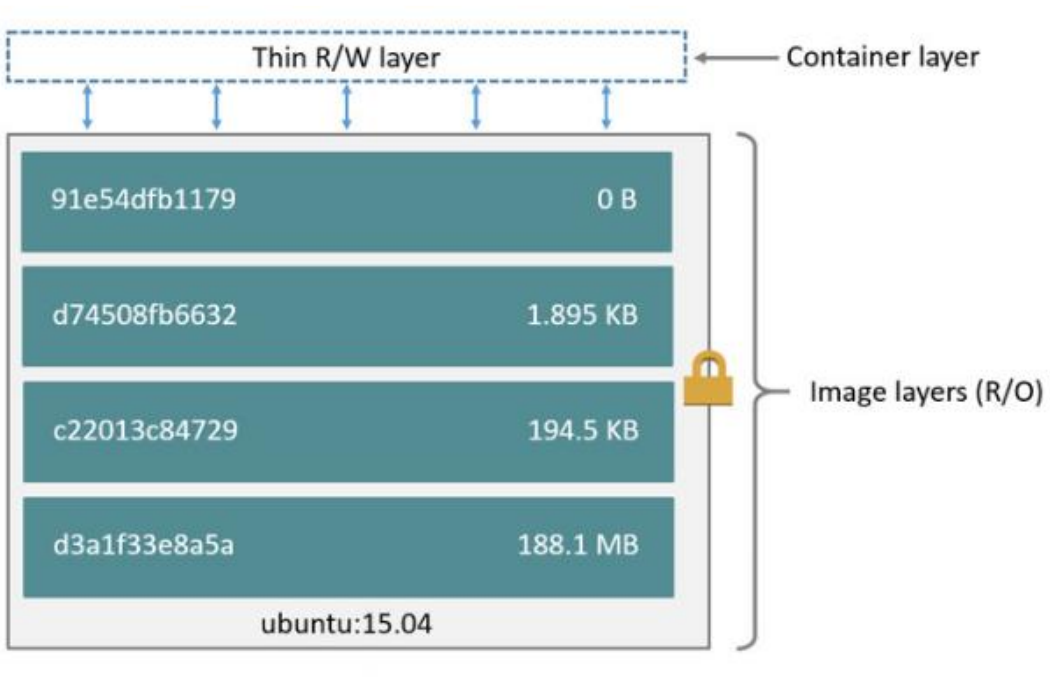
What's a Docker Image?

- **Image** is an **executable package** that includes everything needed **to run** an application.
- It contains **code**, **runtime libraries**, **environment variables** & **config files**.
- **Images** can exist **without containers**, whereas container needs to run an image to exist.
- A **container** is a **runtime instance** of an **image**.
- From **each image**, **hundreds of containers** can be created.

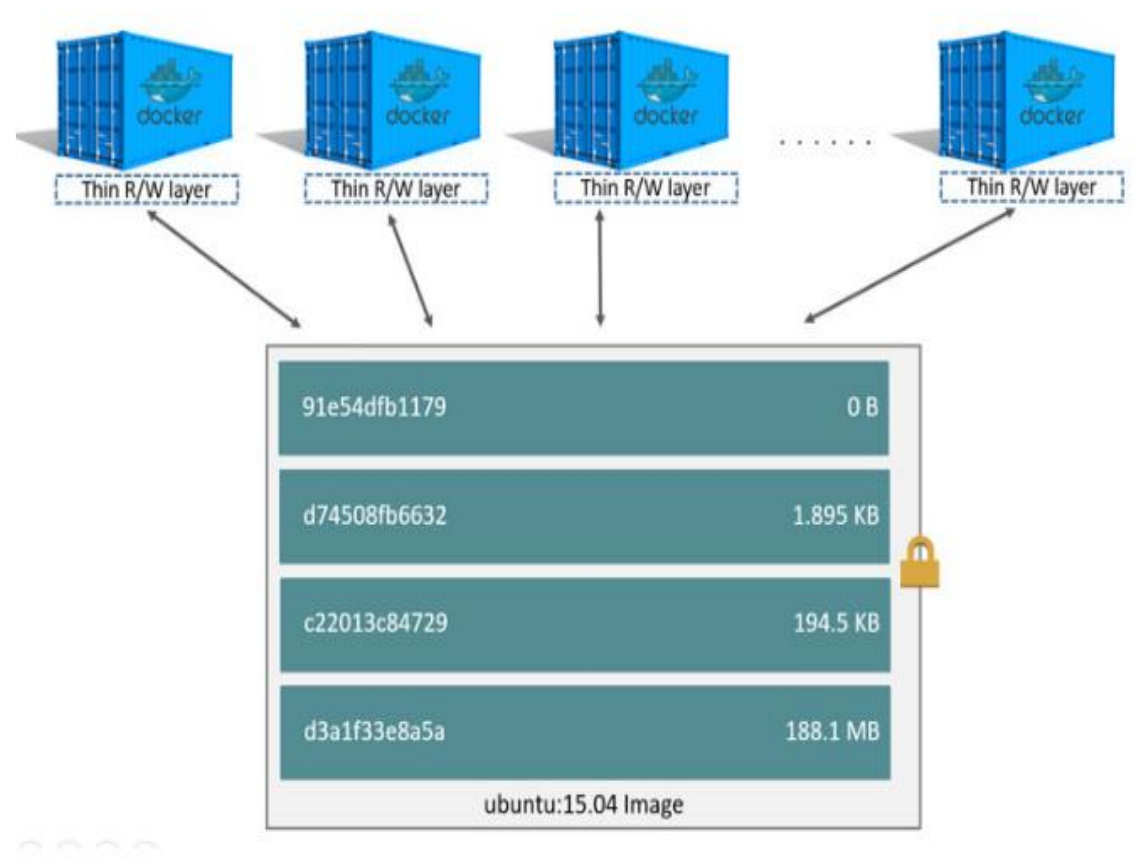
Docker run = docker create + docker start *--create first check the local if it doesn't exist it'll pull the image from repository*
Container has different status : **Created** , **Running(UP)** , **Paused** , **Restarting** , **Exited** , **Dead**
If we use repetitive Ips and Names ,we'd have port and name conflict! *Docker run –rm -d nginx*
Docker run –name zag -d nginx 'd' means detach or demon (start in background)
Zxcvbnhgdyksdoi783bghn container will be start in background whit this id we can exec the container.
Docker run –name zag-nginx -p 8080:80 nginx 8080 for host and 80 inside the container
Docker rmi -f nginx this command makes the container that has been built with this image untagged.
docker **kill** vs **stop**:
stop in **graceful** but kill **terminate** the container
Docker stats
-shows resource usage of each container ,as default if we don't specify limits for the container it'll use **all the resource** and it's fucking dangerous!

Docker run -it ubuntu interactive mode we have shell
Docker run –name first-nginx ubuntu
Docker exec -it skajcdidjsb sh
Docker **export** first-nginx > /home/zag-nginx.tgz
we can save our changes on container in new image and after that we can deploy the image in new server via import command:
Docker **import** -m "zag message" zag-nginx.tgz nginx:latest
Docker save -o [filename].tar [image]
This command used for save an image in a TAR file.
Docker history image-name:version
gives history about the image, when has been created and etc.





- **Each container** has **specified upper layer** that it is Readable/Writable.
- With removing container, **just upper layer** removed & lower ones will remain.
- Because each container has its **own upper layer**, so, **multiple containers** can **share** one image with **its lower layers** together.



Docker ee vs docker ce ?
Docker ee used for cloud environment ,it's cost1500 dollar!
Docker ce used for **bare metal and we used this and it's open source and free** ☺
Docker **CE** (**Community Edition**) is the simple classical OSS (Open Source Software) Docker Engine. Docker **EE** (**Enterprise Edition**) is Docker CE with certification on some systems and support by Docker Inc.
Docker info :
Storage driver: overlay2
Container is isolated
Not in this two : port and volume
Cgroup driver : cgroupfs and systemd
Kernel uses Cgroup for allocate resource to the containers.
Docker root dir : **/var/lib/docker**
Kernel version : jhdkjdhkasjhdak
Insecure registries : novinrepo:8082

Dockerfile Docker Image Docker Container

- Everyone can create **his own images** or use images **created by others**.
- For **build an image**, need to **make a Dockerfile** for creating an image & run it.
- Each **instruction** in a Dockerfile **creates a layer in the image**.
- With changing Dockerfile and **rebuilding image**, only **changed layers** are rebuilt.
- It will make images so **lightweight**, **small** & **fast**.
- **A Dockerfile executed** by **`docker build`**.

Dockerfile syntax

FROM	We specify the base image here.
RUN	We define the **commands** here.
CMD	Used just for **a command.**
COPY	Copy files into the image, preferred over ADD.
ADD	Add files into the image.

Vim Dockerfile
FROM nginx
RUN echo "zag zag zag" > /home/test-zag.log
After creating the image, we can change Just the last layer.
Docker build -tag test-zag:2 .
Because in each level we have specific layer
Docker run -it test-zag:2 sh
#cat /home/test-zag.log
#zag zag zag
We use \ for going next line. It doesn't mean next layer.
We can use Dockerfile for other images.

Vim /etc/hosts
novinrepo ip-address

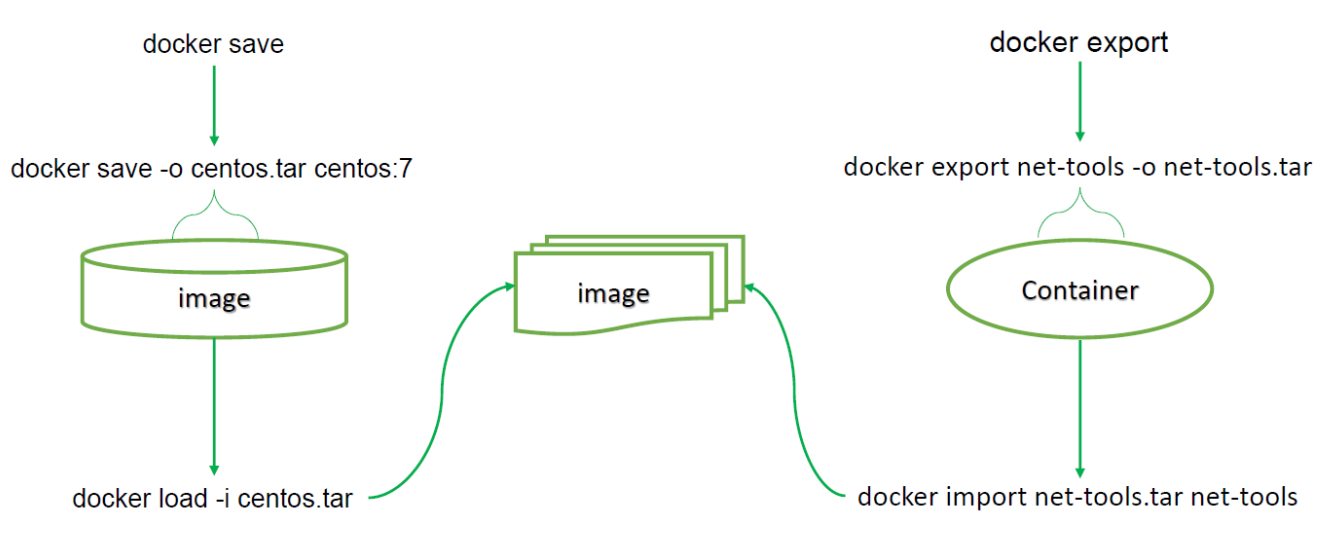
Docker image ls
Novinrepo:8082/docker/nginx-zag:2 first section is for docker registry.

Dangle images : images that don't have repository and tag
Docker image –filter dangling=true

Docker image prune --delete dangling images.
Docker image prune **-a** -- delete images without container
Docker system **prune** -- plus delete stopped container and networks without usage
Docker system prune **-a** -- plus **catch** images
Docker search nginx -- searches in docker-registry for nginx shows starts and official and automated(created automatically)

Docker save nginx > nginx.tar -- for installing the **image** in another server
Docker load -i nginx.tar -- for loading **image**
Docker save nginx | gzip > nginx.tar.gz -- for zipping ☺
Use docker save to save the image to a TAR file.

Note: **export** and **import** used for creating an image from a container. We can send another server.
Docker export first-nginx > /home/zag-nginx.tgz **–we can save our changes in container in new image**
Docker import -m "zag message" zag-nginx.tgz nginx:latest
What is docker commit ?
Use 'docker commit' to commit your running container to a new image. **In your local repository**.
For example you exec the container and made some changes, now you can commit that changes on new image on your machine after that you can do what ever you want ☺

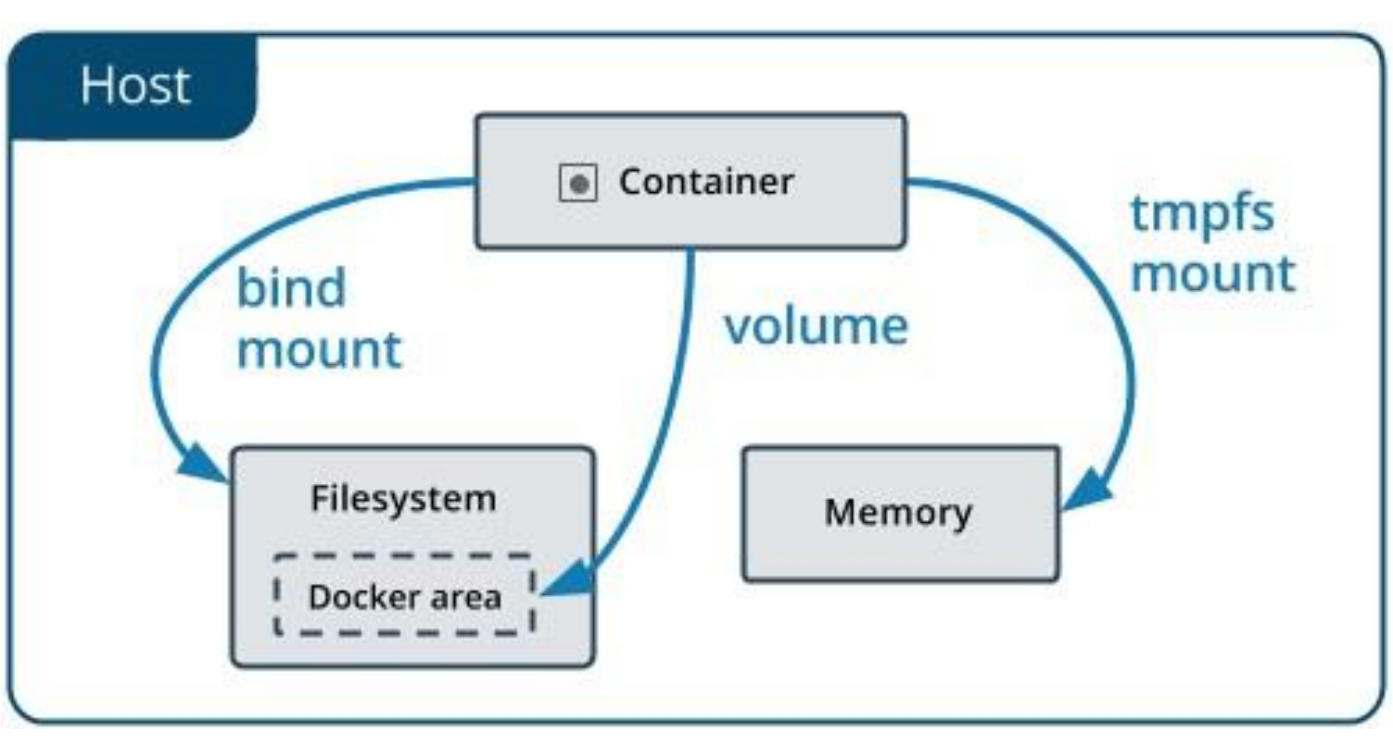


Docker for connecting to the kernel uses some drivers.
For example for using storage:

- **Docker Storage Driver** controls how images & containers are **stored & managed** on host.
- **Docker Engine** provides following **storage drivers** on Linux:

Driver	Description
overlay2	It is preferred storage driver for all currently Linux distros & requires no extra configs.
btrfs & zfs	These storage drivers has advanced options, like "**snapshots**", but require more configs.
vfs	It is used for **testing purposes**. Its performance is poor & not recommended for live use.
aufs	It was preferred storage driver for Docker 18.06 & older on Ubuntu 14.04 & kernel 3.13 which had no support for overlay2.
devicemapper	Devicemapper requires direct-lvm environment. It was preferred storage driver for CentOS & RHEL, as their kernel version did not support overlay2.
overlay	The legacy overlay driver was used for kernels that did not support the "multiple-lowerdir" feature required for overlay2.

Volume:
Used for data in docker.
Applications on docker containers **need data** & data **need to be stored** some where.
Docker can store container data on **Host Hard Disk** or **Memory**:
Volumes: Recommended way to **persist** data, stored at `/var/lib/docker/volumes/`.
Bind Mounts: have **limited functionality** & **exact file path on the host must use.**
tmpfs mounts: Stored only in a host's memory in Linux (least recommended) --for example redis (catching servers -in memory database)



Supported backing filesystem:

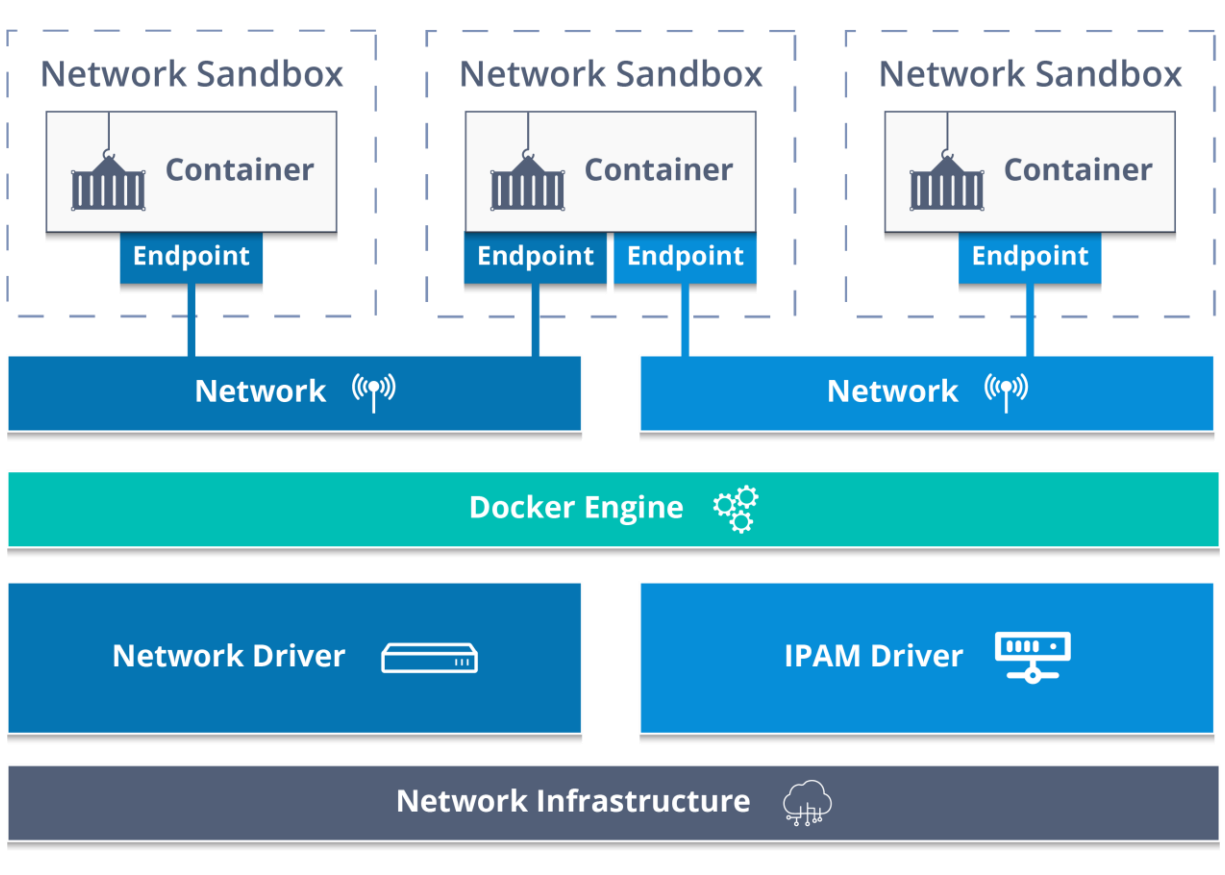
- The **backing filesystem** is the filesystem where `/var/lib/docker/` is **located**.
- Some **storage drivers only** work with **specific backing filesystems**.

Storage Driver	Supported Backing Filesystem
overlay2, overlay	xfs, ext4
aufs	xfs, ext4
devicemapper	direct-lvm
btrfs	btrfs
zfs	zfs
vfs	any filesystem

```
root@ubuntu:~# docker info
  Server:
    Containers: 1
    Images: 4
    Server Version: 20.10.7
    Storage Driver: overlay2
      Backing Filesystem: extfs
      Supports d_type: true
      Native Overlay Diff: true
      userxattr: false
```

Volume commands:
Docker volume create zag-volume
ll /var/docker/volumes/zag-volume/data
zag-volume
We can create a container and mount this volume to it.
Docker run -d –name zag-nginx –volume zag-volume:/app '–volume' this create volume if doesn't exist.
Now we shared this path /var/docker/volumes/zag-volume/data with /app inside the container.
We can share specific path between one or more containers.
We use volume:
-For injection the data into container.
-Save the data from the container.
-Share data between containers.
Docker inspect shows information and you can see the path that has been mounted for the container.
Docker volume ls
Docker volume rm zag-first we can remove a volume which is unused.

This is bind mount
Docker run -d –name zag –volume /home:/app nginx



Docker networking:

Docker network drivers:
There are 5 types of network drivers: **Bridge**, **Host**, **None**, **Overlay**, **Macvlan**, :
Bridge:

- The **private default network driver**.
- All containers get an **internal IP**, So they can **access** each other.
- Bridge networks usually used when apps run in **standalone containers** that need to communicate.

ip a s
docker0 172.17.0.1/16 the containers will receive ip addr in this range
Because docker has internal dns-service (built-in) we can call the name for example in ping command.
We can assign two IPs for a particular container(and then we can use it as router after defining rout table):
-we must create new network with driver bridge (don't use default network bridge)
-docker network create –driver bridge net-zag
-docker network **connect** net-zag second-zag
Host:
It uses host's network directly. So ports are unique and we can't give different ports.
Uses host's network and ports in container starts same port on the host.
Overlay:

- Overlay networks connect **multiple Docker daemons** together.
- Only available with **Docker EE** & **Docker Swarm** enabled.

Docker ee vs docker ce ?
Docker ee used for cloud environment ,it's cost1500 dollar!
Docker ce used for bare metal and we used this and it's open source and free ☺
Docker CE (Community Edition) is the simple classical OSS (Open Source Software) Docker Engine. Docker EE (Enterprise Edition) is Docker CE with certification on some systems and support by Docker Inc.
Macvlan:

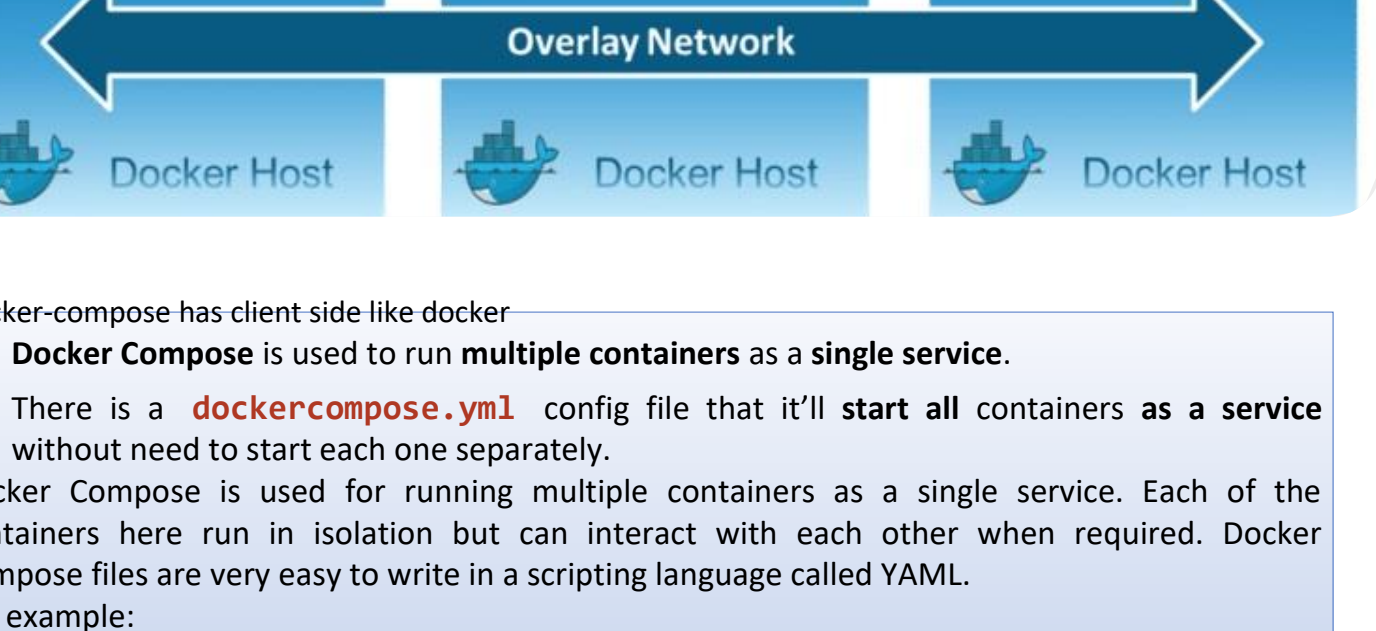
- Macvlan networks allow you to **assign a MAC address to a container**, making it appear **as a physical device** on network.
- Docker daemon **routes traffic** to containers by **their MAC addresses**.

None:

- For this container, **disable all networking**.

Docker network ls
Docker inspect bridge it gives information and range for our network.
Docker network create –driver bridge zag-network
Ip a we have new virtual network ip
Docker run –rm -d –**network** zag-network –name zag-container redis:v1
In this new container we don't have ping for others containrs that have been created in default network.
Docker network ls if we start a container it'll receive IP in this range.
We can publish same ports in different containers.
ip addr show for watching type of driver docker creates a virtual network.
For deleting a network we must first delete the container that assigned to it.
Docker network remove zag-network.
Docker inspect ZAG
Docker top CONTAONER-ID
shows usage of specific container
Docker stats
shows memory and CPU that docker are using.
Nginx1 0.1/8G **default allocate whole the resource**
Limit and **Request** in Kubernetes:
Limit how much resource a specific pod can use?
Request how much resource should allocate to the pod ☺
Docker run -d –name zag-memory –memory 200m nginx
Docker stats
Zag-memory 0.1/200m
Docker engine uses **Cgroup** to allocate resource to the container (**cgroupfs** and **systemd**)

Docker run –rm -d –name nginx-num6 **-m 200m** -P -h zaaaaag-name-host nginx
200 MiB
-P expose random port 32768->80 -p specific port : 8080:80
-h change hostname



Docker-compose has client side like docker

- **Docker Compose** is used to run **multiple containers** as a **single service**.
- There is a `dockercompose.yml` config file that it'll **start all** containers **as a service** without need to start each one separately.

Docker Compose is used for running multiple containers as a single service. Each of the containers here run in isolation but can interact with each other when required. Docker Compose files are very easy to write in a scripting language called YAML.
For example:
If you have an application that requires an NGINX server and Redis database, you can create a Docker Compose file that can run both the containers as a service without the need to start each one separately.

```
Services:
      nginx:
        image : nginx
        ports:
          -8080:80
        restart : always
Docker-compose up -d
Docker-compose ps
Docker-compose logs -f
```