



# Docker deep dive

Prepared by: Aamir Pinger



[fb.com/AamirPingerOfficial](https://fb.com/AamirPingerOfficial)

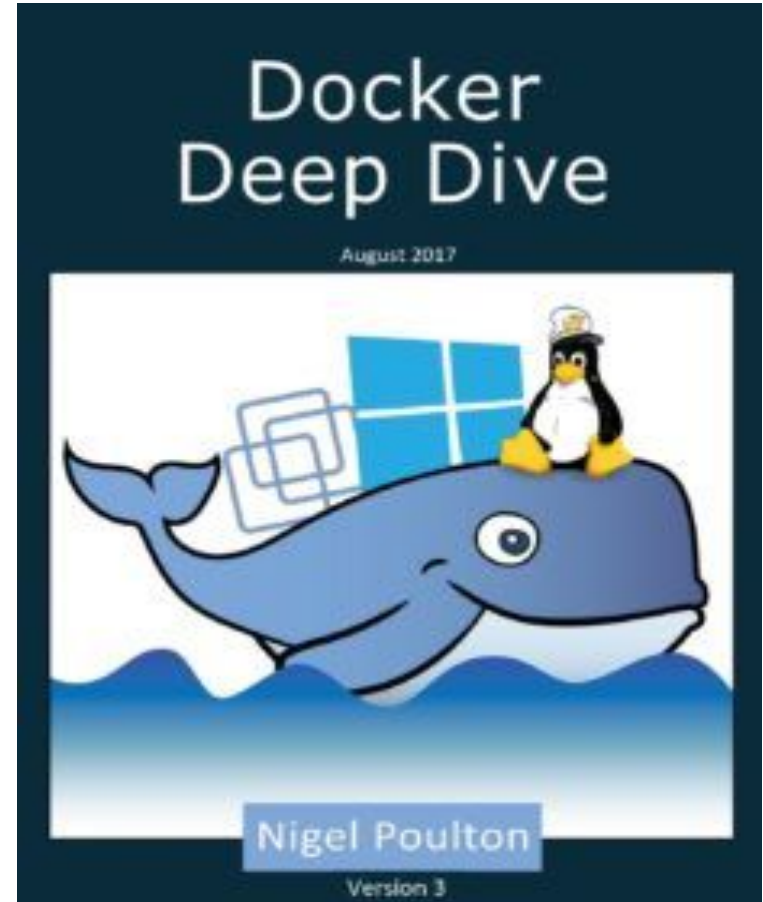


[github.com/AamirPinger](https://github.com/AamirPinger)



[linkedin.com/in/AamirPinger](https://linkedin.com/in/AamirPinger)

## BOOK TO FOLLOW



# THE BAD OLD DAYS

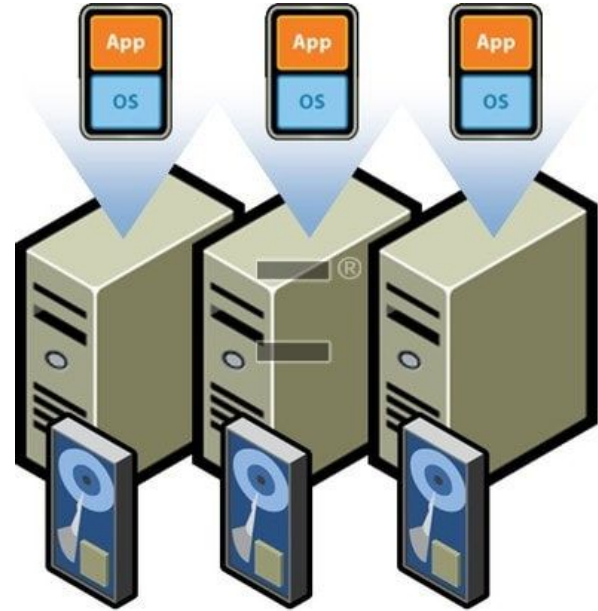
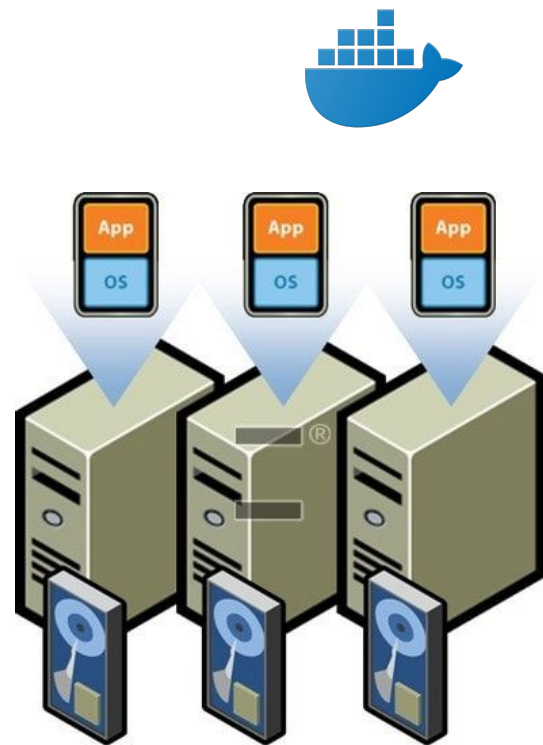


Image source:

[https://www.thomas-krenn.com/en/products/application/virtualization/virtualizing-with-vmware/vmware-server-systems/functionality\\_and\\_construction.html](https://www.thomas-krenn.com/en/products/application/virtualization/virtualizing-with-vmware/vmware-server-systems/functionality_and_construction.html)

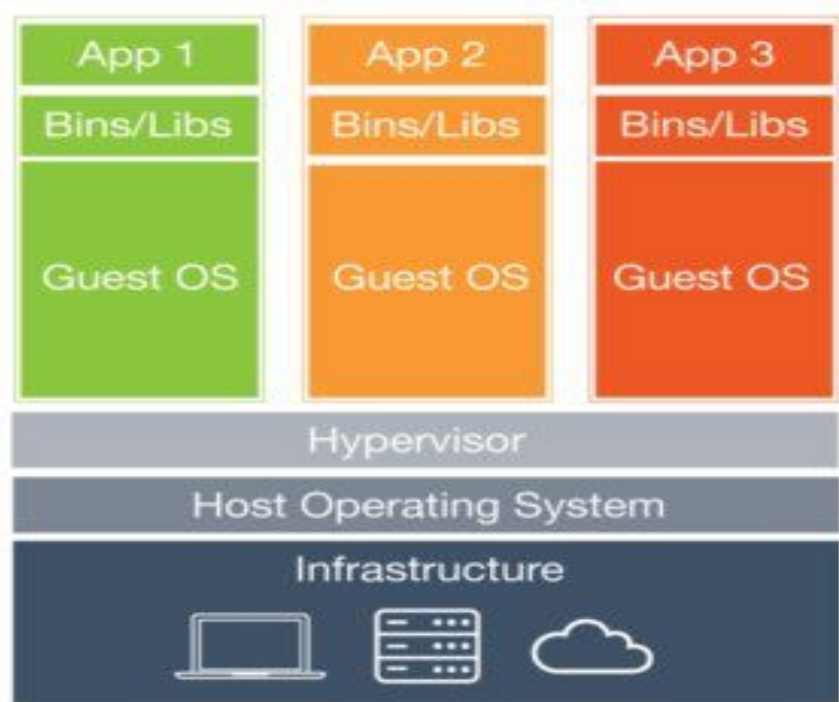
# THE BAD OLD DAYS

- Usually one app one server rule
- Reason were
  - Unable to judge resource requirements
  - Different infrastructure and dependencies
- Disadvantages
  - Very costly
  - Resource wastage
  - Many servers to manage





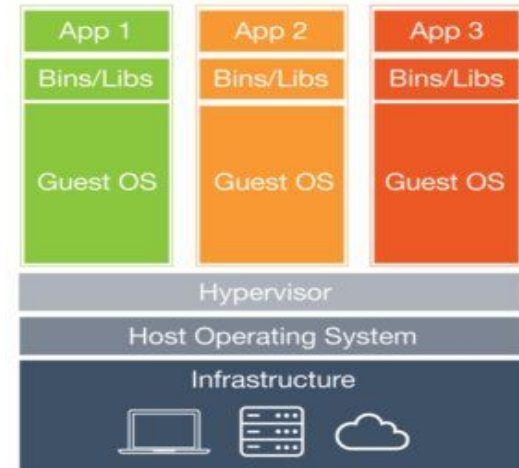
# VMware



# VMware



- VMware is a company that was established in 1998 and provides different applications for virtualization
- VMware Workstation, introduced in 1999, was the first product launched by VMware
- Benefits came with virtualization
  - Multiple app on single server
  - Different OS and dependencies on same server using VMs
  - Much better than old one
  - Saves lot of resources
- Disadvantages
  - OS consumes lots of resources
  - Licensing cost of every OS instance



# HYPERVISOR

---



- A hypervisor is a function which abstracts -- isolates -- operating systems and applications from the underlying computer hardware
- This abstraction allows the underlying host machine hardware to independently operate one or more virtual machines as guests
- This makes it possible for multiple guest VMs to effectively share the system's physical compute resources, such as
  - Processor cycles,
  - Memory space,
  - Network bandwidth and so on
- A hypervisor is sometimes also called a virtual machine monitor

# CONTAINERS



Image source:

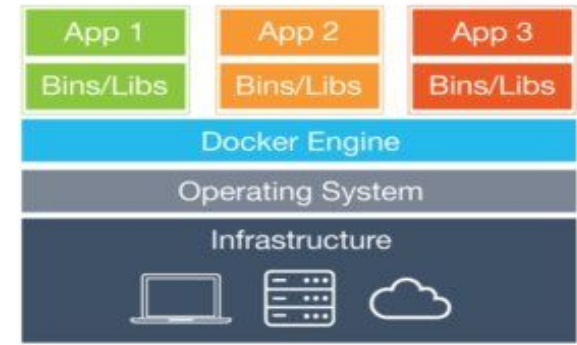
[https://www.monitis.com%2Fblog%2Ftop-5-benefits-of-containerization%2F&psig=AOvVaw3l6VvAu\\_TMOaxYS4DZBNbn&ust=1556006137919830](https://www.monitis.com%2Fblog%2Ftop-5-benefits-of-containerization%2F&psig=AOvVaw3l6VvAu_TMOaxYS4DZBNbn&ust=1556006137919830)



# CONTAINERS



- Container technology, also known as just a container, is a method to package an application so it can be run, with its dependencies, isolated from other processes
- For a long time, the big web-scale players like Google have been using container technologies to address these shortcomings of the VM model
- Major difference between containers and VM model
  - Single OS
    - All containers on a single host share a single OS
  - Less Hardware resource
    - This frees up huge amounts of system resources such as CPU, RAM, and storage



Source2: Book Docker Deep Dive v4 by Nigel Poulton

Source1: <https://www.infoworld.com/article/3204171/docker/what-is-docker-docker-containers-explained.html>

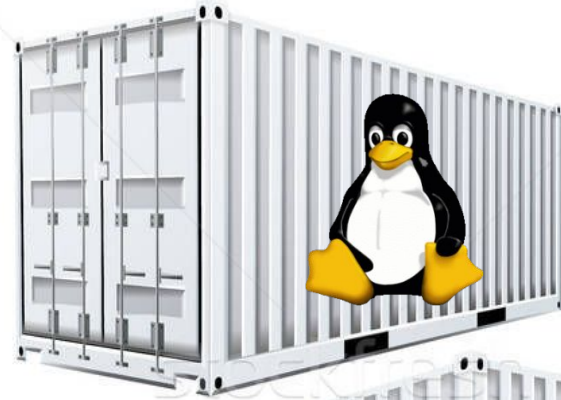
# CONTAINERS (cont...)



- Licence fee
  - It also reduces potential licensing costs and reduces the overhead of OS patching and other maintenance
- Portable and fast
  - Containers are also fast to start and ultra-portable
- Once worked will work everywhere
  - Moving container workloads from your laptop, to the cloud, and then to VMs or bare metal in your data center is a breeze
- Containers result in savings on the cap-ex (Money) and op-ex (Human resource) fronts

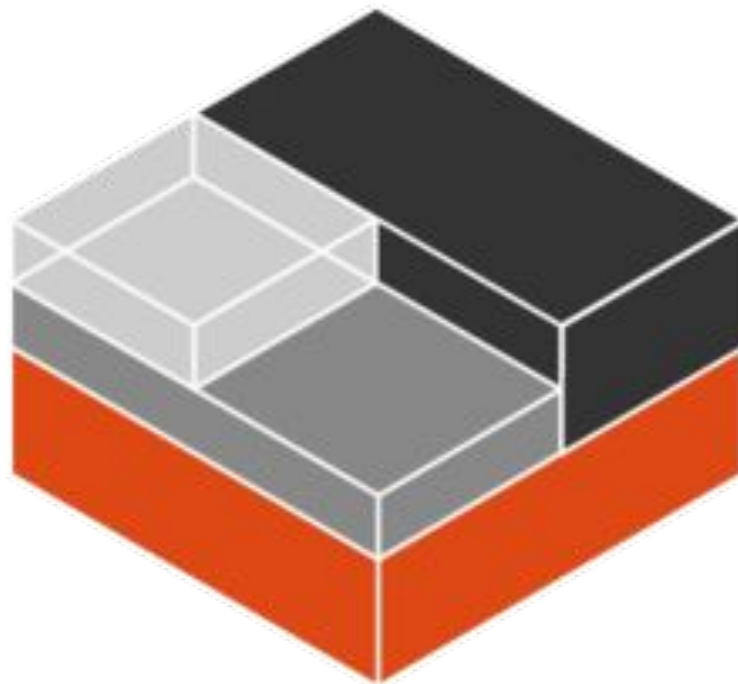
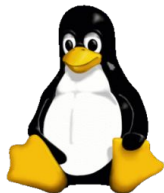


# TYPE OF CONTAINERS





# LINUX CONTAINERS



# LINUX CONTAINERS

---



- Container technology is not a new phenomenon, and has long been a core feature for Linux
- The advance in recent years of container technology it has become easier to use, and software developers have embraced them for their simplicity, and avoiding compatibility problems
- They also enable a program to be broken down into smaller pieces, which are known as microservices
- Just as one example, Google Inc. has contributed many container-related technologies to the Linux kernel
- Google is not alone, hundreds of individual developers and companies have contributed to linux kernel
- Without these contributions, we wouldn't have modern containers today



# WINDOWS CONTAINERS



# WINDOWS CONTAINERS

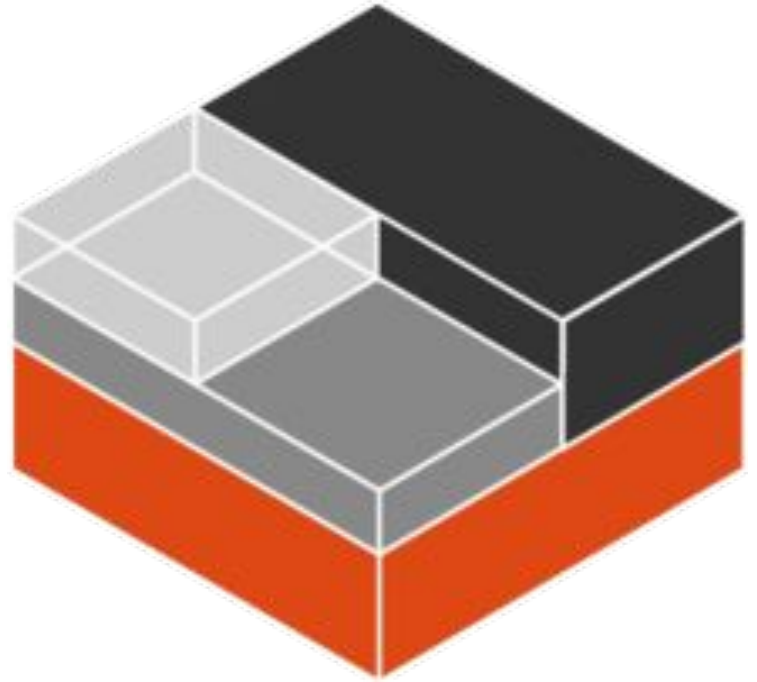
---



- Over the past few years, Microsoft Corp. has worked extremely hard to bring Docker and container technologies to the Windows platform
- The core Windows technologies required to implement containers are collectively referred to as Windows Containers
- The user-space tooling to work with these Windows Containers is Docker
- This makes the Docker experience on Windows almost exactly the same as Docker on Linux
- This way developers and sysadmins familiar with the Docker toolset from the Linux platform will feel at home using Windows containers



# MAC CONTAINERS







# MAC CONTAINERS

---

- There is currently no such thing as Mac containers
- However, you can run Linux containers on your Mac using the Docker for Mac product
- This works by seamlessly running your containers inside of a lightweight Linux VM running on your Mac
- It's extremely popular with developers who can easily develop and test their Linux containers on their Mac



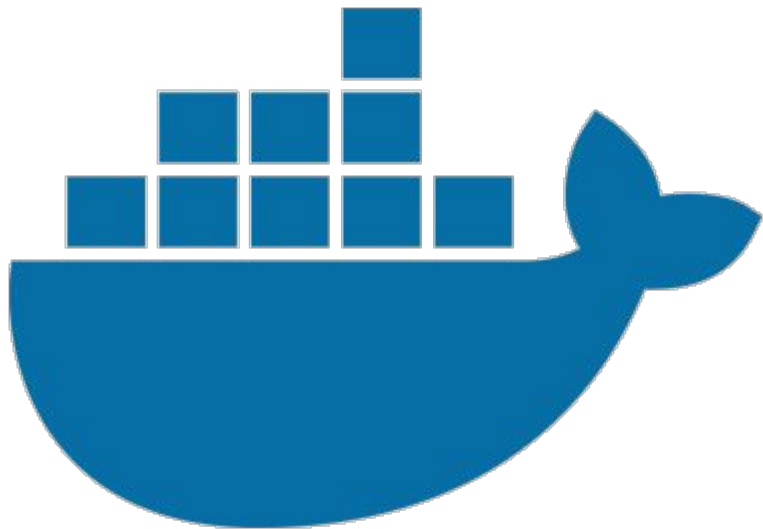
# TYPE OF CONTAINERS (cont...)

---

- It's vital to understand that a running container uses the kernel of the host machine it is running on
- This means that a container designed to run on a host with a Windows kernel will not run on a Linux host
- This means that you can think of it like this at a high level - Windows containers require a Windows Host, and Linux containers require a Linux host
- However, it is possible to run Linux containers on Windows machines. For example, Docker for Windows (a product offering from Docker, Inc. designed for Windows 10) can switch modes between Windows containers and Linux containers



# DOCKER





# IS DOCKER A CONTAINER?

---

- No!
- Docker is software that runs on Linux and Windows
- It creates, manages and also has the ability to orchestrate containers
- The software is developed as part of the Moby open-source project on GitHub
- Docker, Inc. is a company based out of San Francisco and is the overall maintainer of the open-source project
- Despite all of this, containers remained complex and outside of the reach of most organizations
- It wasn't until Docker came along that containers were effectively democratized and accessible to the masses
- It's would be not wrong to say that Docker made containers simple!

# DOCKER (cont...)

---



- When most technologists talk about Docker, they're referring to the Docker Engine
- The Docker Engine can be downloaded from the Docker website or built from source from GitHub
- It's available on Linux and Windows, with open-source and commercially supported offerings
- Docker, Inc. also has offers commercial versions of Docker with support contracts etc
- There two main editions:
  - Enterprise Edition (EE)
  - Community Edition (CE)




# THE CONTAINER ECOSYSTEM

# THE CONTAINER ECOSYSTEM

---



- One of the core philosophies at Docker, Inc. is often referred to as Batteries included but removable
- This is a way of saying you can swap out a lot of the native Docker stuff and replace it with stuff from 3rd parties
- A good example of this is the networking stack that help you connect Docker container together, or connect them to non-Docker workloads
- The core Docker product ships with built-in networking. But the networking stack is pluggable meaning you can rip out the native Docker networking and replace it with something else from a 3rd party



# THE OPEN CONTAINER INITIATIVE (OCI)





# THE OPEN CONTAINER INITIATIVE (OCI)



- The OCI is a relatively new governance council organized under the auspices of the Linux Foundation responsible for standardizing the most fundamental components of container infrastructure such as image format and container runtime
- Uptil now the OCI has published two specifications (standards) -
  - The image-spec
  - The runtime-spec
- An analogy that's often used when referring to these two standards is rail tracks
- Nobody wants two competing standards for rail track sizes
- These two standards are like agreeing on standard sizes and properties of rail tracks
- Leaving everyone else free to build better trains, better carriages, better signalling systems, better stations... all safe in the knowledge that they'll work on the standardized tracks





# INSTALLING DOCKER





# INSTALL DOCKER CE ON UBUNTU

- Update the apt package index

```
aamir@ap-linux:~$ sudo apt-get update
```

- Install the latest version of Docker CE and containerd

```
aamir@ap-linux:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```



# **POST INSTALLATION ACTIVITIES**



# POST INSTALLATION ACTIVITIES



```
aamir@ap-linux:~$ docker --version
```

```
Docker version 18.09.2, build 6247962
```

```
aamir@ap-linux:~$ docker info
```

```
Containers: 18
```

```
Running: 0
```

```
Paused: 0
```

```
Stopped: 18
```

```
Images: 59
```

```
Server Version: 18.09.2
```

```
Storage Driver: overlay2
```

```
Backing Filesystem: extfs
```

```
Supports d_type: true
```

```
Native Overlay Diff: true
```

```
...
```



# POST INSTALLATION ACTIVITIES

- When you install Docker, you get two major components:
  - the Docker client
  - the Docker daemon (sometimes called “server” or “engine”)
- Use the following command to check if client and server is working

```
aamir@ap-linux:~$ docker version
```

## Client:

```
Version:      18.09.2
API version:   1.39
Go version:    go1.10.4
Git commit:    6247962
Built:         Tue Feb 26 23:52:23 2019
OS/Arch:       linux/amd64
Experimental:  false
```

## Server:

```
Engine:
Version:      18.09.2
API version:   1.39 (minimum version 1.12)
Go version:    go1.10.4
```

```
...
```

---

# DOCKER ENGINE



# DOCKER ENGINE

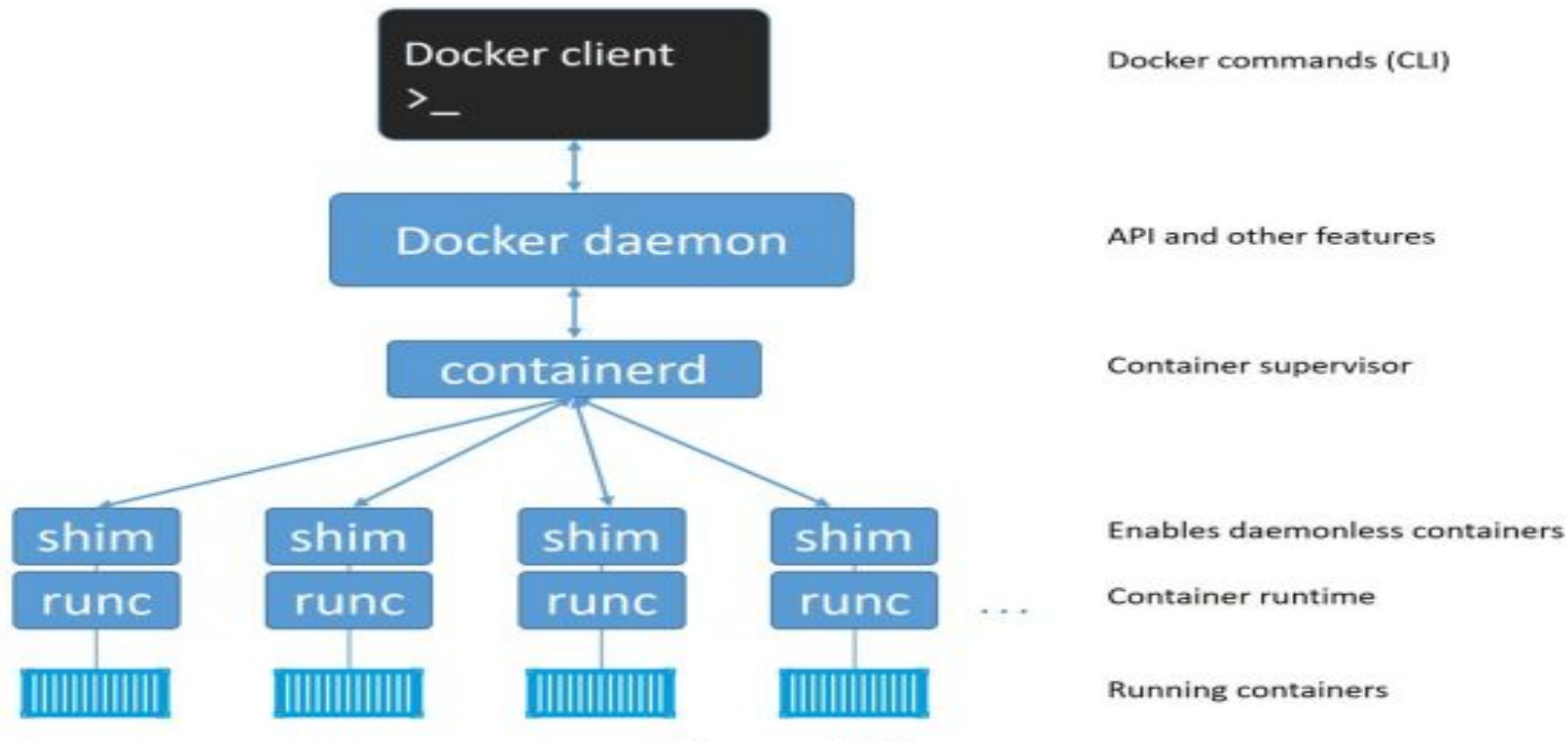
---



- The Docker engine is the core software that runs and manages containers
- We often refer to it simply as Docker, or the Docker platform
- The Docker engine is modular in design with many swappable components. Where possible, these are based on open-standards outlined by the Open Container Initiative (OCI).
- The Docker Engine is made from many specialized tools that work together to create and run containers - images, APIs, execution driver, runtime etc
- The major components that make up the Docker engine are: the Docker client, the Docker daemon, containerd, and runc. Together, these create and run containers



# DOCKER ENGINE



# DOCKER ENGINE



## The Docker daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes

## runc

- runc often refer as a container runtime
- It has a single purpose in life is to create containers

## shim

- The shim is integral to the implementation of daemonless containers
- shim makes it possible to perform maintenance and upgrades on the Docker daemon without impacting running containers!

# DOCKER ENGINE

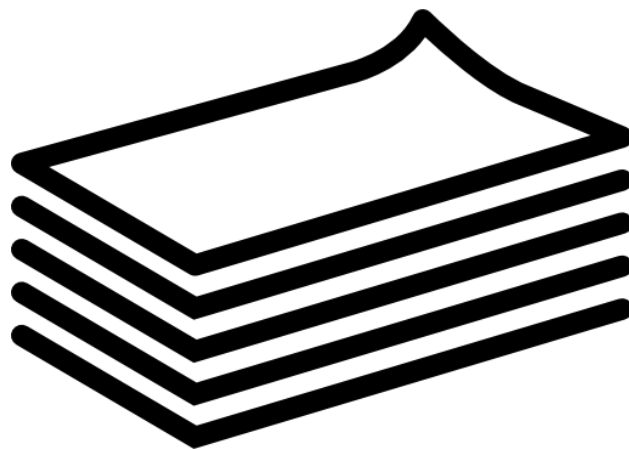


## containerd

- containerd act as a bridge between the daemon and runc
- It's helpful to think of containerd as a container supervisor - the component that is responsible for container lifecycle operations such as;
  - Starting and stopping containers
  - Pausing and un-pausing them
  - Destroying the containers
- Containerd is designed for a single task in life containerd is only interested container lifecycle operations



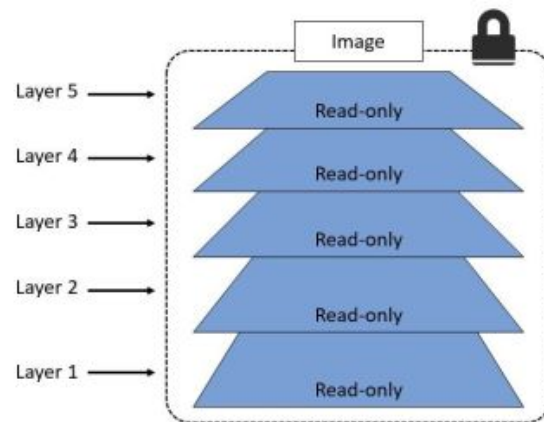
# IMAGES



# IMAGES



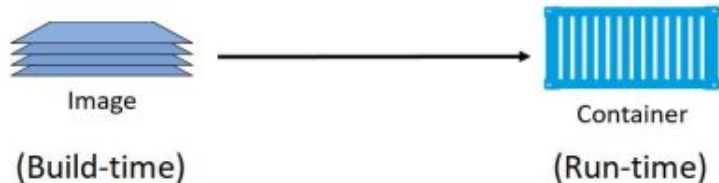
- A Container Image is a lightweight, standalone, executable package of software that includes everything needed to run an application:
  - Code
  - Runtime
  - System tools
  - System libraries
  - Settings
- Images become containers when they run on Docker Engine
- Images are made up of multiple layers that get stacked on top of each other and represented as a single object



# IMAGES



- Inside of the image is A cut-down operating system (OS) and all of the files and dependencies required to run an application
- In this way, each layer contains different things required to run a containerized app
- Common layers among different images are downloaded only once and are stored only once and get re-use in all images
- You build container based on images and that is why images are sometimes called stopped containers
- You can also create images from actual stopped containers



# IMAGES

---

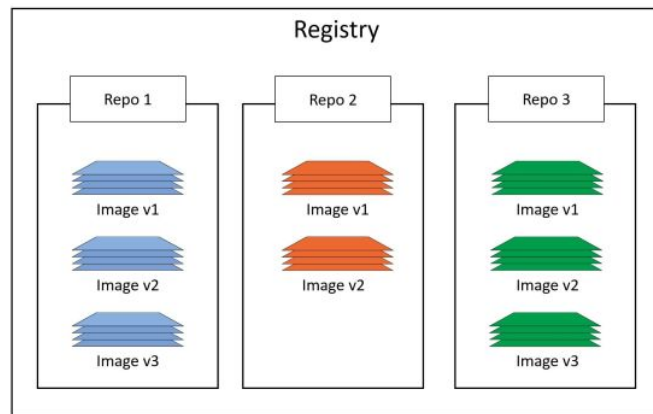


- Once a container is up and running made from an image, the two constructs become dependent on each other and you cannot delete the image until the last container using it has been stopped and destroyed
- The whole purpose of a container is to run an application
- However, containers are all about being fast and lightweight
- This means that the images they're built from are usually small and stripped of all non-essential parts
- The official Alpine Linux Docker image is about 4MB in size and is an extreme example of how small Docker images can be
- Official Ubuntu Docker image which is currently about 120MB. These are clearly stripped of most non-essential parts

# IMAGE REGISTRIES



- Docker images are stored in image registries
- There are many registries options are available, the most common registry is Docker Hub (<https://hub.docker.com>)
- Docker client is also opinionated and defaults to using Docker Hub
- Image registries contain multiple image repositories
- image repositories can contain multiple images
- It is similar that you take a example of a room, files racks and File in it, consider
  - Room as Registry
  - File racks as repositories
  - Files as images







# IMAGE REGISTRIES

---

- Docker Hub also has the concept of official repositories and unofficial repositories
- Official repositories contain images that have been critically examined and verified by Docker, Inc
- Most of the popular operating systems and applications have their own official repositories on Docker Hub
- Unofficial repositories are repository of normally a general users like us
- You can visit [www.hub.docker.com](https://www.hub.docker.com) and explore the list of official images and even sign up your own account which we will be needing to push images you created
- You can then push (upload) images you created to this account and then pull (download) it from anywhere to use it
- You can even make your uploaded images private which will not let anyone else pull it from you account



# IMAGE NAMING AND TAGGING

- Addressing images from official repositories is as simple as giving the repository name and tag separated by a colon ( : )

`docker image pull <repository>:<tag>`

**For example:** `docker image pull nginx:latest`

- Pulling images from an unofficial repository is almost the same. You just need to add Docker Hub username before repository name

`docker image pull username/<repository>:<tag>`

**For example:** `docker image pull aamirpinger/helloworld:latest`

- Two important thing to remember
  1. If you do not specify an image tag after the repository name, Docker will assume you are referring to the image tagged as latest
  2. The latest tag doesn't have any magical powers! Just because an image is tagged as latest does not guarantee it is the most recent image in a repository!



# PULLING IMAGES

---

- A cleanly installed Docker host has no images in its local repository
- To pull an image we will use following command

```
aamir@ap-linux:~$ docker image pull alpine:latest
latest: Pulling from library/alpine
e7c96db7181b: Pull complete
Digest: sha256:769fddc7cc2f0a1c35abb2f91432e8beecf83916c421420e6a6da9f8975464b6
Status: Downloaded newer image for alpine:latest
```

- You can also use **docker pull alpine:latest** instead of **docker image pull alpine:latest**

# LISTING IMAGE



- Now to see what images you have in your system

```
aamir@ap-linux:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	055936d39205	15 hours ago	5.53MB
alpine	3.9.4	055936d39205	2 days ago	5.53MB

- You can also use **docker images** instead of **docker image ls**
- The image ID of any image will be unique and also remain same even if we create new image with different tag



# REMOVING IMAGE

- We have learned that images are use for creating a container
- We have also learned that from one particular image we can create multiple containers that runs as an instance of that image
- One important thing to remember that once you've started a container from an image, the two constructs become dependent on each other
- You cannot delete the image until the last container using it has been stopped and destroyed

```
aamir@ap-linux:~$ docker image rm alpine:latest
Untagged: alpine:latest
```

- You can now confirm that the above image is removed from your system

```
aamir@ap-linux:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	055936d39205	15 hours ago	5.53MB

# DOCKER CONTAINERS



Image source:

[https://www.monitis.com%2Fblog%2Ftop-5-benefits-of-containerization%2F&psig=AOvVaw3l6VvAu\\_TMOaxYS4DZBNbn&ust=1556006137919830](https://www.monitis.com%2Fblog%2Ftop-5-benefits-of-containerization%2F&psig=AOvVaw3l6VvAu_TMOaxYS4DZBNbn&ust=1556006137919830)

# DOCKER CONTAINERS



- A container is the runtime instance of an image
- Technically from image you create a copy of your app that is now up and running
- In the same way that we can start a virtual machine (VM) from a virtual machine template, we start one or more containers from a single image
- The big difference between a VM and a container is that containers are faster and more lightweight
- Instead of running a full-blown OS like a VM, containers share the OS/kernel with the host they're running on
- You can even containerize big monolithic application but does not benefit you as it does doing microservices based application
- To run a container from an image in an interactive mode

```
aamir@ap-linux:~$ docker run -it aamirpinger/helloworld sh
#
```

# DOCKER CONTAINERS



- To exit from container shell type **exit**

```
# exit  
aamir@ap-linux:~$
```

- Check container list

```
aamir@ap-linux:~$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

- Run same container once again

```
aamir@ap-linux:~$ docker run -it aamirpinger/helloworld sh  
#
```

- Now press **ctrl PQ**



# DOCKER CONTAINERS



- Check container list again, with the ps command which is alternate to docker container ls command

```
aamir@ap-linux:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
987f94958d57	aamirpinger/helloworld	"sh"	13 seconds ago	Up 10 seconds	80/tcp	festive_noether

- To go back to running container shell

```
aamir@ap-linux:~$ docker exec -it festive_noether sh
#
```

- Now exiting this container won't kill container, to stop container we have to do it manually

```
aamir@ap-linux:~$ docker container stop festive_noether
festive_noether
```

# DOCKER CONTAINERS



- Check container list again

```
aamir@ap-linux:~$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

- Check container list again with **-a**, we can even see **docker ps -a**

```
aamir@ap-linux:~$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
987f94958d57	aamirpinger/helloworld	"sh"	13 seconds ago	Exited (137) 9 seconds ago	80/tcp	festive_noether

- To restart stopped container

```
aamir@ap-linux:~$ docker container start festive_noether  
festive_noether
```

- Stopping a container does not destroy your data inside the file system of the container

# DOCKER CONTAINERS



- To remove container permanently

```
aamir@ap-linux:~$ docker container rm festive_noether  
festive_noether
```

- Removing container will now permanently delete files inside container filesystem
- You can even give container a proper name when create by adding **--name <name>** option in **docker run** command
- You can also use **-d** instead of **-it** to run you container in background and does not attach shell after creation but of course you can access shell using **docker exec** command

```
aamir@ap-linux:~$ docker run -d --name mycontainer aamirpinger/helloworld  
f47c169271691820f08dba445e90550b6a2c4878b179bfff2b27e09736d54eaf
```

# DOCKER CONTAINERS



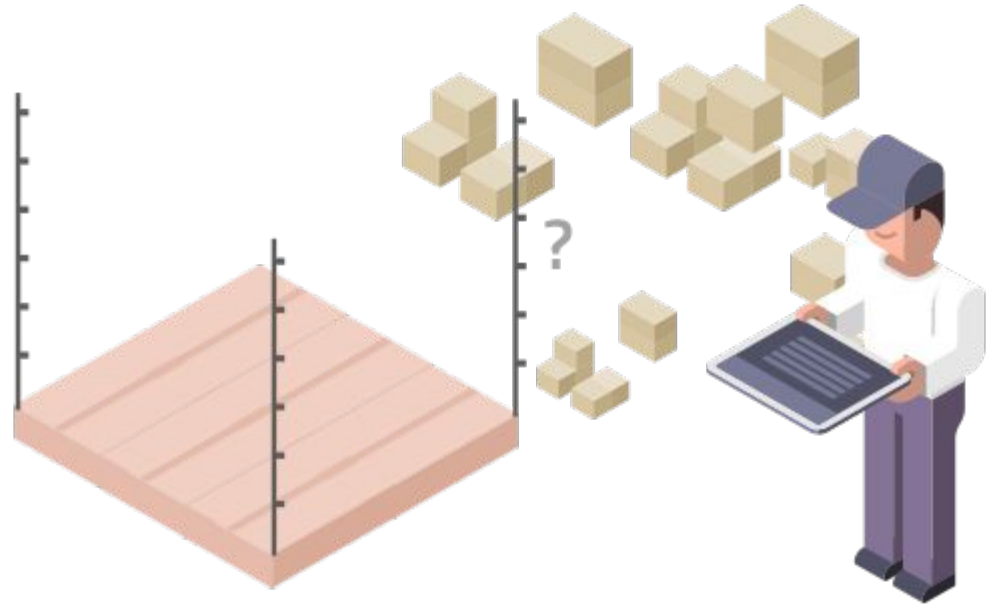
- To access the running container app through browser we need to expose port while creating container

```
aamir@ap-linux:~$ docker container rm mycontainer  
mycontainer  
aamir@ap-linux:~$ docker run -d --name mycontainer -p 8100:80 aamirpinger/helloworld  
b44df7132eba29a8bc1cc026af9601a853d37252137891ce52c639f8a34a3634
```

- Now you can go to browser and type **localhost:8100**



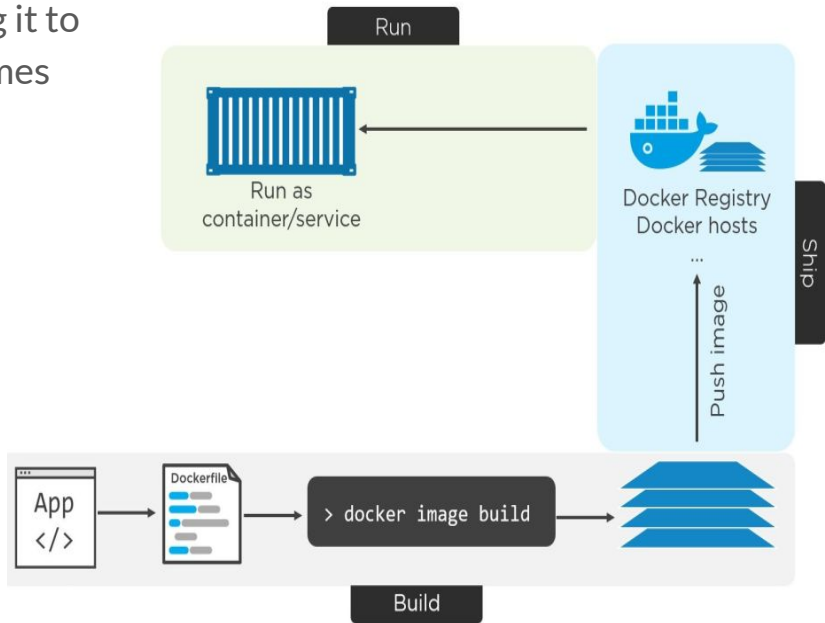
# CONTAINERIZING AN APP FROM SCRATCH



# CONTAINERIZING AN APP FROM SCRATCH

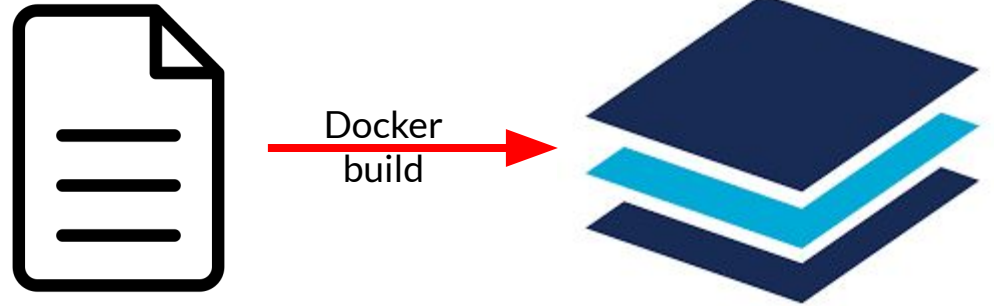


- The process of taking an application and configuring it to run as a container is called “containerizing”. Sometimes we call it “Dockerizing”
- The process of containerizing an app looks like this:
  - Start with your application code
  - Create a Dockerfile that describes your app, its dependencies, and how to run it
  - Feed this Dockerfile into the docker image build command to create an image
  - Create and run a container from that image





# Dockerfile





# Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile
- A Dockerfile is a simple text file with instructions on how to build your images
- The Dockerfile has two main purposes:
  - To describe the application
  - To tell Docker how to containerize the application (create an image with the app inside)
- Docker file cannot be reverse engineered from any image
- This is the simplest Dockerfile to create simple HTML and JS based website

```
FROM nginx
```

```
COPY . /usr/share/nginx/html
```



# Dockerfile



- Another Dockerfile example with more instruction for creating an image
- This is a typical Dockerfile required for any node.js application
- Let's go to next slide to do little practical on our systems

```
FROM alpine
LABEL maintainer="aamirpinger@yahoo.com"
RUN apk add --update nodejs nodejs-npm
COPY . /src
WORKDIR /src
RUN npm install
ENV CREATEDBY="Aamir Pinger"
EXPOSE 8080
ENTRYPOINT ["node", "./app.js"]
```

# CONTAINERIZING AN APP FROM SCRATCH



- Now let's try making a containerized app and run it on our systems
- Goto to the link <https://github.com/aamirpinger/docker-slide-code>
- Follow the instruction in README.md file



# PUSHING IMAGES

---

- One of the main advantage of image is portability, meaning you can use it from anywhere in the world
- To achieve above you need to do only one thing, save your image to registry like docker hub
- This saving image is also referred as pushing image to the docker hub
- You need to sign up for the account first at <https://hub.docker.com/>
- After creating an account you can simply push your image to docker hub repository
- One thing important to know is for pushing an image to docker hub, we need our images to be built as **username/repository:tag**
- Previously we created image with “**docker build -t first-docker-app .**” which does not include username

# PUSHING IMAGES



- We have two options either we create a new image from same docker file with

```
aamir@ap-linux:~$ docker build -t aamirpinger/first-docker-app .  
Sending build context to Docker daemon 9.728kB  
...  
Successfully tagged aamirpinger/first-docker-app:latest
```

OR

- Second option is docker tag command which help make a new image from existing image with different name tags

```
aamir@ap-linux:~$ docker tag first-docker-app aamirpinger/first-docker-app
```



# PUSHING IMAGES

- After proper tagging we need to do following

```
aamir@ap-linux:~$ docker push aamirpinger/first-docker-app
```

```
The push refers to repository [docker.io/aamirpinger/first-docker-app]
```

```
d29f287876ad: Pushed
```

```
86df2a1b653b: Mounted from aamirpinger/phw
```

```
bc5b41ec0cfa: Mounted from aamirpinger/phw
```

```
237472299760: Mounted from aamirpinger/phw
```

```
latest: digest: sha256:6c1329c151549cfc46ac2cfaf14808660d50468c149e47b4bdbed02faa41c55f size: 1156
```

- Congratulation! You have uploaded your image to docker hub registry and your image is now portable
- You can pull your image using docker pull from anywhere in the world and deploy with docker run command



# INSPECTING IMAGE



# INSPECTING IMAGE



- To inspect layers of your image
  - `docker history <image name>`

```
aamir@ap-linux:~$ docker history node-app-image
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
9f744124db2b	About an hour ago	/bin/sh -c #(nop) ENTRYPOINT ["/app...	0B	
199ebfe0cfe2	About an hour ago	/bin/sh -c #(nop) EXPOSE 8081	0B	
a74652abf0a4	About an hour ago	/bin/sh -c #(nop) ENV CREATEDBY=Aamir Pinger	0B	
68299b823abe	About an hour ago	/bin/sh -c npm install	226B	
2fd8b5083f35	About an hour ago	/bin/sh -c #(nop) WORKDIR /src	0B	
ae5de6a4389d	About an hour ago	/bin/sh -c #(nop) COPY dir:47833d35df9142891...	1.64MB	
ab370774ca00	About an hour ago	/bin/sh -c apk add --update nodejs nodejs-npm	42.6MB	
7eb226732929	About an hour ago	/bin/sh -c #(nop) LABEL maintainer=aamirpin...	0B	
055936d39205	3 days ago	/bin/sh -c #(nop) CMD ["/bin/sh"]	0B	
<missing>	3 days ago	/bin/sh -c #(nop) ADD file:a86aea1f3a7d68f6a...	5.53MB	

# INSPECTING IMAGE

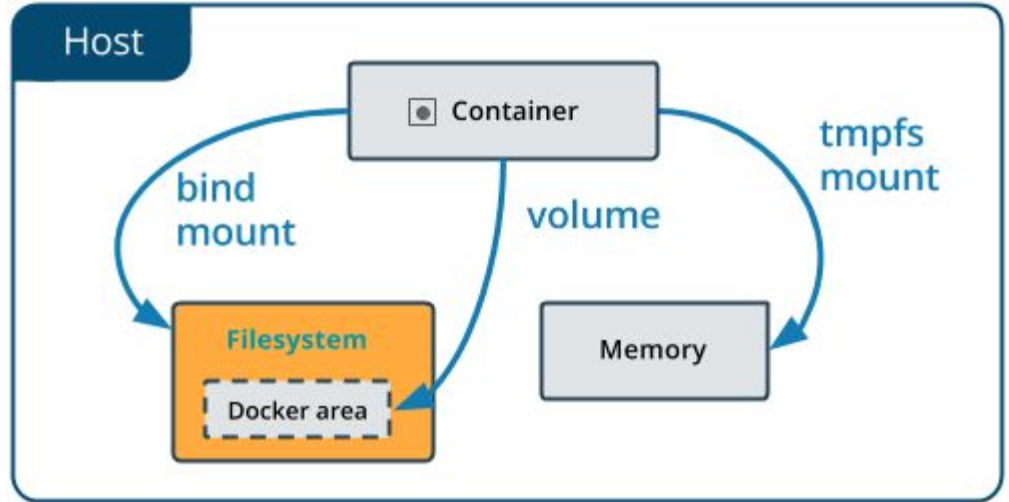


- To inspect layers of your image
  - **docker inspect <image name>**

```
aamir@ap-linux:~$ docker inspect node-app-image
[
  {
    "Id": "sha256:9f744124db2b49720188e754a0bf21e98af38b3113f2a0cb183d4a360658b5ee",
    "RepoTags": [
      "node-app-image:latest"
    ],
    "RepoDigests": [],
    "Parent": "sha256:199ebfe0cfe2589025b625e4c1681f2bd5f65fbd4f41bab199ce74118788687c",
    "Comment": "",
    "Created": "2019-05-14T20:21:47.546926085Z",
    "Container": "11a3c1271a4c29b9290bbf1c79a95279fff1c3676e67c48d7e69b83fc9ddaf4d",
    "ContainerConfig": {
      "Hostname": "11a3c1271a4c",
      "Domainname": "",
      "ExposedPorts": {
        "8081/tcp": {}
      }
    }
  }
]
```



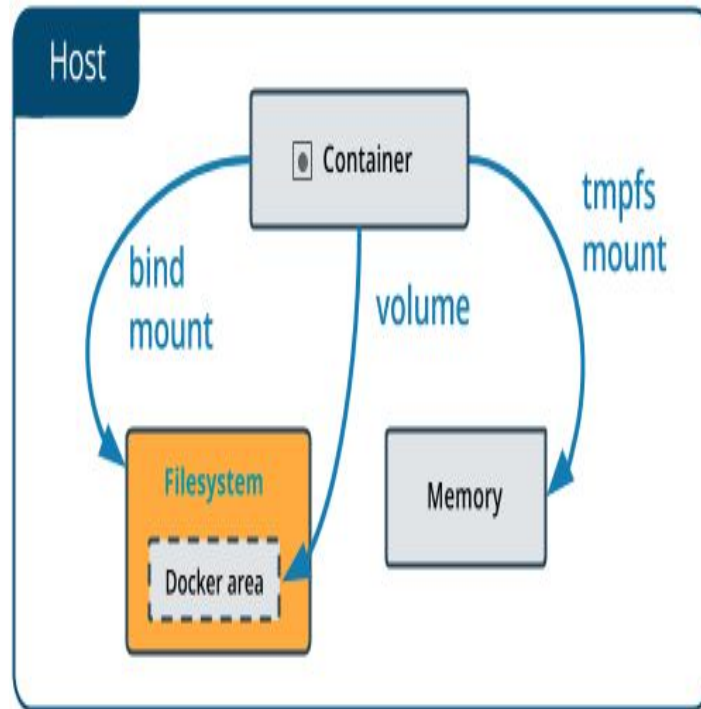
# BIND MOUNTS



# BIND MOUNT



- By default all files created inside a container are stored on a writable container layer.
- This means that the data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops:
  - Volumes,
  - Bind mounts
  - If you're running Docker on Linux you can also use a tmpfs mount





# BIND MOUNT

---

- By using bind mount, a file or directory on the host machine is mounted into a container
- The file or directory is referenced by its full path on the host machine
- The file or directory does not need to exist on the Docker host already. It is created on demand if it does not yet exist
- Bind mounts are very performant, but they rely on the host machine's filesystem having a specific directory structure available
- We will be using
  - `-v` or `--volume` tag
  - Target folder at host
  - `:` as a separator
  - Source folder of container



# BIND MOUNT

```
aamir@ap-linux:~$ docker run -d --name bindtest -v ~/target-on-host:/app nginx:latest  
B44df7132eba29a8bc1cc026af9601a853d37252137891ce52c639f8a34a3634
```

```
aamir@ap-linux:~$ docker exec -it bindtest sh  
# cd app  
# echo "this is example file for docker bind" > test.txt  
# ls  
test.txt  
# exit
```

```
aamir@ap-linux:~$ cd target-on-host  
aamir@ap-linux:~/target-on-host$ ls  
test.txt  
aamir@ap-linux:~/target-on-host$ cat test.txt  
"this is example file for docker bind"
```



# Thank you and God bless you all!



[fb.com/AamirPingerOfficial](https://fb.com/AamirPingerOfficial)



[linkedin.com/in/AamirPinger](https://linkedin.com/in/AamirPinger)



[github.com/AamirPinger](https://github.com/AamirPinger)