

# COMP 350

## Introduction to DevOps

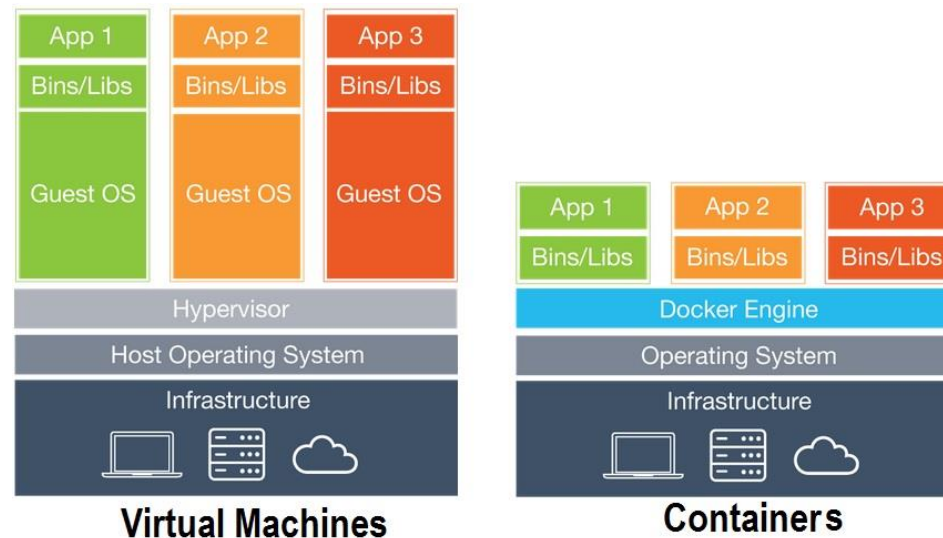
### Lecture 6

### Containers and Docker

Hakan Ayrar

# What is Docker?

- Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called **containers**.
- Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.
- Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines.



# What is Docker?

- Docker can package an application and its dependencies in a virtual container that can run on any OS/computer.
  - This enables the application to run in a variety of locations, such as on-premises, in a public cloud, and/or in a private cloud.
- When running on Linux, Docker uses the resource isolation features of the Linux kernel (*such as cgroups and kernel namespaces*) and a union-capable file system (*such as OverlayFS*) to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.
- Because Docker containers are lightweight, a single server or virtual machine can run several containers simultaneously.
  - A 2018 analysis found that a typical Docker use case involves running eight containers per host, and that a quarter of analyzed organizations run 18 or more per host.

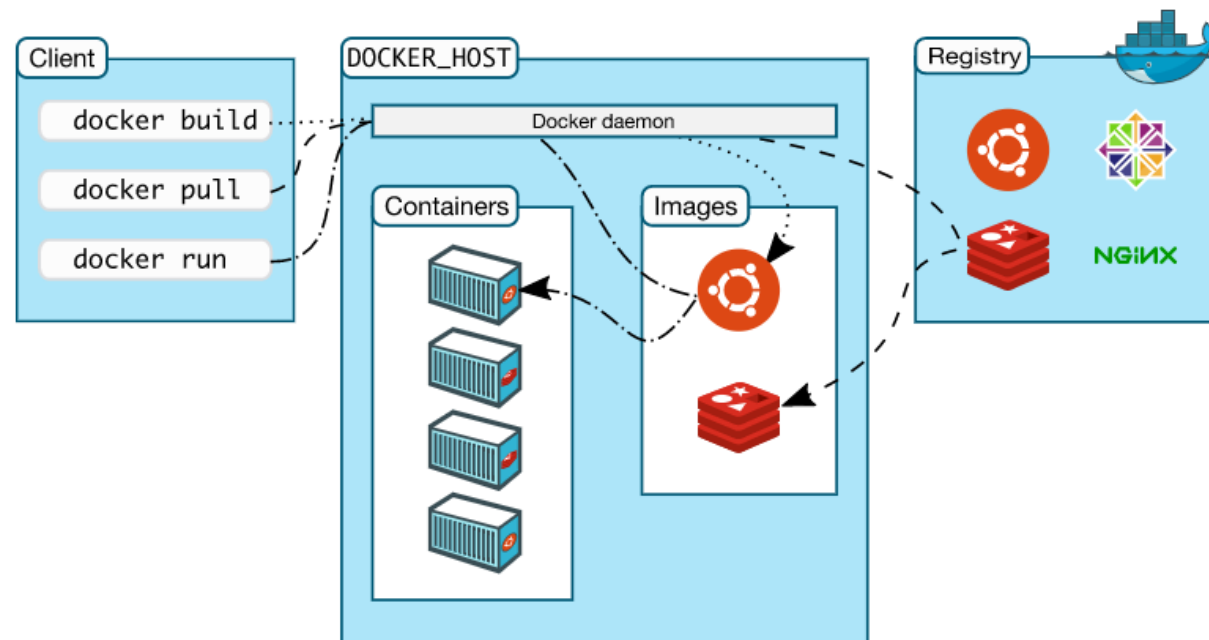
# Components

The Docker software as a service offering consists of three components:

- **Software:**
  - The **Docker daemon**, called **dockerd**, is a persistent process that manages Docker containers and handles container objects. The daemon listens for requests sent via the Docker Engine API.
  - The **Docker client** program, called **docker**, provides a command-line interface, CLI, that allows users to interact with Docker daemons.
- **Objects:** Docker objects are various entities used to assemble an application in Docker. The main classes of Docker objects are **images**, **containers**, and **services**.
  - A **Docker container** is a standardized, encapsulated environment that runs applications. A container is managed using the Docker API or CLI.
  - A **Docker image** is a read-only template used to build containers. Images are used to store and ship applications.
  - A **Docker service** allows containers to be scaled across multiple Docker daemons. The result is known as a swarm, a set of cooperating daemons that communicate through the Docker API.
- **Registries:** A Docker registry is a repository for Docker images. Docker clients connect to registries to download ("pull") images for use or upload ("push") images that they have built. Registries can be public or private. Docker Hub is the default registry where Docker looks for images.

# Docker Client, Daemon and Registry

- Docker **Client** and Docker **Daemon** can be on the same physical computer or they can be on different computers and communicate through network.
- **Registry** is almost always a server on the internet or LAN.
- You can have multiple instances of the same **image** running as different **containers** (e.g. having multiple *MySQL servers running*)



# Why the different package names?

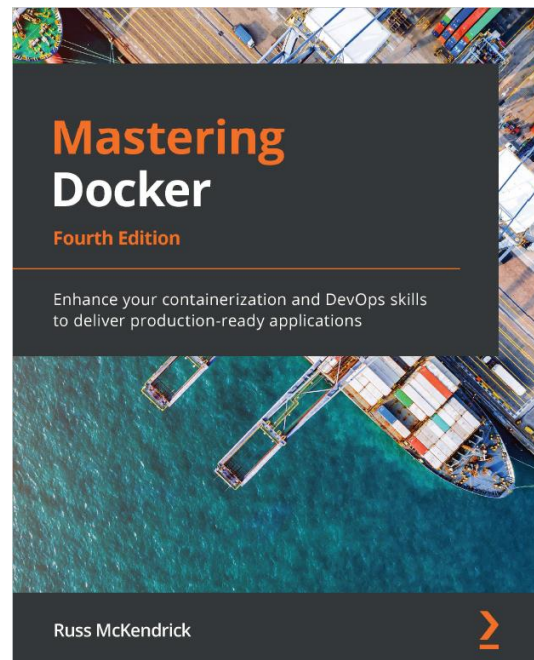
- Older versions of the Docker binary were called docker or docker-engine or docker-io
  - **docker-io** package is still the name used by Debian/Ubuntu for the docker release provided on their official repos.
  - **docker-ce** is a certified release provided directly by [docker.com](https://docker.com) and can also be built from source.
- Main reason for using the name **docker-io** on Debian/Ubuntu platform was to avoid a name conflict with docker system-tray binary which used to have the package name “docker” (*it is now renamed as “wmdocker”*).
- Docker has an enterprise version (EE) and a free community Edition version(CE), the “-ce” on “**docker-ce**” comes from “community edition”

# Why the different package names?

- docker-ce is provided by docker.com, docker.io is provided by Debian.
- This means you can install docker.io directly from your linux distro repository, while for docker-ce you have to first add the official repository from docker.com.
- More importantly, however, although both packages provide properly released versions of Docker, they have a very different internal structure:
  - **docker.io** does it the Debian (or Ubuntu) way: Each external dependency is a separate package that can and will be updated independently.
  - **docker-ce** does it the Golang way: All dependencies are pulled into the source tree before the build and the whole thing forms one single package afterwards. So you always update docker with all its dependencies at once.

# Reference Material

- During the lectures on Docker the information on the slides will come from various sources but if you feel the need to use a reference material you can refer to Mastering Docker 4<sup>th</sup> ed. by Russ McKendrick.





# Uninstalling previous versions

- If you are going to install Docker Community Edition (docker-ce from docker.com) you may need to remove older binaries especially if you have previously installed docker from your distro repo instead of docker.com repo.

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

- In our case this is not necessary because we haven't installed any version of Docker to our VM yet.
- The docker-ce binaries will tend to be the latest versions, while the version provided by your distro repo may be one version behind.

# Installing Docker from debian/ubuntu package repository

- You can install docker from your Linux distribution's repository with `apt-get install docker.io` if you are using a distribution which is based on Debian.
  - Note:
    - This download is over 300MB.
    - You might need to do `apt-get update` first depending on when you last updated your local metadata cache!

```
hayral@Computer1:~$ sudo apt-get install docker.io
[sudo] password for hayral:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils cgroupfs-mount containerd git git-man liberror-perl pigz runc
  ubuntu-fan
Suggested packages:
  aufs-tools debootstrap docker-doc rinse zfs-fuse | zfsutils git-daemon-run
  | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cvs
  git-mediawiki git-svn
The following NEW packages will be installed:
  bridge-utils cgroupfs-mount containerd docker.io git git-man liberror-perl
  pigz runc ubuntu-fan
0 upgraded, 10 newly installed, 0 to remove and 268 not upgraded.
Need to get 74,8 MB of archives.
After this operation, 372 MB of additional disk space will be used.
Do you want to continue? [Y/n] █
```

# Installing Docker from Docker repository

- First, update your existing list of packages:

```
sudo apt update
```

- Next, install a few prerequisite packages which let apt use packages over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

- Then add the GPG key for the official Docker repository to your system:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Add the Docker repository to APT sources:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

- This will also update our package database with the Docker packages from the newly added repo.

# Installing Docker from Docker repository

- Finally, install Docker:

```
sudo apt install docker-ce
```

- Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
sudo systemctl status docker
```

- The output should be similar to the following, showing that the service is active and running:

```
Output
• docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-05-19 17:00:41 UTC; 17s ago
 TriggeredBy: • docker.socket
     Docs: https://docs.docker.com
    Main PID: 24321 (dockerd)
       Tasks: 8
      Memory: 46.4M
     CGroup: /system.slice/docker.service
             └─24321 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

# Starting the Service

- If you installed Docker from debian/ubuntu package repository, after installing docker you should start the docker service with “systemctl start” and set it to auto-start at each boot by “systemctl enable”.
- or you can start the docker service manually with “systemctl start” everytime you are going to use it

```
hayral@Computer1:~$ sudo systemctl start docker
hayral@Computer1:~$ sudo systemctl enable docker
hayral@Computer1:~$ █
```

- Let's see if Docker has installed correctly by checking it's version:

```
hayral@Computer1:~$ docker --version
Docker version 20.10.2, build 20.10.2-0ubuntu1~20.04.2
hayral@Computer1:~$ █
```

# Hello World!

Let's try running the "hello-world" image:

- Currently we don't have any images on our local cache, if an image is not available on the cache, docker automatically downloads it from docker-hub on internet.
- What the hello-world image does is just showing the message shown on the figure; nevertheless, it contains a full linux environment which uses your kernel but has a user space of its own.

```
hayral@Computer1:~$  
hayral@Computer1:~$ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
b8dfde127a29: Pull complete  
Digest: sha256:f2266cbfcl27c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519  
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

```
To generate this message, Docker took the following steps:
```

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

```
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash
```

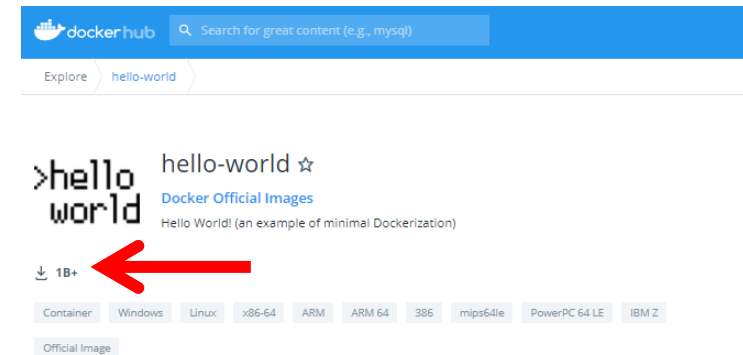
```
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/
```
















```
For more examples and ideas, visit:  
https://docs.docker.com/get-started/
```

```
hayral@Computer1:~$ █
```

# Hello World!

- This official hello-world image has been downloaded more than one billion times!
- List below shows most used docker images as of May 2023. Check this list from DockerHub because it also shows which skills are currently valuable on job market!



 <b>alpine</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 9.9K Updated a month ago A minimal Docker image based on Alpine Linux with a complete package index and only 5 ... Linux IBM Z riscv64 x86-64 ARM ARM 64 386 PowerPC 64 LE <a href="#">Learn more</a>	 <b>redis</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 5 days ago Redis is an open source key-value store that functions as a data structure server. Windows Linux 386 mips64le PowerPC 64 LE IBM Z x86-64 ARM ARM 64 <a href="#">Learn more</a>	 <b>memcached</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 2.0K Updated 4 days ago Free & open source, high-performance, distributed memory object caching system. Linux PowerPC 64 LE IBM Z ARM 64 386 x86-64 ARM mips64le <a href="#">Learn more</a>
 <b>nginx</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 4 days ago Official build of Nginx. Linux 386 mips64le PowerPC 64 LE IBM Z ARM x86-64 ARM 64 <a href="#">Learn more</a>	 <b>postgres</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 5 days ago The PostgreSQL object-relational database system provides reliability and data integrity. Linux PowerPC 64 LE IBM Z x86-64 ARM ARM 64 386 mips64le <a href="#">Learn more</a>	 <b>mysql</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 5 days ago MySQL is a widely used, open-source relational database management system (RDBMS). Linux ARM 64 x86-64 <a href="#">Learn more</a>
 <b>busybox</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 3.0K Updated 2 months ago Busybox base image. Linux 386 mips64le PowerPC 64 LE riscv64 IBM Z x86-64 ARM ARM 64 <a href="#">Learn more</a>	 <b>node</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 3 days ago Node.js is a JavaScript-based platform for server-side and networking applications. Linux IBM Z 386 x86-64 ARM ARM 64 PowerPC 64 LE <a href="#">Learn more</a>	 <b>traefik</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 2.9K Updated 11 days ago Traefik, The Cloud Native Edge Router Windows Linux x86-64 IBM Z ARM ARM 64 <a href="#">Learn more</a>
 <b>ubuntu</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 10K+ Updated 5 days ago Ubuntu is a Debian-based Linux operating system based on free software. Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE <a href="#">Learn more</a>	 <b>httpd</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 4.4K Updated 4 days ago The Apache HTTP Server Project Linux ARM 386 mips64le PowerPC 64 LE ARM 64 IBM Z x86-64 <a href="#">Learn more</a>	 <b>mariadb</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 5.4K Updated 4 days ago MariaDB Server is a high performing open source relational database, forked from MySQL. Linux IBM Z 386 x86-64 ARM 64 PowerPC 64 LE <a href="#">Learn more</a>
 <b>python</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 8.7K Updated 4 days ago Python is an interpreted, interactive, object-oriented, open-source programming language. Linux Windows 386 PowerPC 64 LE IBM Z mips64le x86-64 ARM ARM 64 <a href="#">Learn more</a>	 <b>mongo</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 9.6K Updated 4 days ago MongoDB document databases provide high availability and easy scalability. Linux Windows ARM 64 IBM Z x86-64 <a href="#">Learn more</a>	 <b>rabbitmq</b> <span>DOCKER OFFICIAL IMAGE</span> · 1B+ · 4.8K Updated 5 days ago RabbitMQ is an open source multi-protocol messaging broker. Linux PowerPC 64 LE IBM Z 386 riscv64 x86-64 ARM ARM 64 <a href="#">Learn more</a>

# Detailed Version Information

- You can check the version of different components of Docker with `docker version`
- You can see that it consists of a client and a server, furthermore the server consists of components like
  - Engine
  - containerd
  - runc
  - docker-init

```

hayral@Computer1:~$
hayral@Computer1:~$ sudo docker version
Client:
 Version:           20.10.2
 API version:       1.41
 Go version:        gol.13.8
 Git commit:        20.10.2-0ubuntu1~20.04.2
 Built:             Tue Mar 30 21:24:57 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server:
 Engine:
  Version:           20.10.2
  API version:       1.41 (minimum version 1.12)
  Go version:        gol.13.8
  Git commit:        20.10.2-0ubuntu1~20.04.2
  Built:             Mon Mar 29 19:10:09 2021
  OS/Arch:           linux/amd64
  Experimental:      false
 containerd:
  Version:           1.3.3-0ubuntu2.3
  GitCommit:
 runc:
  Version:           spec: 1.0.2-dev
  GitCommit:
 docker-init:
  Version:           0.19.0
  GitCommit:
hayral@Computer1:~$

```



# More info

- You can get more detailed info about the drivers, installed plugins and versions of the components by issuing the `docker info` command.

```
hayral@Computer1:~$ sudo docker info
```

```
Client:
```

```
Context: default
```

```
Debug Mode: false
```

```
Server:
```

```
Containers: 1
```

```
Running: 0
```

```
Paused: 0
```

```
Stopped: 1
```

```
Images: 1
```

```
Server Version: 20.10.2
```

```
Storage Driver: overlay2
```

```
Backing Filesystem: extfs
```

```
Supports d_type: true
```

```
Native Overlay Diff: true
```

```
Logging Driver: json-file
```

```
Cgroup Driver: cgroupfs
```

```
Cgroup Version: 1
```

```
Plugins:
```

```
Volume: local
```

```
Network: bridge host ipvlan macvlan null overlay
```

```
Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
```

```
Swarm: inactive
```

```
Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
```

```
Default Runtime: runc
```

```
Init Binary: docker-init
```

```
containerd version:
```

```
runc version:
```

```
init version:
```

```
Security Options:
```

```
apparmor
```

```
seccomp
```

```
Profile: default
```

```
Kernel Version: 5.4.0-58-generic
```

```
Operating System: Linux Mint 20.1
```

```
OSType: linux
```

```
Architecture: x86_64
```

```
CPUs: 2
```

```
Total Memory: 855.1MiB
```

```
Name: Computer1
```

```
ID: IRUL:MOBE:MGSP:I3AH:TTBZ:N07U:5U7Q:I60W:2JCT:UKJE:HFE2:QVSI
```

```
Docker Root Dir: /var/lib/docker
```

```
Debug Mode: false
```

```
Registry: https://index.docker.io/v1/
```

```
Labels:
```

```
Experimental: false
```

```
Insecure Registries:
```

```
127.0.0.0/8
```

```
Live Restore Enabled: false
```

# Docker Commands

- Docker is administered with many commands which you specify as a command option of docker command (the use of the term command in two contexts is a little confusing you pass the name of a “docker command” as a command option to “docker” which is the command to manage docker.)

`docker <options> command`

- There are two kinds of docker commands
  - management commands and
  - regular commands (listed on the next slide)

```
hayral@Computer1:~$ docker help
```

```
Usage:  docker [OPTIONS] COMMAND
```

```
A self-sufficient runtime for containers
```

```
Options:
```

<code>--config</code> string	Location of client config files (default "/home/hayral/.docke
<code>-c, --context</code> string	Name of the context to use to connect to the daemon (override and default context set with "docker context use")
<code>-D, --debug</code>	Enable debug mode
<code>-H, --host</code> list	Daemon socket(s) to connect to
<code>-l, --log-level</code> string	Set the logging level ("debug" "info" "warn" "error" "fatal")
<code>--tls</code>	Use TLS; implied by --tlsverify
<code>--tlscacert</code> string	Trust certs signed only by this CA (default "/home/hayral/.do
<code>--tlscert</code> string	Path to TLS certificate file (default "/home/hayral/.docker/c
<code>--tlskey</code> string	Path to TLS key file (default "/home/hayral/.docker/key.pem")
<code>--tlsverify</code>	Use TLS and verify the remote
<code>-v, --version</code>	Print version information and quit

```
Management Commands:
```

<code>builder</code>	Manage builds
<code>config</code>	Manage Docker configs
<code>container</code>	Manage containers
<code>context</code>	Manage contexts
<code>image</code>	Manage images
<code>manifest</code>	Manage Docker image manifests and manifest lists
<code>network</code>	Manage networks
<code>node</code>	Manage Swarm nodes
<code>plugin</code>	Manage plugins
<code>secret</code>	Manage Docker secrets
<code>service</code>	Manage services
<code>stack</code>	Manage Docker stacks
<code>swarm</code>	Manage Swarm
<code>system</code>	Manage Docker
<code>trust</code>	Manage trust on Docker images
<code>volume</code>	Manage volumes

# Docker Commands

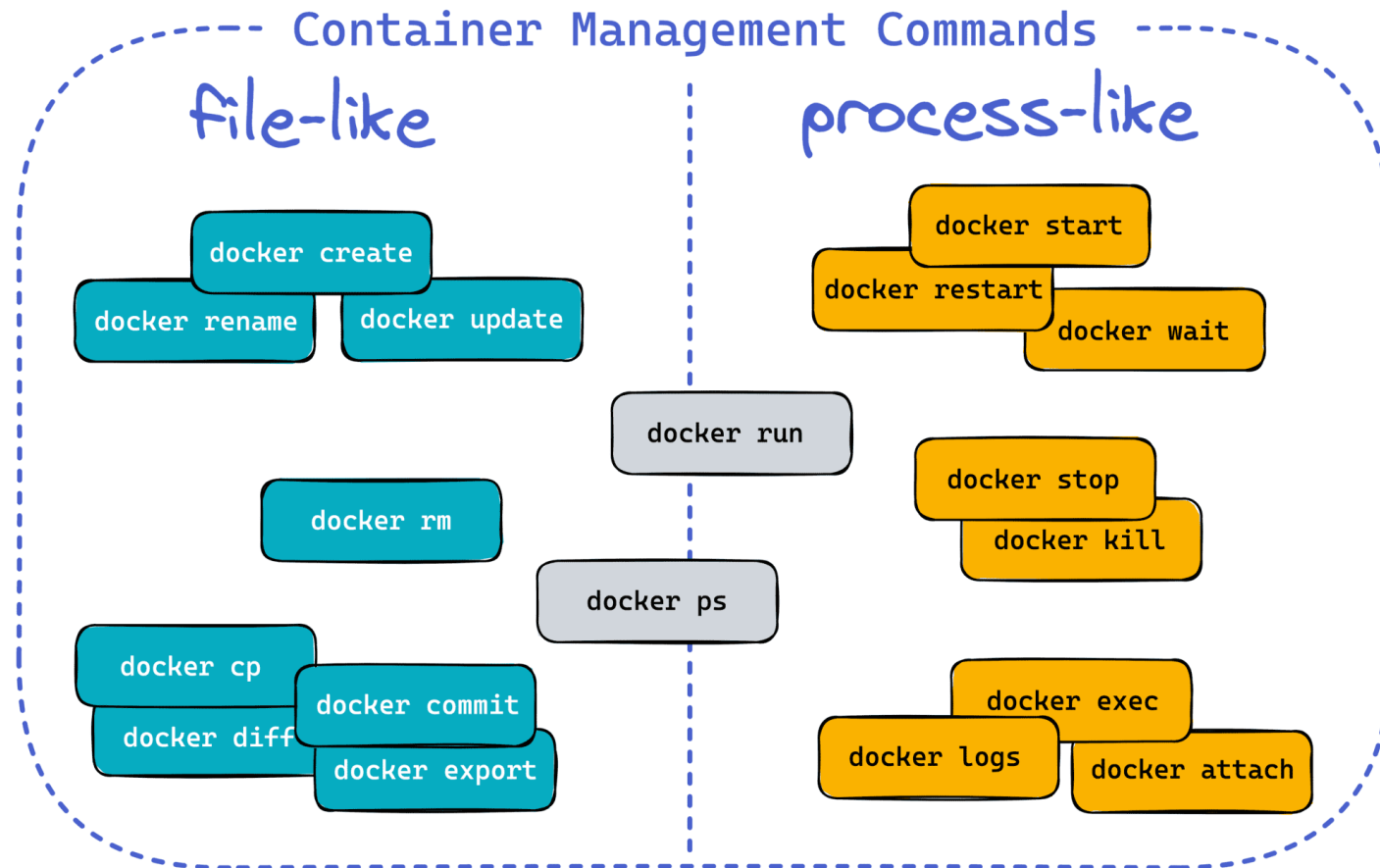
- Regular docker commands allow you to
  - create/list/delete the containers/images
  - start/stop/pause/resume/kill running containers
  - connect to the console of a running container
  - send shell commands to a running container
  - copy files from/to container's internal file system
  - etc..

Commands:	
attach	Attach local standard input, output, and error streams to a running container
build	Build an image from a Dockerfile
commit	Create a new image from a container's changes
cp	Copy files/folders between a container and the local filesystem
create	Create a new container
diff	Inspect changes to files or directories on a container's filesystem
events	Get real time events from the server
exec	Run a command in a running container
export	Export a container's filesystem as a tar archive
history	Show the history of an image
images	List images
import	Import the contents from a tarball to create a filesystem image
info	Display system-wide information
inspect	Return low-level information on Docker objects
kill	Kill one or more running containers
load	Load an image from a tar archive or STDIN
login	Log in to a Docker registry
logout	Log out from a Docker registry
logs	Fetch the logs of a container
pause	Pause all processes within one or more containers
port	List port mappings or a specific mapping for the container
ps	List containers
pull	Pull an image or a repository from a registry
push	Push an image or a repository to a registry
rename	Rename a container
restart	Restart one or more containers
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save one or more images to a tar archive (streamed to STDOUT by default)
search	Search the Docker Hub for images
start	Start one or more stopped containers
stats	Display a live stream of container(s) resource usage statistics
stop	Stop one or more running containers
tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top	Display the running processes of a container
unpause	Unpause all processes within one or more containers
update	Update configuration of one or more containers
version	Show the Docker version information
wait	Block until one or more containers stop, then print their exit codes

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

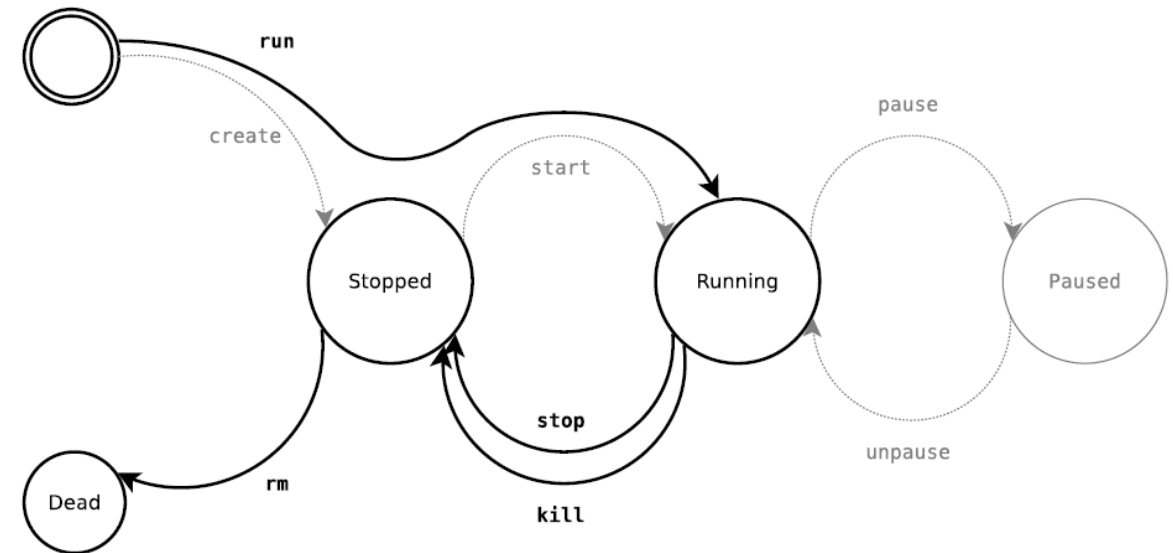
To get more help with docker, check out our guides at <https://docs.docker.com/go/guides/>  
hayral@Computer1:~\$ █

# Docker Commands



# Container Management Commands & States

command	description
<code>docker create image [ command ]</code>	create the container
<code>docker run image [ command ]</code>	= <b>create</b> + <b>start</b>
<code>docker start container...</code>	start the container
<code>docker stop container...</code>	graceful <sup>2</sup> stop
<code>docker kill container...</code>	kill (SIGKILL) the container
<code>docker restart container...</code>	= <b>stop</b> + <b>start</b>
<code>docker pause container...</code>	suspend the container
<code>docker unpause container...</code>	resume the container
<code>docker rm [ -f<sup>3</sup> ] container...</code>	destroy the container



<sup>2</sup>send SIGTERM to the main process + SIGKILL 10 seconds later

<sup>3</sup>-f allows removing running containers (= `docker kill` + `docker rm`)

# Inspecting & Interacting with the containers

command	description
<code>docker ps</code>	list running containers
<code>docker ps -a</code>	list all containers
<code>docker logs [ -f<sup>6</sup> ] container</code>	show the container output ( <i>stdout+stderr</i> )
<code>docker top container [ ps options ]</code>	list the processes running inside the containers
<code>docker diff container</code>	show the differences with the image (modified files)
<code>docker inspect container...</code>	show low-level infos (in json format)

command	description
<code>docker attach container</code>	attach to a running container (stdin/stdout/stderr)
<code>docker cp container:path hostpath -</code> <code>docker cp hostpath - container:path</code>	copy files from the container copy files into the container
<code>docker export container</code>	export the content of the container (tar archive)
<code>docker exec container args...</code>	run a command in an existing container ( <b>useful</b> for debugging)
<code>docker wait container</code>	wait until the container terminates and return the exit code
<code>docker commit container image</code>	commit a new docker image (snapshot of the container)

# Image Management & Transfer Commands

command	description
<code>docker images</code> <code>docker history image</code>  <code>docker inspect image...</code>	list all local images show the image history (list of ancestors) show low-level infos (in json format)
<code>docker tag image tag</code>	tag an image
<code>docker commit container image</code>  <code>docker import url - [tag]</code>	create an image (from a container) create an image (from a tarball)
<code>docker rmi image...</code>	delete images

## Using the registry API

<code>docker pull repo[:tag]...</code> <code>docker push repo[:tag]...</code> <code>docker search text</code>	pull an image/repo from a registry push an image/repo from a registry search an image on the official registry
<code>docker login ...</code> <code>docker logout ...</code>	login to a registry logout from a registry

## Manual transfer

<code>docker save repo[:tag]...</code> <code>docker load</code>	export an image/repo as a tarball load images from a tarball
<code>docker-ssh<sup>10</sup> ...</code>	proposed script to transfer images between two daemons over ssh

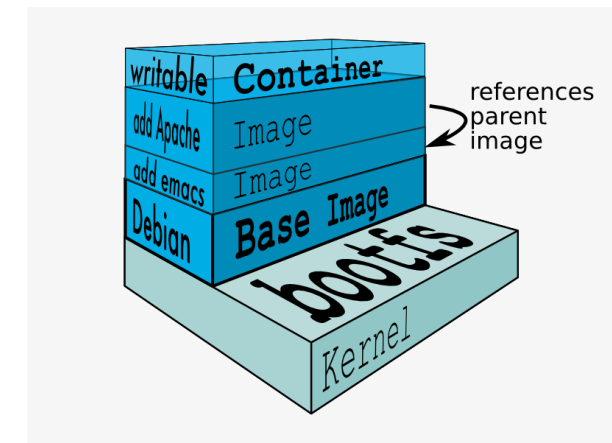


# What is a Docker Image/Container ?

A docker image is an immutable snapshot of the filesystem

A docker container is

- a temporary file system
  - layered over an immutable fs (docker image)
  - fully writable (copy-on-write<sup>1</sup>)
  - dropped at container's end of life (unless a `commit` is made)
- a network stack
  - with its own private address (*by default in 172.17.x.x*)
- a process group
  - one main process launched inside the container
  - all sub-processes SIGKILLED when the main process exits



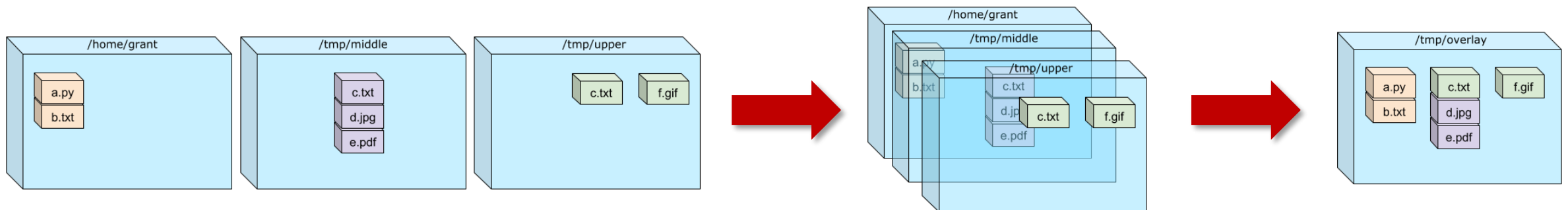
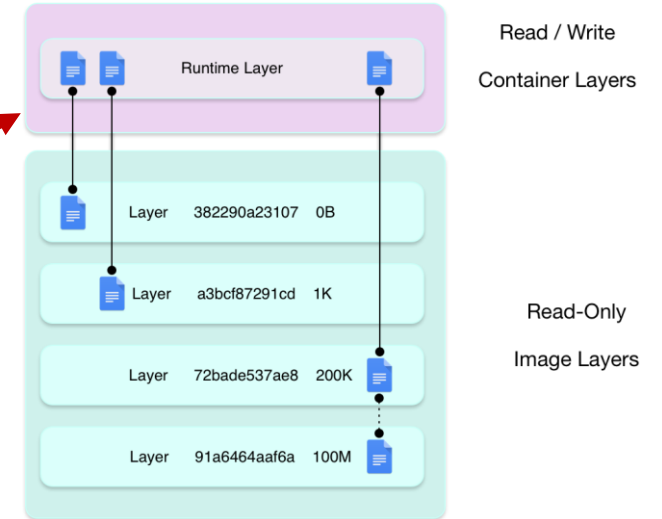

---

<sup>1</sup>several possible methods: overlaysfs (default), btrfs, lvm, zfs, aufs



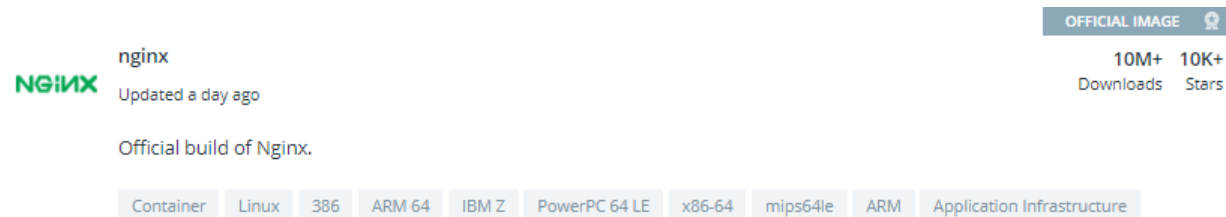
# How layered file systems work?

- Image: Layers of files stacked on top of each other. You view the final state of the file system when looked from the top, where changes from higher layers override the lower layers
- Container: When a container is created from an image, **a temporary layer** is added on top which accumulates all the changes made during runtime, but because it is temporary all changes will be lost when container is removed.
  - You can map an external path to a path inside container and use that location to store persistent data.



# Another image, another container

- Nginx is a web server, load balancer and reverse proxy.
- It is the 2<sup>nd</sup> most downloaded image on Docker Hub (as of May 2023):

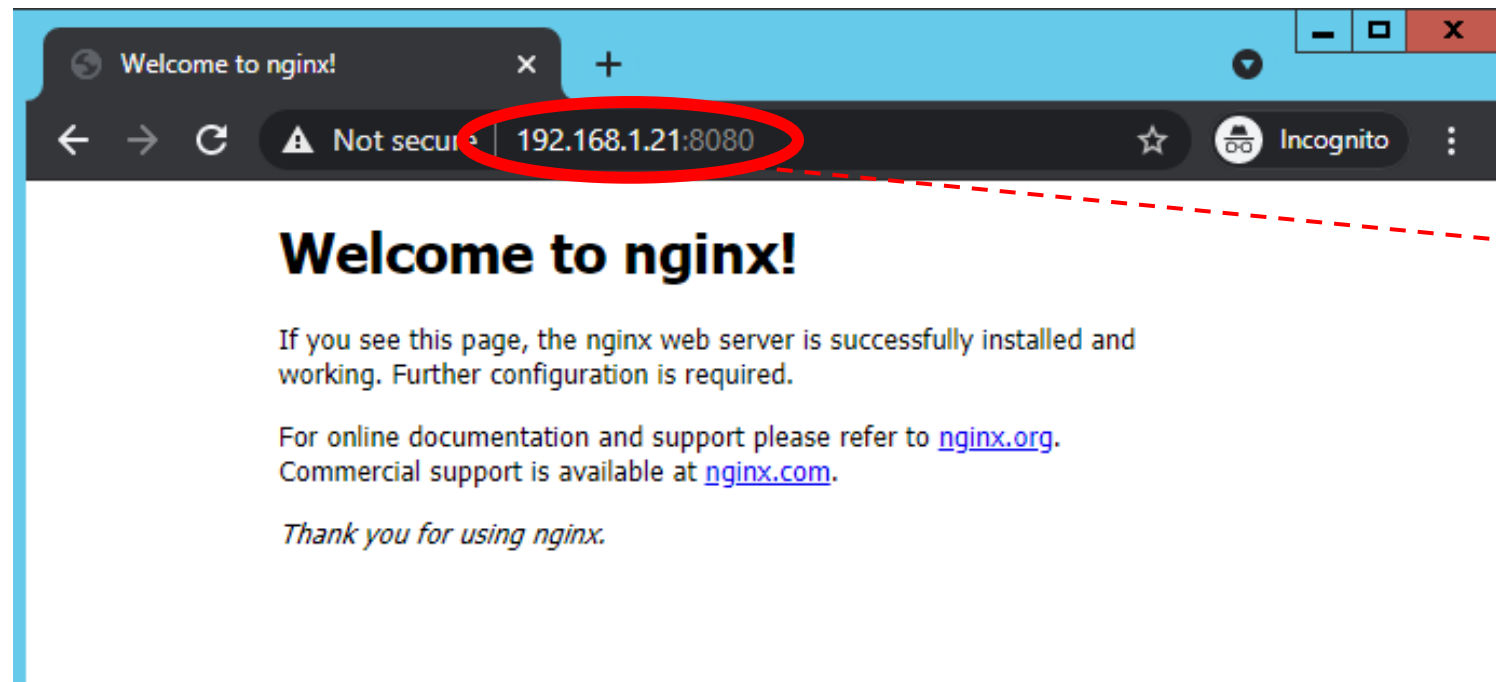


- You can run a container with this image using the command below (*we will look at command options in detail on later slides*):

```
hayral@Computer1:~$ sudo docker container run -d --name nginx-test -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f7ec5a41d630: Pull complete
aalefa14b3bf: Pull complete
b78b95af9b17: Pull complete
c7d6bca2b8dc: Pull complete
cf16cd8e71e0: Pull complete
0241c68333ef: Pull complete
Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
Status: Downloaded newer image for nginx:latest
4d0c949313178273c381ae35623d9c80dfc054bd95035a2c630b5939f0d93b2d
hayral@Computer1:~$
```

# Services exposed by container

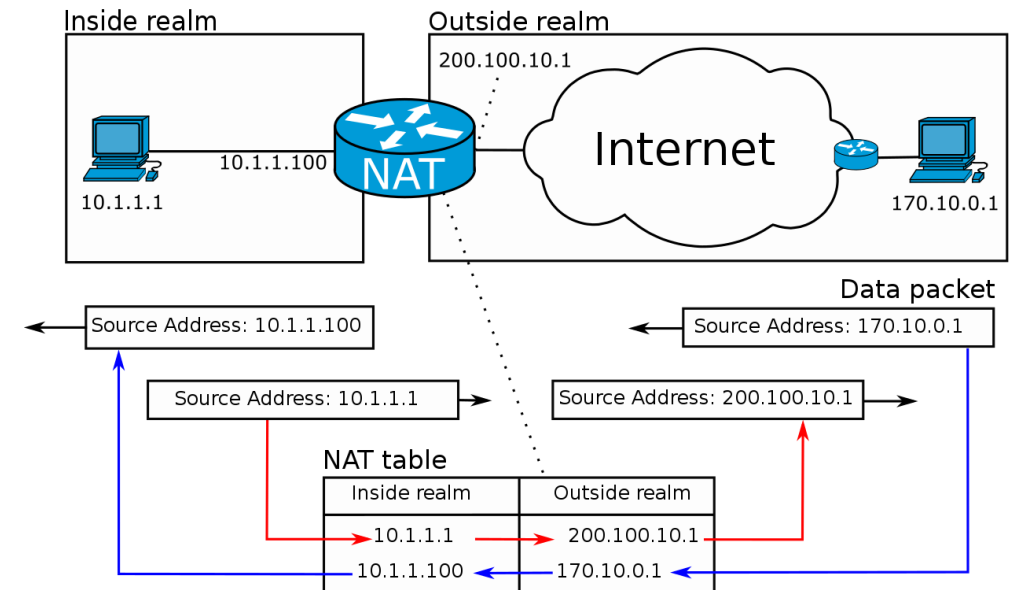
- Congratulations! You have just installed a new web server.
- You can connect to the Nginx server running inside the container by using a browser on Host OS and connect to the port 8080 of the Guest OS.
  - the end point <containerIP:80> is mapped to <guestOSIP:8080> with the command parameter **-p 8080:80** on docker command
  - Why we need mapping? → Docker and browser are on different networks.



write your  
own IP here

# Network Boundary

- When two networks have a **boundary** (*a device or devices which connects to both*) the two networks can communicate with each other by using the device on the boundary as a **router**.
- NAT** (network address translation) allows multiple machines on an internal network to communicate with a different outer i.e. LAN (**private IP**)  $\leftrightarrow$  Internet (**public IP/real**)



# Your current setup

- For the following slides we are going to assume the following three network setup which is currently used by the majority of you:

- A Windows computer (*Host OS*) running on the physical computer.

Network: LAN, network range: 192.168.1.X

- A Linux Virtual Machine (*Guest OS*) running on top of Host OS.

Network: depends on adapter type **bridged adapter** (→ part of LAN) or **NAT** (→ VirtualBox virtual network)

- Docker running on top of Guest OS.

Network: Docker virtual network, network range: 172.17.X.X

# What is Network Address Translation (NAT)?

- **NAT** allows multiple machines on an internal network to communicate with a different outer network by translating the addresses from one to the next at the **boundary** to provide transparent communication.
  - It does this by keeping track of incoming and outgoing traffic in a table of the form <src-ip:src-port>,<target-ip:target-port> to translate and map internal address/port pairs to external address/port pairs.
- There are two **boundaries** on the previous slide:
  - Docker NAT is at the boundary of Docker Network (its internal network) and the network outside (LAN if you run it directly, in our case VirtualBox's network because we run it on VM)
    - Docker provides NAT between your **Guest OS** and **Containers**.
  - VirtualBox is at the boundary of VirtualBox virtual network (**Guest OS** network, i.e. network as seen by the VM) and **Host OS** network (i.e. your LAN)
    - VirtualBox provides NAT between your Host OS and Guest OS.
    - *if you set connection settings to **bridged connection** and not NAT then VirtualBox will use the same network as Host (it'll be part of your LAN) and it will be visible to other computers on your LAN.*
- Actually there is a 3rd boundary: Your ADSL modem is at the **boundary** of your **LAN** and **Internet**, and it provides NAT between them.
- Different networks use different types of IP addresses:

Internet use **public IP** addresses.  $\longleftrightarrow$  Local networks use **private IP** addresses.

# Three networks, three IP addresses

- Host OS IP address
  - This is the IP address of your main OS (the outer most OS, most probably a Windows)
    - The IP address for this is most probably assigned by **DHCP** automatically (i.e. by your ADSL modem if you are at home) unless you specifically assigned a number manually.
- Guest OS (VM) IP address
  - This one depends on your VM settings;
    - if you set it to **NAT** the internal DHCP server of VirtualBox will assign an IP address automatically.
    - if you set it to **bridged connection** (and you are not connected with wireless 😞) it will try to get an IP from a **DHCP** server if one is available on your LAN (*almost all ADSL modems also act as a **DHCP** server*), otherwise it will require you to assign an IP address manually.
- Container (running inside VM) IP address
  - This is assigned automatically by Docker; but you can access with a host name which acts similar to DNS.

# What is DHCP?

- The Dynamic Host Configuration Protocol (DHCP) is a network management protocol used on IP networks for automatically assigning IP addresses and other communication parameters to devices connected to the network using a client–server architecture.
- DHCP eliminates the need for individually configuring network devices manually.





# Public vs Private IP addresses

- A **public IP** address is a globally routable unicast IP address, meaning that the address is not an address reserved for use in private networks.
  - Public IP addresses are used for communication between hosts on the global Internet.
- Not all computers connected to the Internet need a **public IP** address.
  - **Private networks** are widely used and typically connect to the Internet with network address translation (NAT), when needed.
  - In a home network/LAN, computers use **private IP** addresses for local communication, and they share a **public IP** address assigned by the ISP to their .
- Most public IP addresses on domestic use (*i.e. not used on a server or business setting*) change relatively often.
  - Any type of IP address that changes is called a **dynamic IP** address.
  - In home networks, the ISP usually assigns a **dynamic IP**.
- There are three non-overlapping ranges of IPv4 addresses reserved for **private networks**.
  - These addresses are not routed on the Internet and thus their use need not be coordinated with an IP address registry.
  - Any user may use any of the reserved blocks.
  - A network administrator can further divide a block into subnets;
    - for example, many home routers automatically use the default address range of 192.168.1.0/24.

Private address range		
Class	start address	finish address
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.31.255.255
C	192.168.0.0	192.168.255.255

Public address range		
Class	start address	finish address
A	0.0.0.0	126.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	254.255.255.255

Public IP vs Private IP	
Public IP	Private IP
Used over the Public Network ex. WAN	Used with in the private network ex. LAN
Recognized over the Internet	Not recognized over the Internet
Public IP are unique over the Globe.	Private IP are unique with in the network or LAN.
Public IP are paid.	Private IP are free of cost.
Assigned by Network Administrator.	Assigned by Internet Service Provider / IANA

# IP Addresses

- At this point we are dealing with 3 different networks and 3 different IP addresses on our machine:
  - LAN / Host OS IP address
  - Network of Virtual Machine / Guest OS (VM) IP address
  - Network of Container (running inside VM) / Container IP address

```
hakan@hakan-main-host:~$ ifconfig
docker0: flags=4099<BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:1f:0e:e8:5b txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.5 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::228:614e:a925:d602 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:e3:4a:2f txqueuelen 1000 (Ethernet)
    RX packets 47 bytes 8430 (8.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 100 bytes 11297 (11.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 171 bytes 18643 (18.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 171 bytes 18643 (18.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Windows (172.21.171.193 ← Koç Uni. wifi IP)

VirtualBox

Linux (10.0.2.5 ← Virtualbox NAT IP)

Docker

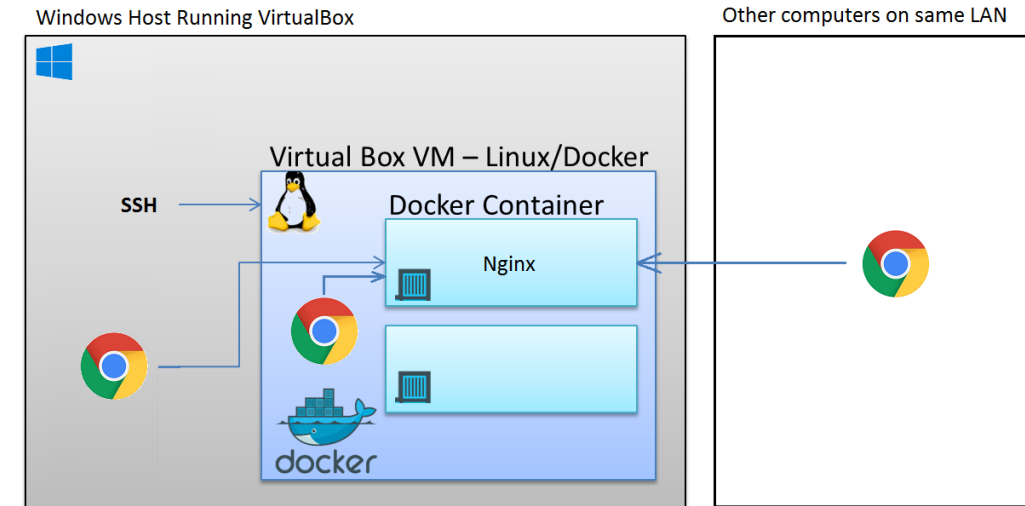
Container1  
(172.17.0.1)

Container2  
(172.17.0.X)

Container3  
(172.17.0.Y)

# Different locations to connect from

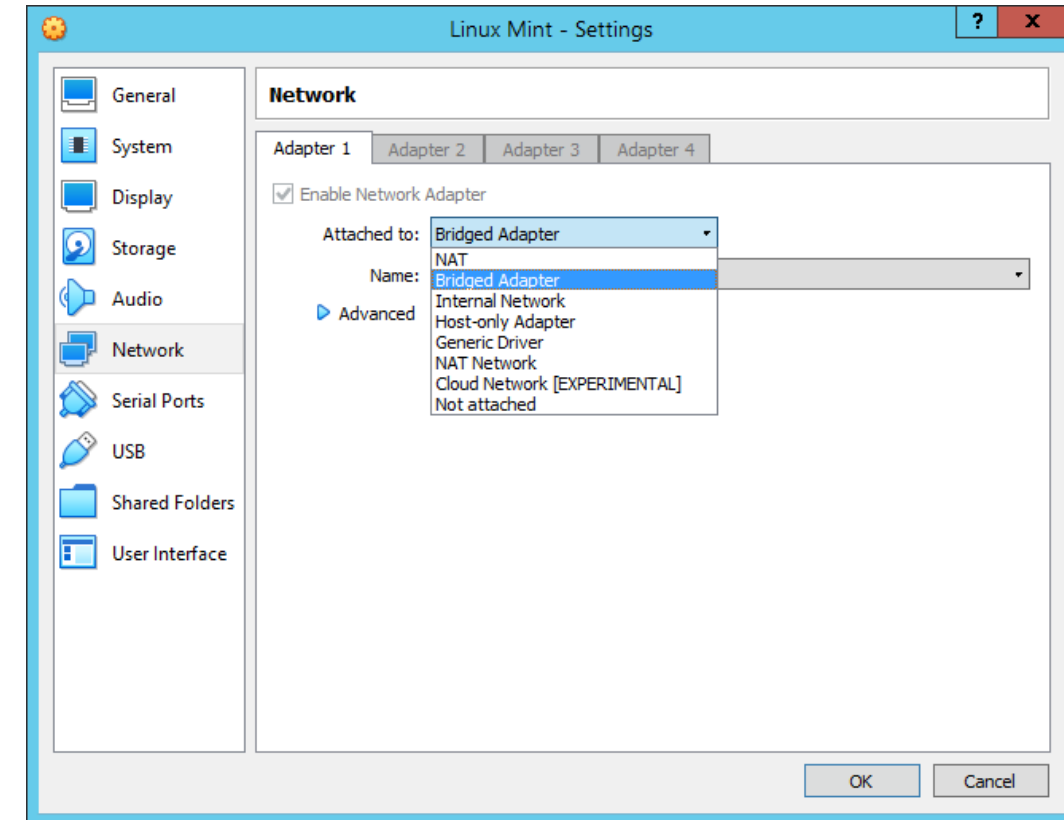
- You can connect to the services exposed by docker from multiple locations and each time the target IP may or may not be different.
  - If you try to connect to the container from within the guest OS (VM) you have to enter the IP of the container.
    - If we set the command parameter `-p 8080:80` then docker maps the port 80 of container to port 8080 of VM and we can use the guest OS (VM) IP instead of container IP.
    - Otherwise the NAT functionality of Docker will allow us to use the container IP on guest OS by translating between the two networks.
  - If you try to connect to the container from the host OS you have to enter the IP address of the guest OS (VM) and the container inside must have its port mapped to guest OS as mentioned above.
    - If you choose bridged adapter on Virtual Machine settings then the IP address of guest OS will be in the same address space as host OS, otherwise it will be in a different address space and NAT functionality of VirtualBox will translate between these two networks.
  - If you try to connect to this container from a different container on the same Docker internal network you have to use the internal IP assigned by Docker to this container.
    - Docker allows you to assign a network alias to each container so that each container can access each other by name, without needing to know the automatically assigned IP to the containers by the internal DHCP of Docker.



# VirtualBox Network Settings

- If you choose **NAT network**
  - VirtualBox will create a virtual network for your Virtual Machine which is different from your host OS network (i.e. LAN).
  - These networks will have IP spaces of their own, and VirtualBox will act as a translation layer between the two.
- If you choose **Bridged Adapter**
  - your Virtual Machine will act just like another machine which is directly connected to the same network as your host.
  - They will share the same IP space, therefore if you assign an IP to your VM which is already used on your LAN you'll get an IP conflict.

	VM ↔ VM	VM → Host	VM ← Host	VM → LAN	VM ← LAN
Not attached	–	–	–	–	–
NAT	–	+	Port Forward	+	Port Forward
NAT Network	+	+	Port Forward	+	Port Forward
Bridged	+	+	+	+	+
Internal Network	+	–	–	–	–
Host-only	+	+	+	–	–



# Different VM Network Options

- There are two different NAT zones on the diagram to the right.
  - The first one is behind the router/firewall which uses the private address range 192.168.X.X.
    - The NAT address translation is performed by the firewall/router.
    - In domestic settings the ADSL modem provides the router, firewall, NAT, local DNS and local DHCP services.
  - The second one is provided by VirtualBox to the VMs (private IP range 172.16.1.X)
  - If you run Docker on the VMs, the Docker will provide a third layer of Network Address Translation
- Observe the difference between the Virtual Adapter which provides bridged networking and NAT

