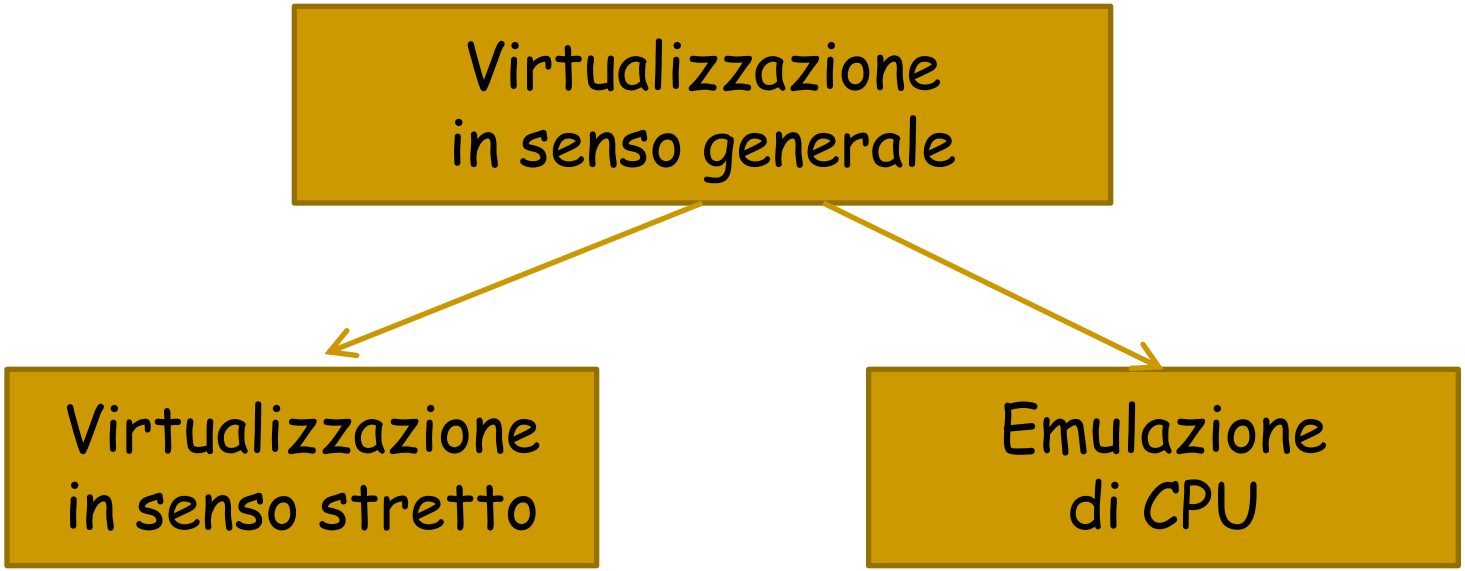
Virtualization & Container & Docker

Sistemi Operativi

Virtualizzazione vs. Emulazione

- distinguiamo due tipi principali di virtualizzazione



Virtualizzazione vs. Emulazione

- Tramite **virtualizzazione** è possibile eseguire uno o più sistemi operativi (ed il relativo software applicativo) da un unico PC, in un ambiente protetto e monitorato che prende il nome di *macchina virtuale (VM)*.
- Il sistema operativo in cui viene eseguita la macchina virtuale, viene detto *ospitante* (host)
- La macchina virtuale è chiamata *ospite* (guest).
- Il codice della macchina virtuale viene eseguito direttamente dal sistema ospitante, ma il sistema ospite "pensa" di essere eseguito su una macchina reale.
  - Il codice della macchina virtuale, perciò deve essere codice macchina eseguibile dalla macchina hardware reale sottostante.
  - Non posso virtualizzare un sistema operativo, che dovrebbe girare su un x86 a 64 bit, su un processore ARM.

- **Emulazione di processore**. In questo caso l'hardware viene completamente emulato dal programma di controllo, cioè ogni istruzione che il sistema guest esegue viene tradotta in una sequenza di istruzioni della macchina host. Il processo di emulazione risulta più lento rispetto alle due forme di virtualizzazione precedenti, a causa della traduzione delle istruzioni dal formato del sistema ospite a quello del sistema ospitante.

Domanda:

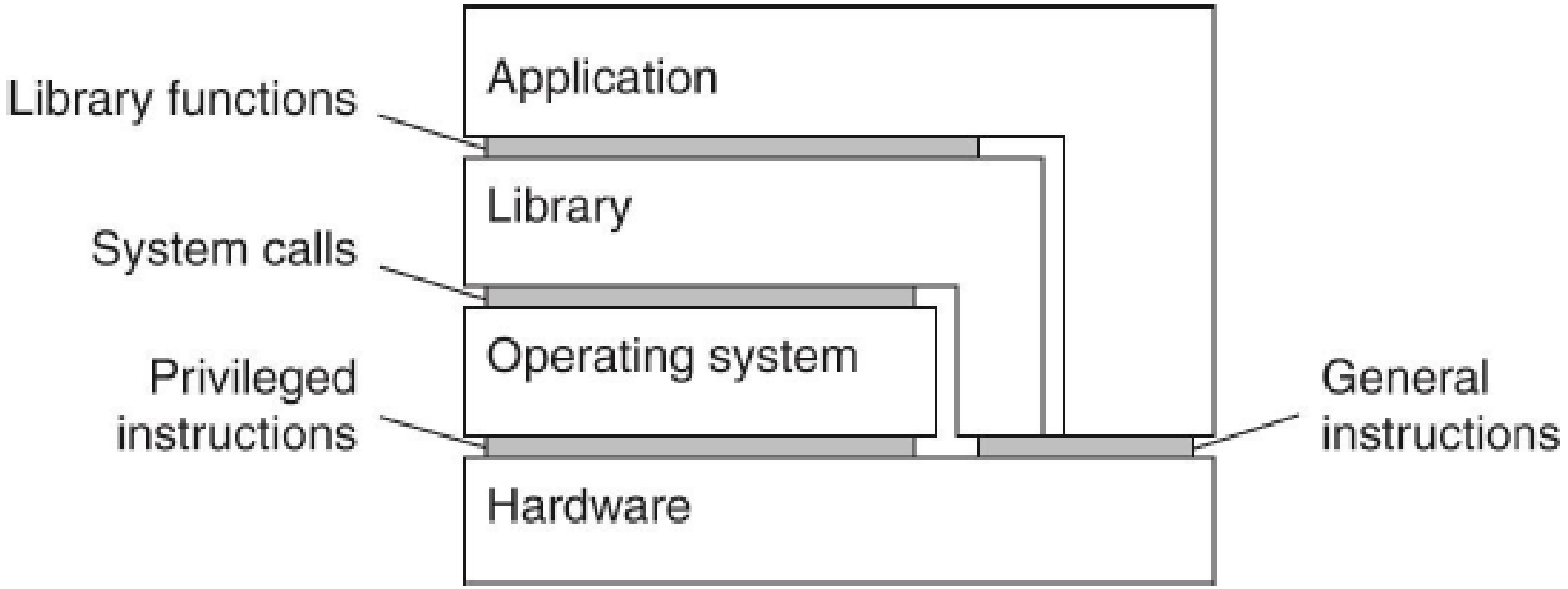
- **Ma allora perchè ad esempio, posso installare una macchina virtuale per x86 a 32 bit su un sistema operativo che gira su un processore fisico x86 a 64 bit ?**

- Puoi farlo perché un processore x86 a 64 bit mantiene la **retrocompatibilità** con le istruzioni a 32 bit, ovvero è in grado di distinguere le istruzioni a 32 e a 64 bit ed è in grado di eseguirle entrambe.
  - In un certo senso, un processore x86 a 64 bit riesce ad emulare automaticamente un processore x86 a 32 bit.

- Tanto è vero che se fate il contrario, cioè cercare di installare una macchina virtuale x86 a 64 bit su un processore fisico x86 a 32 bit, non ce la fate (caso capitato in laboratorio di sist.op. sul portatile a 32 bit di qualcuno).

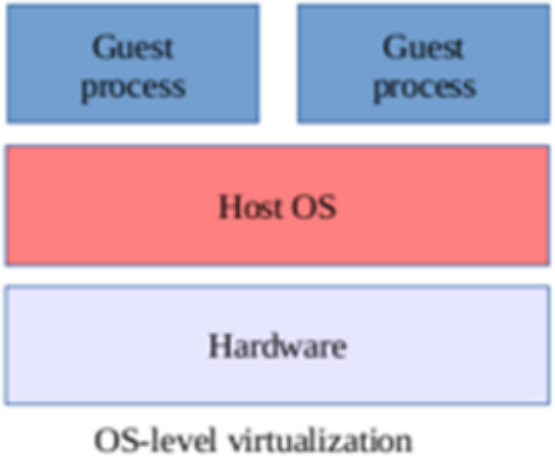
Spazio Kernel e Spazio Utente

- Le moderne CPU prevedono confini di sicurezza (ring).
- Le istruzioni macchina (fornite dal livello ISA) si distinguono in Istruzioni Privilegiate e Istruzioni Generali
- Le istruzioni Privilegiate possono essere eseguite solo dal kernel del sistema operativo, entrando in un apposito Ring di esecuzione della CPU.
- Le istruzioni Generali possono essere eseguite anche dallo spazio utente



Virtualizz. del Livello HW o Livello OS

- La prima grande distinzione riguarda il tipo di risorse virtuali che si vuole presentare all'utente: **macchine virtuali** oppure **partizioni (container) isolate**.
- Nel caso delle **macchine virtuali** (livello Hardware) all'utente del sistema di virtualizzazione viene presentata un'interfaccia su cui installare un sistema operativo, quindi una CPU virtuale (ma dello stesso tipo della CPU fisica) e risorse HW virtuali.
- Nel caso dei **container** (livello OS) all'utente viene presentata una partizione (container) del sistema operativo corrente, su cui installare ed eseguire applicazioni che rimangono isolate nella partizione, pur accedendo ai servizi di uno stesso o.s.



OS-level virtualization

Hardware virtualization

Livello HW vs. Livello OS(container)

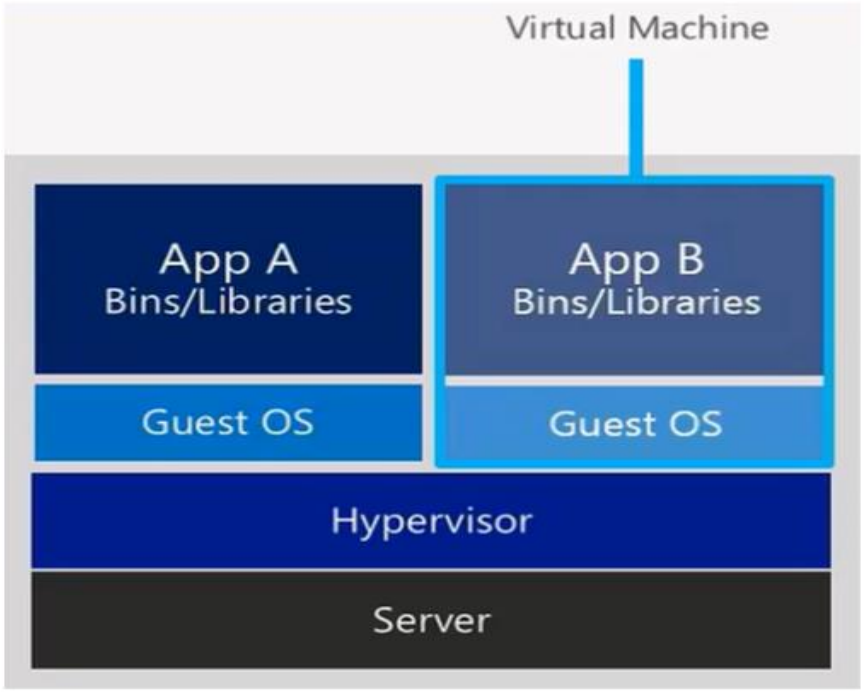
- La virtualizzazione OS-level è composta da un solo kernel (quello del sistema operativo host) e multiple istanze isolate di user-space (chiamate anche partizioni o contenitori), che possono essere avviate e spente in maniera indipendente tra loro. Ogni container contiene un proprio filesystem e proprie interfacce di rete. Viene garantito l'isolamento del filesystem, IPC e network. Inoltre fornisce un sistema di gestione delle risorse quali CPU, memoria, rete e operazioni I/O.
- Nella virtualizzazione hardware, i sistemi operativi eseguono in modo concorrente sullo stesso hardware e possono solitamente essere eterogenei. L'**hypervisor** (chiamato anche *Virtual Machine Monitor*) si occupa di multiplexare l'accesso alle risorse hardware e garantire protezione e isolamento tra le macchine.

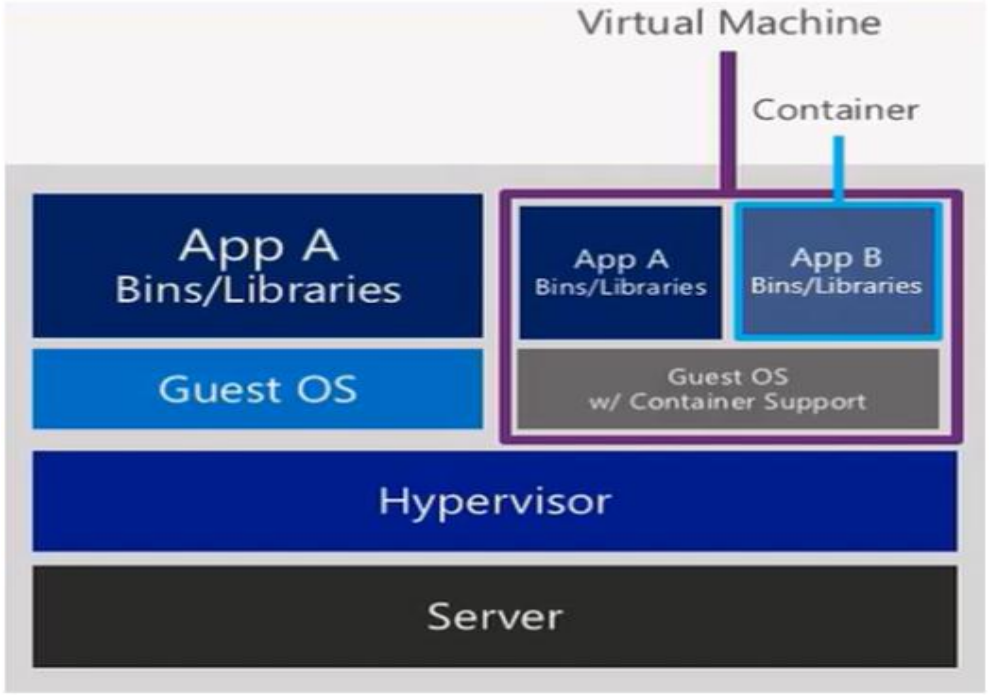
Quindi:

- **OS-level (o container-level)**: offre contenitori per applicazioni ed esegue un solo kernel, quello del o.s. Host.
  - Vantaggi: basso overhead per il context-switch, basso overhead di memoria
  - Svantaggi: non può ospitare sistemi operativi differenti, l'isolamento non può essere del tutto perfetto
- **hardware-level**: quando offre macchine virtuali.
  - Vantaggi/Svantaggi: speculari rispetto a OS-level

2 parole sui Container - perché?

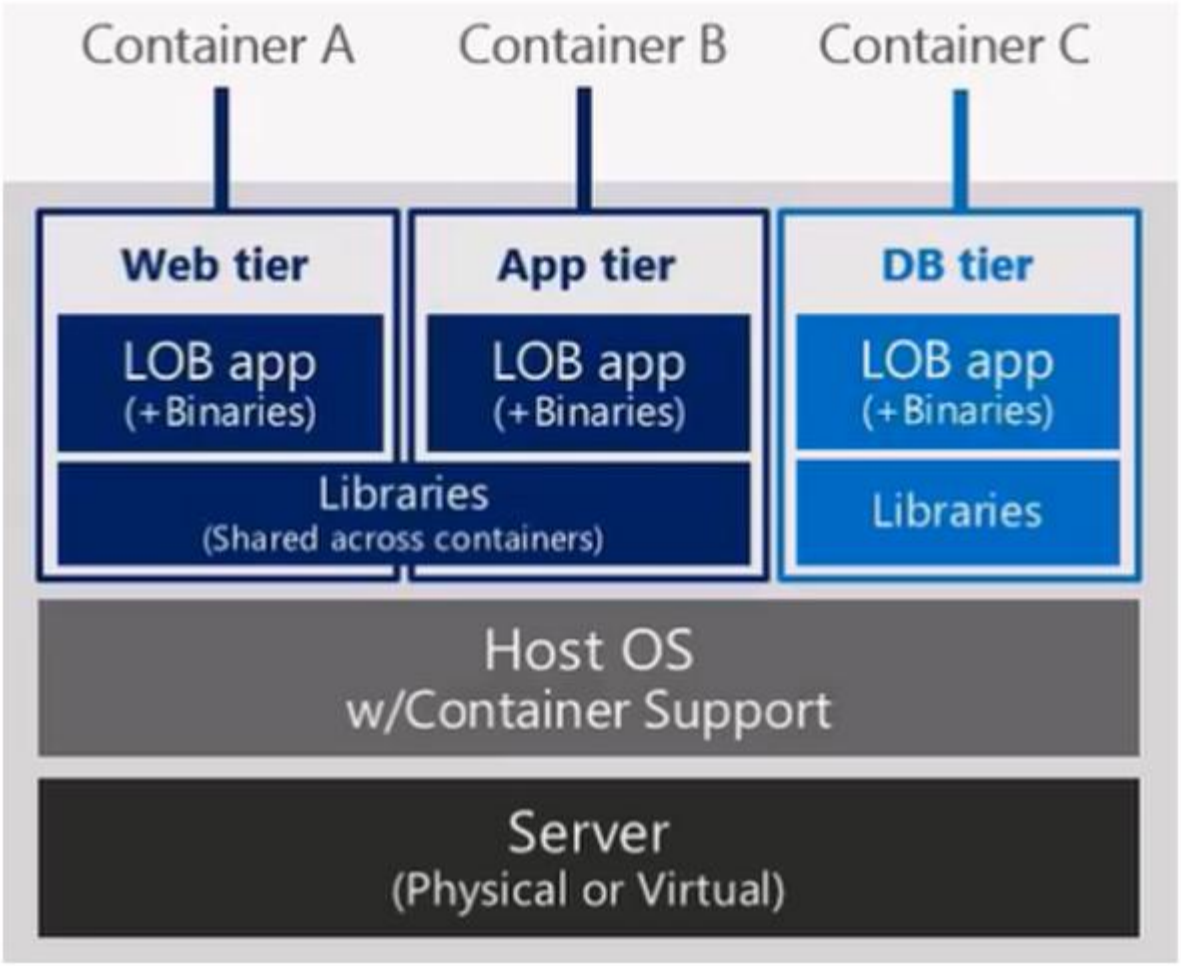
- Creano uno spazio utente isolato, con proprie interfacce di rete, librerie e files, isolando una applicazione dal resto del sistema operativo.
  - **Pacchettizzo una applicazione, rendendola pronta per il deployment.**
- Più container possono appoggiarsi ad uno stesso spazio kernel, risparmiando spazio disco e condividendone i servizi di base.
- Costruito e customizzato un container per una specifica applicazione, posso replicare quel container pià volte, anche su una stessa VM o su VM diverse.





2 parole sui Container - perché?

- Può essere comodo salvare i dati permanenti delle applicazioni di un container su un servizio separato, magari anch'esso contenuto in un container.
- Posso costruire applicazioni composte da tanti micro-componenti riutilizzabili, ciascuno dei quali isolato in un container.



Quali Container ?

Ricordiamo solo i principali:

- **Docker**
  - Si appoggia sul s.o. **Linux** che fornisce, a livello kernel) un supporto per container detto **LXC (Linux Container)**.
    - Non confondete LXC con LXD che è un sistema per dispiegare facilmente virtual machine Linux.
- **Hyper-V** di Microsoft (fornisce unità di isolamento chiamate Container, in realtà sono delle macchine virtuali).
  - Tra l'altro, si parla di container di Hyper-V, ma su sistemi diversi (ad esempio, windows server e windows 10) sono piuttosto differenti, come tipo di isolamento fornito
- **Container di Windows Server** (sono effettivamente dei container)
- NB: E' possibile installare il s.o. Windows server in una configurazione minimale, detto Nano Server, ottimizzato per stare dentro macchine virtuali di hyper-v o per sostenere container di windows server. https://docs.microsoft.com/it-it/windows-server/get-started/getting-started-with-nano-server

Windows Nano Server

Leggere qui per informazioni (06-09-2017)

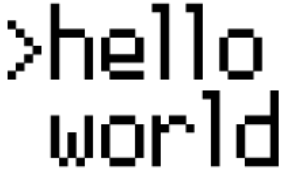
https://docs.microsoft.com/it-it/windows-server/get-started/getting-started-with-nano-server

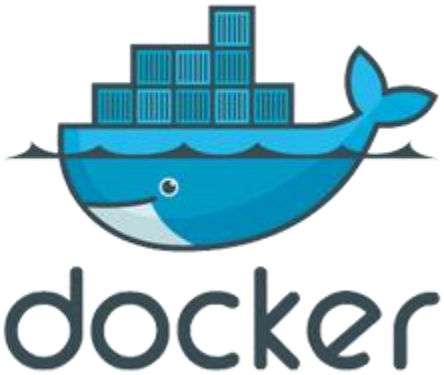
Essenzialmente:

- no interfaccie grafiche.
- solo eseguibili a 64 bit.
- best practices analyzer disabilitato
- consente attivazione automatica senza Product Key
- ...

Livello OS (Container) - Docker

- Virtualizzazione a livello di container su sistemi Linux: **Docker**
- Spessissimo utilizzato per dispiegare micro-servizi su sistemi in cloud.
- Disponibile enorme varietà di immagini di container, spesso su Docker Hub

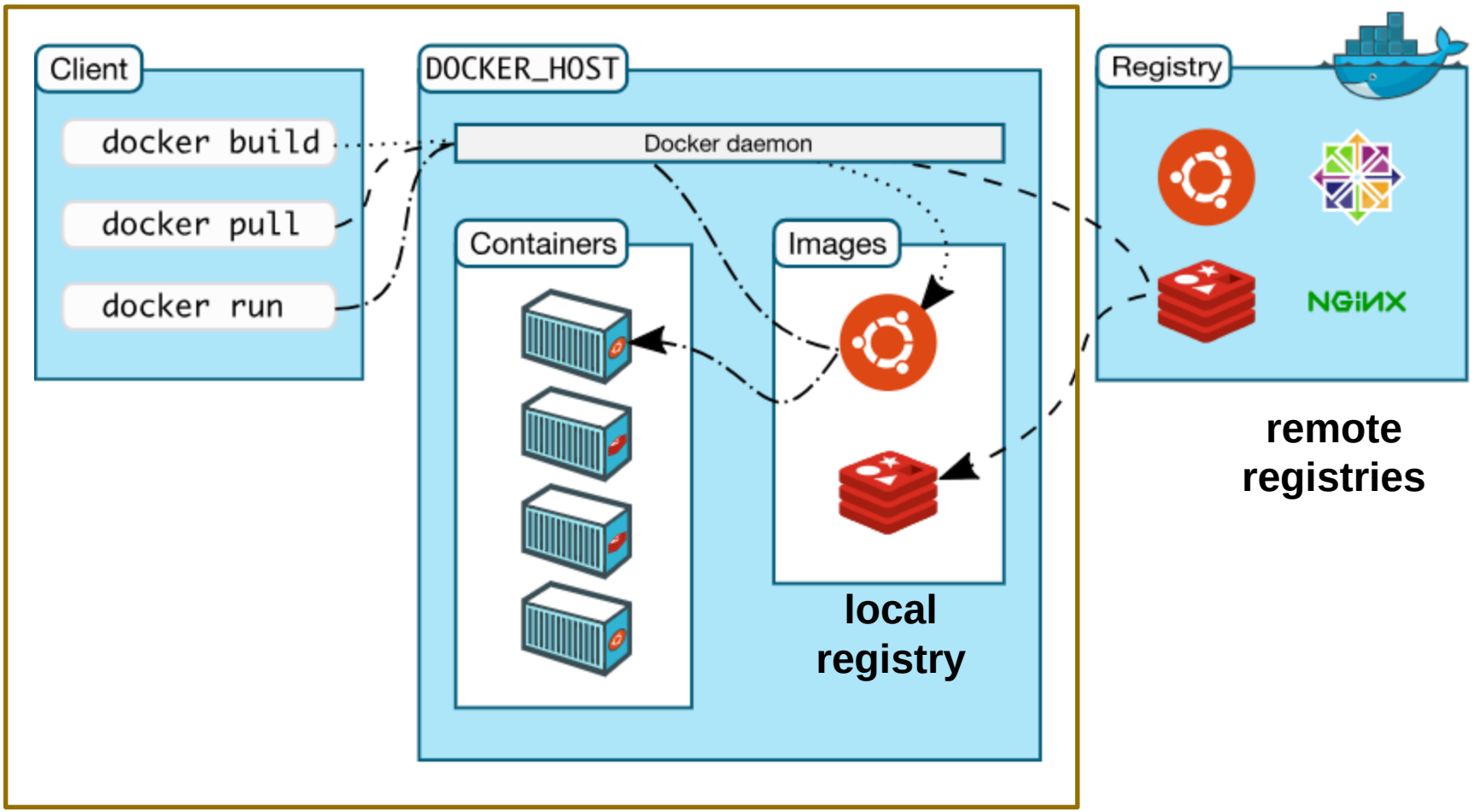




Docker - Contents

- 1. Docker architecture
- 4. Working with Docker images
  - Search, Load, Remove Docker images
- 5. Running a Docker container
  - Exiting Docker container does not remove container
  - Running another container
  - Running bash command in a container
- 6. Managing Docker containers
  - Run, Stop, Start, Remove container
- 7. Commit changes in a container to save a docker image

1. Docker Architecture



2. Working with Docker images (1/3)

- Docker containers are built from Docker images. By default, Docker pulls these images from Docker Hub, a Docker registry managed by Docker, the company behind the Docker project.
- Anyone can host their Docker images on Docker Hub, so most applications and Linux distributions you'll need will have images hosted there.
- You can downloads a container image and run it, by using the run command.
- Once the image downloaded, Docker created a container from the image and the application within the container executed, displaying the message.
- The downloaded images is copied into a local storage (local registry).
- To check whether you can access and download images from Docker Hub, type:

  docker **run** hello-world
- The output will indicate that Docker in working correctly:

```
OutputUnable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

- As you can see, Docker was initially unable to find the hello-world image locally, so it downloaded the image from Docker Hub, which is the default repository.

2. Working with Docker images(2/3)

- You can search for images available on Docker Hub by using the docker command with the search subcommand. To search for the Ubuntu image, type:

  docker **search** ubuntu

- The script will crawl Docker Hub and return a listing of all images whose name match the search string. In this case, the output will be similar to this:

NAME	DESCRIPTION	STARS	OFFICIAL
ubuntu	Ubuntu is a Debian-based Linux operating sys…	9336	[OK]
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface …	283	[OK]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi…	209	[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session…	165	[OK]
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansible	96	[OK]

- In the OFFICIAL column, OK indicates an image built and supported by the company behind the project. Once you've identified the image that you would like to use, you can download it to your computer using the pull subcommand.

2. Working with Docker images(3/3)

- Execute the following command to download the official ubuntu image to your computer, without running the container immediately:

  docker **pull** ubuntu

- After an image has been downloaded, you can then run a container using the downloaded image with the run subcommand.

- To see the images that have been downloaded to your computer, type:

  docker images

- You can remove the images in your computer by using the docker command remove image rmi. The rmi docker command requires a list of images to be removed.
- The generic sintax is:

  docker **rmi** imagename1 imagename2 .... imagenameN
- For example, to remove the ubuntu and httpd docker images, type:

  docker **rmi** ubuntu httpd

3. Running a Docker container interactively

- **Containers can be interactive**. As an example, let's run a container using the latest image of Ubuntu. The combination of the -i and -t switches gives you **interactive shell access** (using stdin/sdtou/stderr) into the container, by running the container in foreground mode:

  docker **run** -it ubuntu

  or

  docker **run** -it --name myubuntu ubuntu

- where the option --name myubuntu assigns the name myubuntu to the container. If no option --name is provided, docker creates and assign a random name to the container.

- Your command prompt should change to reflect the fact that you're now working inside the container and should take this form:

  root@d9b100f2f636:/#

- Note the container id in the command prompt. In this example, it is d9b100f2f636. You'll need that container ID later to identify the container when you want to remove it.

3.1. Running bash commands interactively inside a container

- **Now you can run any command inside the myubuntu container.** For example, let's update the package database inside the container. You don't need to prefix any command with sudo, because you're operating inside the container as the root user:
- Check if the package node.js already exist in your system, but it does not exist

  node -v

  bash: node: command not found
- Install the node.js package.

  apt update
- Then install any application in it. Let's install Node.js:

  apt install nodejs
- This installs Node.js in the container from the official Ubuntu repository. When the installation finishes, verify that Node.js is installed:

  node -v
- You'll see the version number displayed in your terminal: Outputv8.10.0
- Any changes you make inside the container only apply to that container.
- **To exit the container, type exit at the prompt. The shell terminates, the container stops and terminates.**
- If you restart the container, the previous node.js installation has been lost.
- Try again "docker run -it ubuntu" and then run node -v

3.1.1 Stops a container from its interactive bash without terminating it

- Run a container :

  docker **run** -it --name myubuntu ubuntu
- Do some commands from its bash:

  ls; ps; ...
- From the container bash, stop the container without terminating it by typing the stopping sequence.

  Press **CTRL+p** and **Ctrl+q**

3.1.2 Attach to a stopped container

- To attach to a container stopped but not exited, uses the docker command attach and the name or the Id of the container.

  docker attach myubuntu

3.2. Running a container and a specific command

generic docker run syntax:

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

-d	Run container in background and print container ID
-i	Keep STDIN open even if not attached
--name string	Assign a name to the container
--rm	Automatically remove the container when it exits

. . . . .

example:

```
docker run -it --name myubuntu ubuntu /usr/bin/find / -iname '*.sh'
```

when the comman find terminates, the container stops and terminates.

4. Managing Docker containers

- After using Docker for a while, you'll have many active (running) and inactive containers on your computer. To view the active ones, use:

  docker ps

- You will see output similar to the following:

```
CONTAINER ID      IMAGE      COMMAND          CREATED          NAME
....              ...        ....             ....             ....
```

- If you started two containers; one from the hello-world image and another from the ubuntu image. Both containers are no longer running, but they still exist on your system.
- To view all containers — active and inactive, run docker ps with the -a switch:
- docker ps -a
- You'll see output similar to this:

```
CONTAINER ID   IMAGE   COMMAND     CREATED              STATUS                       NAME
d9b100f2f636   ubuntu  "/bin/bash" About an hour ago    Exited (0) 8 minutes ago     myubuntu
01c950718166   hello-world "/hello" About two hours ago Exited (0) About an hour ago festive_williams
```

- To view the latest container you created, pass it the -l switch:

  docker ps -l

```
CONTAINER ID      IMAGE      COMMAND          CREATED          STATUS                       NAME
d9b100f2f636      ubuntu     "/bin/bash"      About an hour ago Exited (0) 8 minutes ago     myubuntu
```

- The -q flag on docker ps command shows the containers ID only

```
docker ps  -a -q
docker stop $(docker ps -a -q)
```

4.1. Stop & Start Docker containers

- To stop a running container, use docker stop, followed by the **container ID** or **container name**, in a different shell of the hosting host.
- docker stop myubuntu
- docker stop d9b100f2f636

- **To start a stopped container, use docker start, followed by the container ID or the container's name**. Let's start the Ubuntu-based container with the ID of d9b100f2f636:

  docker start d9b100f2f636

  or

  docker start -ia d9b100f2f636

  where the flag -i attach container's STDIN

  and the flag -a attach STDOUT/STDERR and forward signals, so as you can give shell commands to the container

- The container will start, and you can use docker ps to see its status:

4.2 Remove Docker containers

- Once you've decided you no longer need a container anymore, remove it with the docker **rm** command, again using either the **container ID** or the **name**. Use the docker ps -a command to find the container ID or name for the container associated with the hello-world image and remove it.

  docker rm festive_williams

  docker rm ec68819568a3

5. Committing Changes in a Container creating a new Docker Image (1/2)

- When you start up a Docker image, you can create, modify, and delete files just like you can with a virtual machine. The changes that you make will only apply to that container. You can start and stop it, but once you destroy it with the docker rm command, the changes will be lost for good.

- Fortunately, you can save the state of a container as a new Docker image.

- As an example, after installing Node.js inside the Ubuntu container, you now have a container running off an image, but the container is different from the image you used to create it. But you might want to reuse this Node.js container as the basis for new images later.
- Then stop the container.
- After the ubuntu container with the node.js package has been stopped, you can commit the changes to a new Docker image instance using the following command.

```
docker commit -m "added node.js" -a "Vic" e602dd6d84c4
    vic/ubuntu_with_node.js
```

5.1. Exercise - create a Docker image from ubuntu with netcat and netstat

apt-cache search netcat — search netcat package - netcat

apt-cache search netstat — search netstat package - net-tools

```
docker run   -it   --name ubuntu ubuntu
    apt update
    apt install netcat
    apt install net-tools
    exit
docker ps -a -q  -f status=exited  -f "name=ubuntu"
docker commit -m added_nc_netstat -a "Vic" $(docker ps -a -q -f
    status=exited -f "name=ubuntu") vic/ubuntu_with_nc_netstat
```

or simply

```
docker commit -m added_nc_netstat -a "Vic" ubuntu
    vic/ubuntu_with_nc_netstat

docker rm ubuntu
```

6. Running a container in Background

-d	Run container in background and print container ID
-i	Keep STDIN open even if not attached

Executes /bin/bash with no stdin, thus it terminates.

```
docker run -d  --name myubuntu ubuntu
docker ps -a
  CONTAINER ID   IMAGE  COMMAND  CREATED   STATUS   PORTS
   NAMES
  37b65ce92fe4    ubuntu  "/bin/bash"  4 seconds ago  Exited (0) 3 seconds ago
   myubuntu

docker run -d  -i  -t  --name myubuntu1 ubuntu
docker ps -a
  CONTAINER ID   IMAGE  COMMAND  CREATED      STATUS          PORTS     NAMES
  fe555f2fb05c     ubuntu  "/bin/bash"  3 seconds ago  Up 1 second              myubuntu1
docker kill myubuntu1
docker ps -a
  CONTAINER ID  IMAGE  COMMAND  CREATED      STATUS          PORTS     NAMES
  fe555f2fb05c   ubuntu  "/bin/bash" 47 seconds ago  Exited (137) 1 second ago     myubuntu1

docker rm myubuntu myubuntu1
```

6.1 Execute interactively a command in a running background container

```
$ docker run -d  -it  --name myubuntu1 ubuntu
$ docker ps -a
  CONTAINER ID  IMAGE  COMMAND  CREATED      STATUS          PORTS    NAMES
  fe555f2fb05c  ubuntu "/bin/bash" 3 seconds ago Up 1 second            myubuntu1
```

docker exec [OPTIONS] CONTAINER COMMAND [ARGS]

-d, --detach	Detached mode: run command in the background
-i, --interactive	Keep STDIN open even if not attached
-t, --tty	Allocate a pseudo-TTY
-u, --user string	Username or UID (format: <name|uid>[:<group|gid>])
-w, --workdir string	Working directory inside the container

```
$ docker exec -it -u root:root -w /usr/local/ myubuntu find ./ -iname '*ma*'
  ./man
  ./share/man

$ docker ps -a
 CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS          PORTS NAMES
 183c45e5d2de  ubuntu "/bin/bash" 10 minutes ago Up 10 minutes      myubuntu
```

6.2 Execute a background command in a running background container

```
$ docker run -d -it --name myubuntu1 ubuntu
$ docker ps -a
  CONTAINER ID  IMAGE  COMMAND  CREATED      STATUS          PORTS  NAMES
  fe555f2fb05c  ubuntu "/bin/bash" 3 seconds ago Up 1 second          myubuntu1
```

```
$ docker exec -d -u root:root -w /usr/local/ myubuntu find ./ -iname '*ma*'
```

```
$ docker ps -a
 CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS          PORTS  NAMES
 183c45e5d2de  ubuntu "/bin/bash" 10 minutes ago Up 10 minutes  myubuntu
```