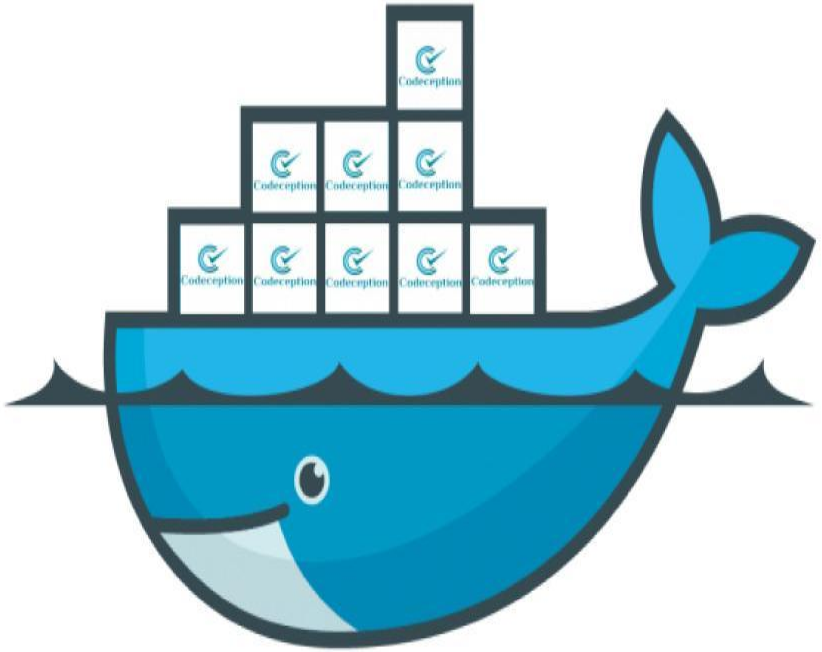
Introduction to Docker Part 1



docker

Requirements

- Have Visual studio code install or use the native terminal (especially for Mac users)
- Have a vagrant Centos 7 Server in your virtualbox
- Connect remotely to the server

NB: All what is done in the course can also be done on an ubuntu server with slight modifications

Table of content

1

What is Docker

Introduction to Docker

What is docker?

- **Docker** is a containerization software.
- **It is used to create and manage containers.**
- A **container** is a standard unit of software that packages up **code** and **all its dependencies**.
- **So the application runs quickly and reliably from one computing environment to another.**

What is docker?

- Containerized softwares are available for both **Linux and Windows-based applications,**
- The softwares will always run the same, regardless of the infrastructure.
- **Containers isolate software from its environment** and ensure that it works uniformly despite differences (**for instance, between development and staging.)**

What is docker: Docker image

- A **Docker container image** is a lightweight, standalone, executable package of software that includes everything needed to run an application: **code**, **runtime**, **system tools**, **system libraries** and **settings**.
- **Container images** become containers at runtime
- In the case of **Docker containers**, images become containers when they run on **Docker Engine.**

What is docker: Docker image

Docker containers that run on Docker Engine are:

- **Standard:** Docker created the industry standard for containers, **so they could be portable anywhere**
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, **driving higher server efficiencies and reducing server and licensing costs**
- **Secure:** Applications are **safer in containers and Docker provides the strongest default isolation capabilities in the industry**

2

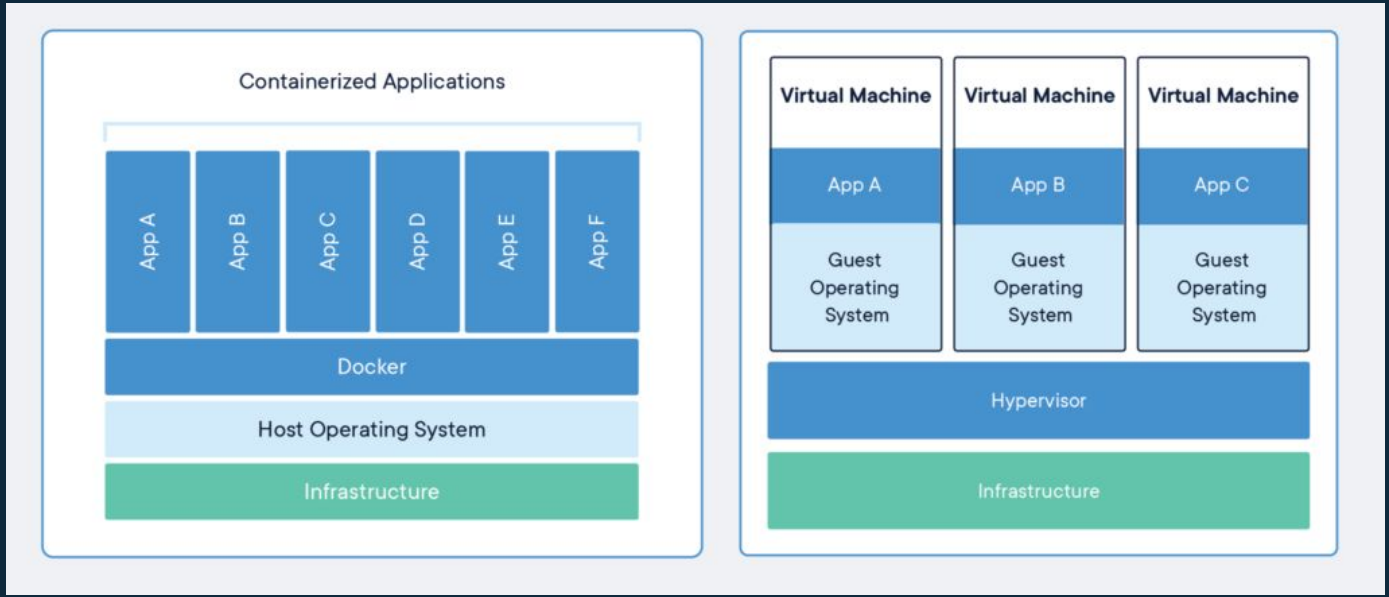
Why using Docker?

Introduction to Docker

Docker vs virtualization architecture

Docker is an improvement of the virtualization architecture

Docker vs virtualization architecture



Let’s bring more explanations on these concepts in simple terms. Starting from the traditional server architecture to the Dockerized architecture

Centos server Charges:

- Price: 5k
- Administration
- Maintenance
- Location (building rent, security, electricity etc.)
- Used size = 40%
- Not used = 60%

Ubuntu server Charges:

- Price: 5k
- Administration
- Maintenance
- Location (building rent, security, electricity etc.)
- Used size = 40%
- Not used = 60%

Debian server Charges:

- Price: 5k
- Administration
- Maintenance
- Location (building rent, security, electricity etc.)
- Used size = 40%
- Not used = 60%

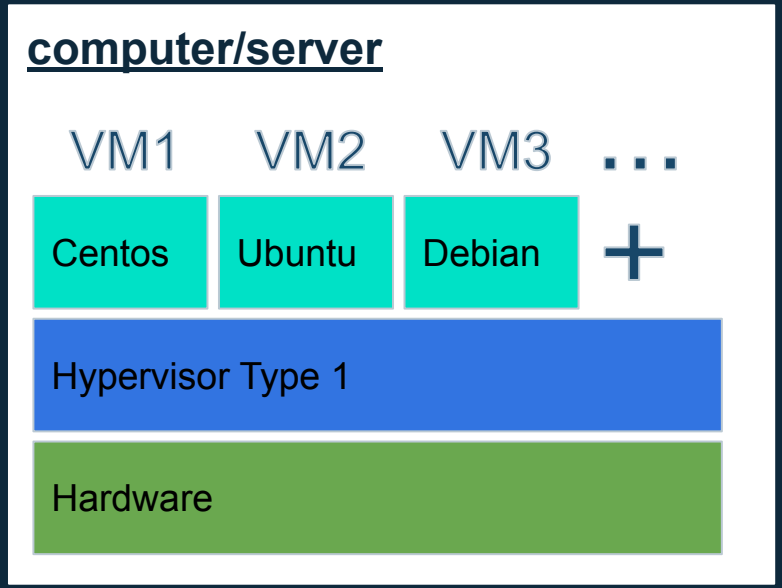
The Traditional server Architecture

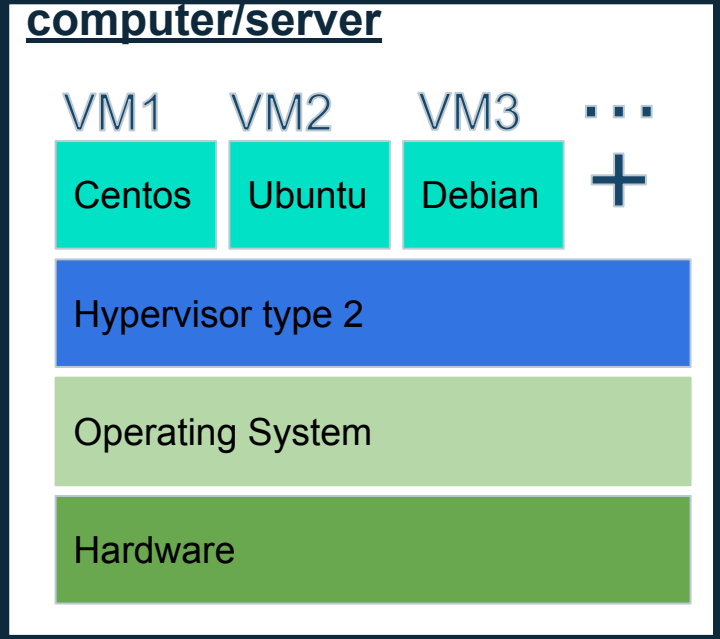
- The company has to buy a new server each time another Operating system is needed.
- Most of the time, not more than **40%** of the total size is used.
- If a server goes down, The company must buy and install a new one (this takes time!)

Virtualization with Hypervisors (type 1 or type 2)

Here, many Operating Systems can run on the same computer.

Each server is installed in one Virtual Machine

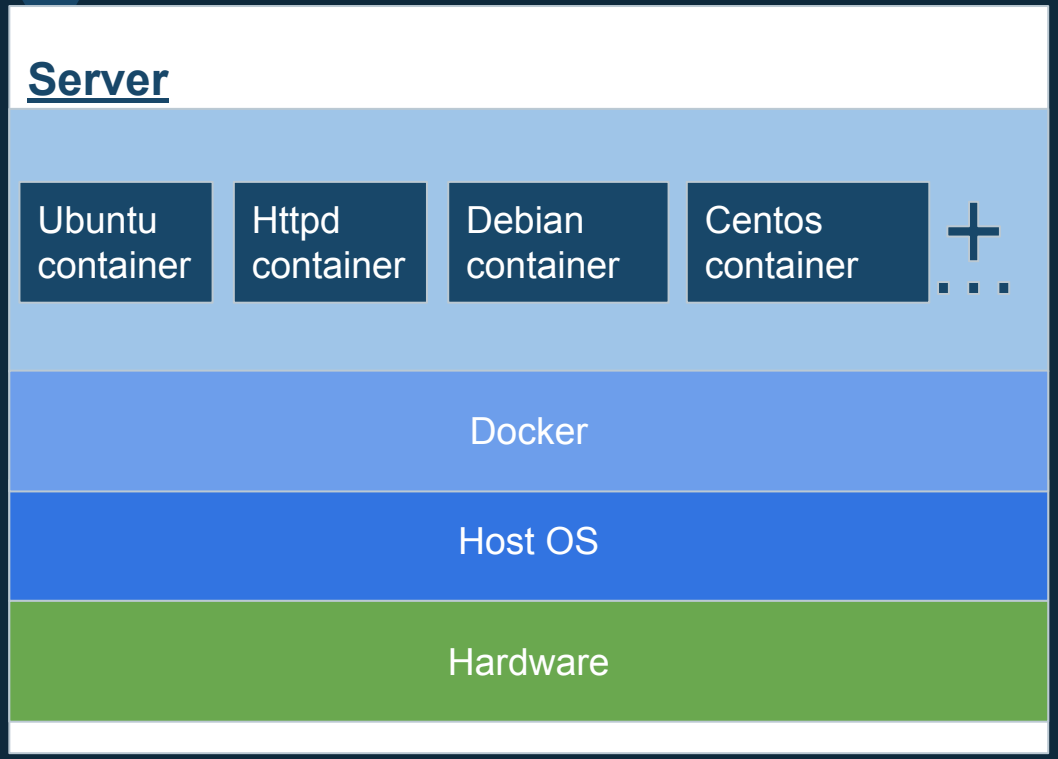




Virtualization with Hypervisors(type 1 or type 2)

- Here you can realize that if a server (a virtual machine) goes down, **you need just some few minutes to delete it and create another one**
- Also, you can customize a virtual server, package it and **move virtual it to be install on a different computer easily** (take for example the project we did in week 4)
- This is **less costly** and has many other advantages.
- This architecture was still improved and that produced **Docker**

Virtualization with Docker

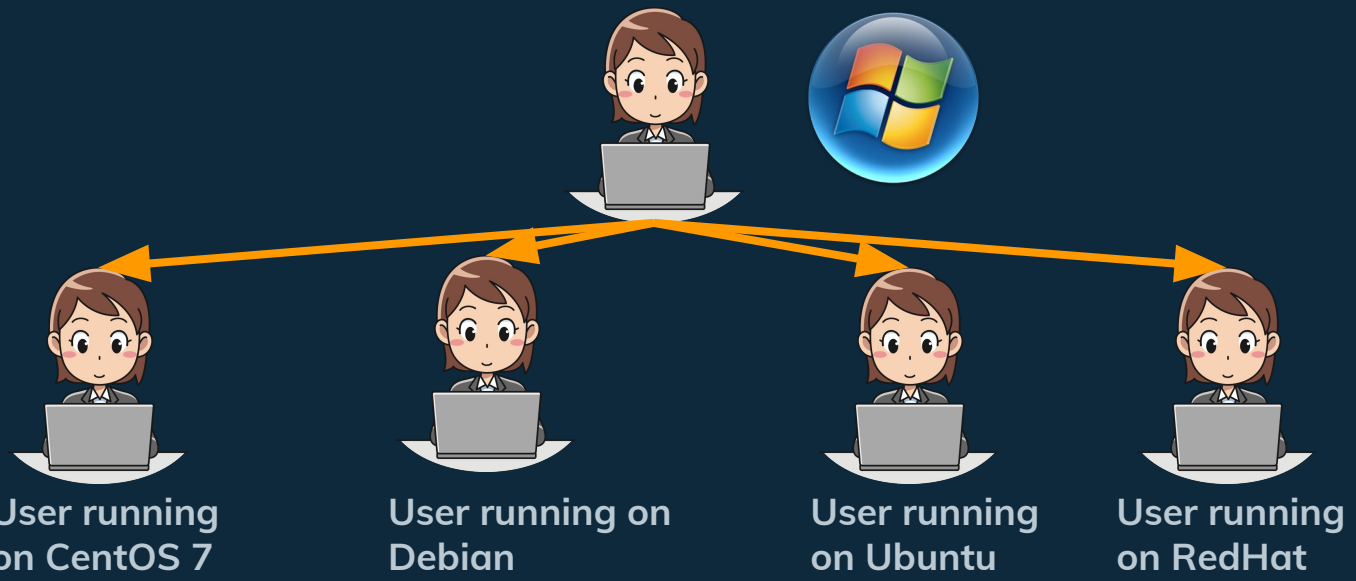


Here you can create multiple Docker containers and in each container you install the image of an OS

This is indeed an effective improvement of the Traditional server Architecture!

A big problem solved by docker

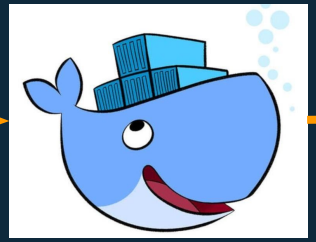
A big problem solved by docker



Either the software breaks and the engineers try to diagnose and fix the errors, or it does not behave the same on every infrastructure

A big problem solved by docker

A user builds a docker image



Docker image

Millions of users running the Docker image in their various infrastructures

The Docker container behaves the same, no matter the infrastructure it is running on

3

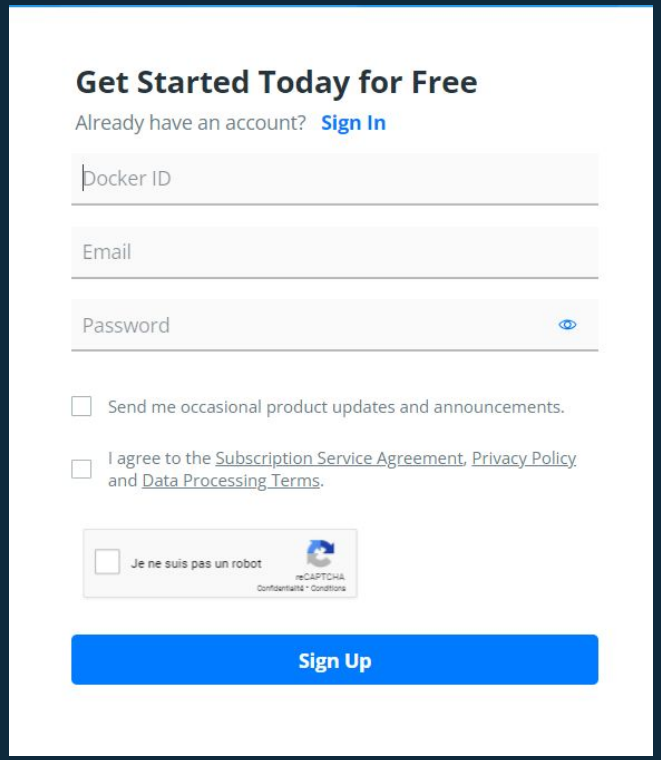
How to use Docker?

Docker Hub, install docker

Sign-up and Sign-in to dockerhub

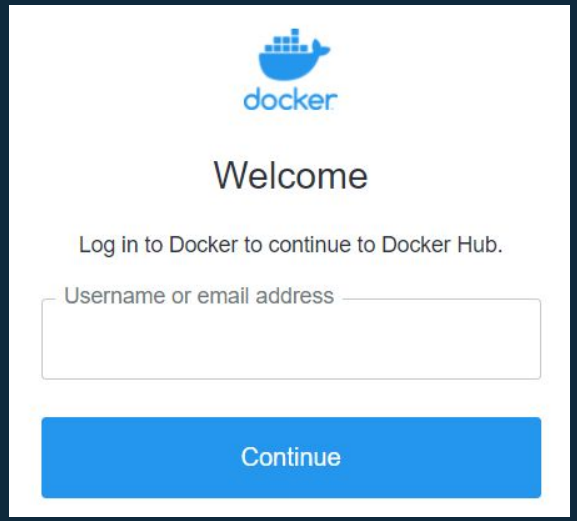
Sign Up to Docker Hub

- Launch your browser
- Open the docker hub site: **hub.docker.com**
- Fill the form and validate
- Note: Don't Check the send me occasional product updates and announcements checkbutton
- Agree to the terms and policy
- If the ID is denied, modify it a little bit and retry

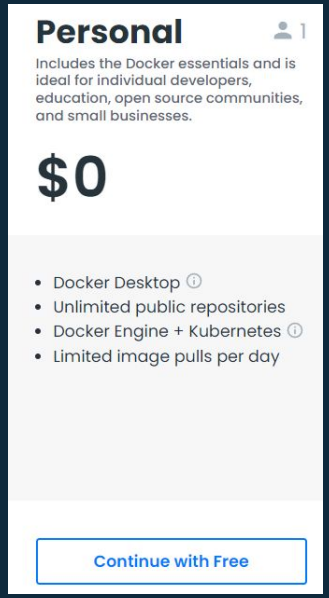


Sign Up to Docker Hub

Now, sign in with your **username** or **email address** and **password**

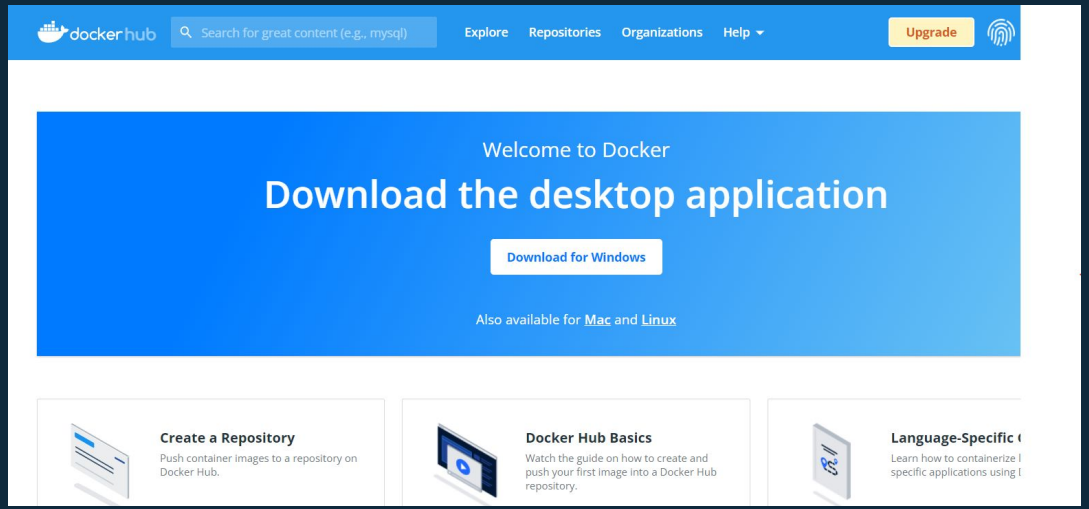


Choose the **Free Personal plan**



Sign Up to Docker Hub

- A mail from the **Docker** team will be sent to the **email address** you signed up with
- Open your mails and click on **Verify email address**
- You are now signed into your **docker hub account**



Install Docker on your server

To install docker with the recommended method, follow the steps on this confluence page: Click here

Login parameters:

username: unixteam24@gmail.com

password: school123

Install Docker

- When you install **Docker** on a Linux server, **a group for it is automatically created**. (**tail -5 /etc/group**)
- Thus, if you want a user to run Docker, **you must add that user to the Docker group**
- Let's create a new user on the system and add it to the docker group
  - **$ sudo useradd student -G docker**
  - **$ sudo passwd student** (Here we use **school1** as password)
  - **$ groups student**
- Switch to the user student: $ **sudo su student**

4

Docker commands

Useful docker commands

Docker info/docker version

To get informations concerning **Docker**,

you can run: **$ docker info**

```
[student@localhost vagrant]$ docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.7.1-docker)
  scan: Docker Scan (Docker Inc., v0.12.0)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 20.10.12
```

Now, you can see there is nothing in it. Cause we have not yet created any container.

To know the version of docker you are running, use:

$ docker version

```
[student@localhost vagrant]$ docker version
Client: Docker Engine - Community
 Version:           20.10.12
 API version:       1.41
 Go version:        go1.16.12
 Git commit:        e91ed57
 Built:             Mon Dec 13 11:45:41 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
```

Docker images

- To check all the images running on our Docker, we run the command:

  $ docker images

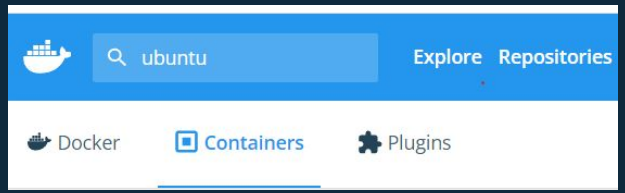
```
[student@localhost vagrant]$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

- For now there is no docker image in the system
- The images that will be displayed are images we have used to create containers on our system
- To create a container using docker, we need to **pull an image** from **Docker Hub (hub.docker.com)** and run it on our server

Docker pull

Let's pull a **Ubuntu Docker image** from the **docker hub.**

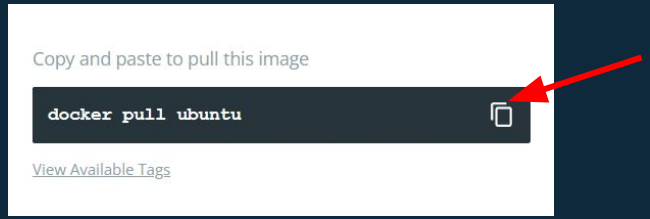
- **Sign in** into your account with the credentials you used to sign up (ie the email and password of your account on docker hub)
- In the search bar, enter **Ubuntu** and validate



Docker pull

- Now, choose the **Official ubuntu** image

- **Copy** the command to pull the image and **paste** it in your terminal
  - **$ docker pull ubuntu**



Docker pull

- Always make sure the **Docker daemon is up and running**. If not, you will get an **error message**.

Let's check the docker images once more:

$ docker images

```
[student@localhost vagrant]$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
ubuntu        latest    54c9d81cbb44   3 weeks ago   72.8MB
```

Image ID

Docker pull : TAGS

What is the meaning of TAGS here?

- A **tag** corresponds to a version of an OS docker image that you can download. When you don't choose one, the **latest** is pulled
- You can click on the **Tags** menu under the image to check the various tags of that docker image.

Exemple: In Docker hub, you can take the **Ubuntu18.04 tag**, copy the command to pull it and paste it in your terminal to install ubuntu 18.04

Docker ps

- To list the containers running on the system, we use the command **docker ps**
- This command displays various elements on the container:
  - The container ID
  - The image that was used to create that container
  - The command or program that is running in the container
  - The creation date
  - The status
  - The ports that are opened in the container
  - The name of the container
- If a container is created with all these informations, it will be displayed when running **docker ps**

Docker run

- To run a container using the ubuntu image we previously pulled, we use the syntax: **docker run [Image_ID]**
- Use the **image ID** you have (from the # **docker images** command)

```
[student@localhost vagrant]$ docker run 54c9d81cbb44
[student@localhost vagrant]$
```

- To get all the options for this command, you can run: $ **docker run --help**
- To check if the container is running run $ **docker ps**

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS   NAMES
[student@localhost vagrant]$
```

- Here, you won't see any running container in the list. **Why?**

Docker run

- Containers with the status **Exited** are not displayed when we run **docker ps**
- To show all the running and even the **exited** containers, you can run:

  $ docker ps -a

- If you run **docker ps -a** now, you will see our ubuntu container with the status **Exited**

```
[student@localhost vagrant]$ docker ps -a
CONTAINER ID   IMAGE          COMMAND   CREATED        STATUS                     PORTS     NAMES
a6ea8435be37   54c9d81cbb44   "bash"    2 minutes ago  Exited (0) 2 minutes ago           inspiring_moser
```

Docker run

- The Ubuntu container exited because **no process was running in it** and **no user was logged into that container.**
- If you don't want a container to exit, you can run it with some specific options like : **docker run -it [Image_id] bash**
- Here **i** means **interactive** and **t** is for **terminal**.
- With this command, we are opening a **bash shell interactive terminal** inside the container and login into it.
- After running the command, you will be logged into the container

Docker run

- Use the $ **docker images** command to get the **image ID**
- **Example:** $ **docker run -it 54c9d81cbb44 bash** will run a new container using the **image ID provided**. A **bash shell terminal** will be opened in the container as soon as it is provisioned.
- Now, you are inside the container,

```
[student@localhost vagrant]$ docker run -it 54c9d81cbb44 bash
root@da0be6f24a1a:/#
```

- You can run some linux commands to check: **# pwd, # ls, cat ...**
- To exit this container's environment, just type: **# exit**

After exiting the container, if we run **docker ps**, it will not display in the list.

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[student@localhost vagrant]$
```

If you run $ **docker ps -a** you will find it in **Exited** status

Docker run: detached mode

- For a container to continue running even when we are out of it, we need to run it in **detached mode**
- To do that, we add the **-d** option: **docker run -itd imageID**

  Example: **$ docker run -itd 54c9d81cbb44** (use $ **docker images** to get your own **imageID**)

```
[student@localhost vagrant]$ docker run -itd 54c9d81cbb44
ea7a701f9b9b8af4ddf720428b80fb4fb4a2139941b119115d8f38254189a137
```

- This command will not log you into the container automatically but will display the **full ID of the container created**
- Now, you can run the command $ **docker ps** to check the status of the container

Docker run: options

- You can run a container with specific ports (internal and external) and a specific name following the syntax:

docker run -itd --name=giveAname -p internal:external imageName

- You can choose to put the image name from docker hub or to use an image ID on your system

Example: $ docker run -itd --name=u_serge -p 82:80 ubuntu

- You can use **-p** twice to expose two ports:

Example: $ docker run -itd --name=serge -p 85:80 -p 89:80 ubuntu

- Use $ **docker ps** to list the containers

Docker exec

- Everytime you run a container with the **-it** option, the system automatically logs you into that container. To log out, you need to exit.
- When you run it adding the **-d** option, it does not log you into the container automatically

How can we run a command in a container while being outside of it?

- We use the **docker exec** command
- The syntax is as follows: **$ docker exec containerID yourCommand**

Docker exec

- **Example:** For this example, you need to copy the ID of the container in which you want to run the command (**docker ps**, select and copy that ID)

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE          COMMAND   CREATED         STATUS         PORTS                                   NAMES
7beaa0151d86   ubuntu         "bash"    3 minutes ago   Up 3 minutes   0.0.0.0:82->80/tcp, :::82->80/tcp       u_serge
ea7a701f9b9b   54c9d81cbb44   "bash"    7 minutes ago   Up 7 minutes                                           festive_mendel
```

- **$ docker exec putTheIDhere cat /etc/*release**

```
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
```

- **Exercice:** Run the commands in a container: **nproc** and **uname -r**

Docker attach

- To log into a specific container, use: **docker attach containerID**
- Log in into the ubuntu server (copy the **container ID** from the **docker ps** command)

```
[student@localhost vagrant]$ docker attach 7beaa0151d86
root@7beaa0151d86:/#
```

- Now you can see that the prompt give us **root@containerID**
- Run **apt-get update** to update the system in the container
- Use **exit** to get out of the container

Docker login

- You can log into your **docker repository** from the terminal using the command: **$ docker login**
- Enter your **username (Docker ID)** and **password** **(**the one you used while creating your account on Docker Hub**)**

Example:

```
Login with your Docker ID to push and pul
Username: adess99
Password:
Login Succeeded
```

Docker login

- When you build your own docker image, you can push it to your docker hub account.
- All the images you have built will then be in you **docker repository.**
- By default, when you create an account, it creates a repository with your name.
- For now you don’t have anything in your docker Hub repository/registry

Docker stop

- To **stop a container**, we use the command: **docker stop containerID**
- You can run **docker ps** to get the ID of the container you want to stop

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE          COMMAND   CREATED          STATUS          PORTS     NAMES
ea7a701f9b9b   54c9d81cbb44   "bash"    19 minutes ago   Up 19 minutes             festive_mendel
b18b0c90c237   54c9d81cbb44   "bash"    20 minutes ago   Up 20 minutes             charming_booth
```

- Let's stop the a container: $ **docker stop ea7a701f9b9b**
- The **docker ps -a** will show that the container that you previously stopped is **exited**

```
ea7a701f9b9b   54c9d81cbb44   "bash"    20 minutes ago   Exited (0) 14 seconds ago   festive_mendel
```

- You can **stop many containers with the same command:**

$ docker stop containerID1 containerID2 containerID3 … containerIDn

Docker start

- To start or restart a container you previously stopped, use the command
  $ **docker start containerID**

  Example: Let's restart that container we stopped earlier:

- **$ docker start ea7a701f9b9b**
- **$ docker ps** to check the status

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE          COMMAND   CREATED          STATUS          PORTS     NAMES
ea7a701f9b9b   54c9d81cbb44   "bash"    25 minutes ago   Up 4 seconds              festive_mendel
b18b0c90c237   54c9d81cbb44   "bash"    25 minutes ago   Up 25 minutes             charming_booth
```

Docker rm/docker rmi

- To clean up and delete a container, we use: **docker rm containerID**
- You can also do it for many containers at once:
  - **$ docker rm containerID1 containerID2 … containerIDn**
- To **remove an image**, we use: **docker rmi imageID**
- Before deleting and image, you must make sure that it is not used by any container
- You can use the option **-f to force the removal**
- **Check all that with docker ps, docker ps -a and docker images**

5

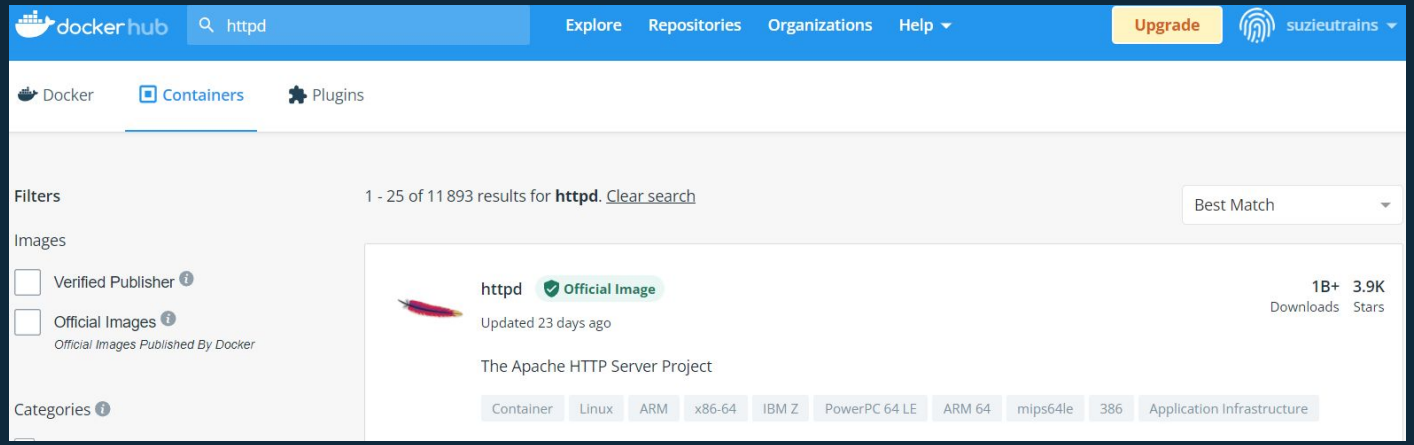
Practice on httpd docker image

From Docker hub

Httpd image

Let's run a **httpd** container (**web server**)

- In the docker hub site, search for **httpd** official image.



Httpd image

- You can read the description to better understand how to use that image
- Now, pull the image using the command: $ **docker pull httpd**
- Now, run the image in detached mode:

  $ docker run -itd --name=web -p 85:80 httpd

```
[root@localhost vagrant]# docker run -itd --name=web -p 85:80 httpd
e834cee028ea7166f93061c7a359681b44aa726ba3b80072f74ae91c2fcc6212
[root@localhost vagrant]#
```

- Remember **Apache** need to be running on a specific port (80).
- Also make sure the internal port you will use is not yet taken by another container (here we use the port 85.)

Httpd image

- The **# docker images** command will show that **there is a new image on the system**

```
[student@localhost vagrant]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       latest    54c9d81cbb44   3 weeks ago    72.8MB
httpd        latest    a8ea074f4566   4 weeks ago    144MB
```

- The **docker ps** command will show the new container (web) running **httpd**

```
[student@localhost vagrant]$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED              STATUS              PORTS                                   NAMES
9cac82d010cb   httpd     "httpd-foreground"   About a minute ago   Up About a minute   0.0.0.0:85->80/tcp, :::85->80/tcp   web
```

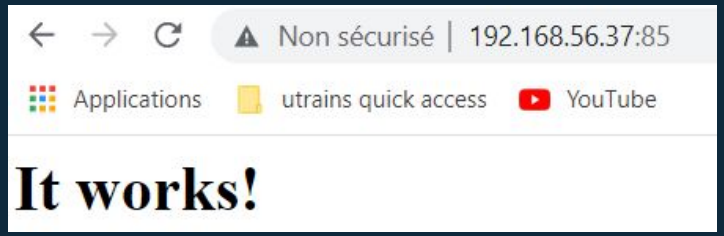
Httpd image

- To make sure this container works as expected, run the command

  # ifconfig, look for **eth1** and copy your **IP address**

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.37  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe47:736d  prefixlen 64  scopeid 0x20<link>
```

- Paste it in the browser precising the port (85)



- You can run another **httpd** on another port (like port **86**) and test it too

Httpd image: Ports

- The port number (the port on which the traffic enters) helps the system to know in which container it must transfer that traffic.

- The traffic from the host may be on differents ports , but **the Apache server is listening only on port 80**

Try to launch the browser with a different port like 87 and see what you get

Always remember to clean up the system by stopping the containers and deleting the ones that you are not using

This might be difficult to understand at first but re-watch the video if necessary and practise to better understand. As always you can make some research to improve your knowledge.

If you get stuck somewhere, always feel free to post your questions in the group.

See you guys in the next lesson! 

Thanks!

Any questions?

You can find us at:

website: http://utrains.org/

Phone: +1 (302) 689 3440

Email: contact@utrains.org

Click on the link below to contact the support team for any issue!

utrains.org/support/

Create a ticket for your problem and we will get back to you soon!