**Install procedures screenshots**

Table des matières

[Check Windows 10 version 2](#_Toc77197284)

[Update Windows 10 version 3](#_Toc77197285)

[Check if WSL is installed 5](#_Toc77197286)

[Check if virtualization is enabled 5](#_Toc77197287)

[Install Windows Subsystem for Linux 7](#_Toc77197288)

[Update the Linux kernel 8](#_Toc77197289)

[Install a Linux distribution 10](#_Toc77197290)

[Update Ubuntu install 14](#_Toc77197291)

[Locate the Linux filesystem 15](#_Toc77197292)

[Install Docker Desktop 15](#_Toc77197293)

[Locate Docker Desktop files 17](#_Toc77197294)

[Test Docker install on Windows 18](#_Toc77197295)

[Test docker install on Linux 20](#_Toc77197296)

[Start and Stop Docker and WSL memory consumption 20](#_Toc77197297)

[WSL 2 Configuration file 24](#_Toc77197298)

[WLS 2 memory management 25](#_Toc77197299)

[Docker Desktop – Deploy local registry 26](#_Toc77197300)

[Rancher k3d – Create a local k3s cluster in containers 26](#_Toc77197301)

[Set up k3s local-path provisioner 27](#_Toc77197302)

[Deploy YugabyteDB with operator 28](#_Toc77197303)

[Set up YugabyteDB authentication 29](#_Toc77197304)

[Set up YugabyteDB ports 29](#_Toc77197305)

[Deploy YugabyteDB – detailed procedure 29](#_Toc77197306)

[Get machine and WSL IP addresses 31](#_Toc77197307)

[Delete k3d cluster 32](#_Toc77197308)

[Delete k3d cluster – Delete persistent storage 32](#_Toc77197309)

[Docker – Set up a local registry 32](#_Toc77197310)

[k3d cluster – Configure local registry 33](#_Toc77197311)

[Create k3d cluster – Yugabyte & Postgresql versions 34](#_Toc77197312)

[k3d cluster - Test local registry + ingress 34](#_Toc77197313)

[Docker - Size of images, containers, volumes 35](#_Toc77197314)

[Login to yugabyteDB from Visual Studio 35](#_Toc77197315)

[Configure remote kubectl access 36](#_Toc77197316)

[Install Postgresql with Crunchy Operator 38](#_Toc77197317)

[Postgresql - Connect from Visual Studio 41](#_Toc77197318)

[Study of memory used 42](#_Toc77197319)

[Set up SSH connection to WSL 45](#_Toc77197320)

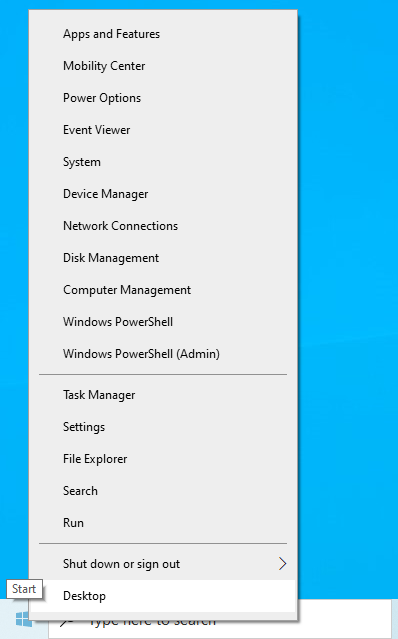
[Diagnose Windows firewall problem 45](#_Toc77197321)

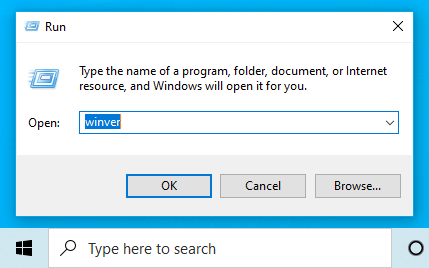
[Install Rancher Longhorn => KO 46](#_Toc77197322)

[Deploy ASP.Net core to Kubernetes 47](#_Toc77197323)

[Deploy Kubernetes dashboard 48](#_Toc77197324)

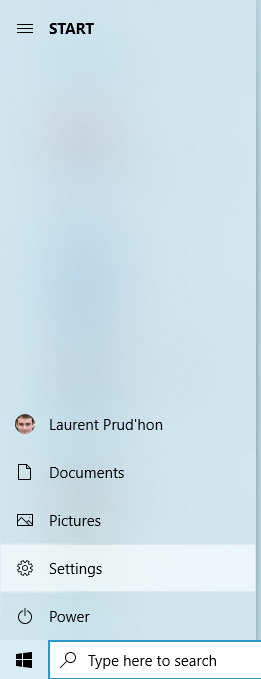
# Check Windows 10 version

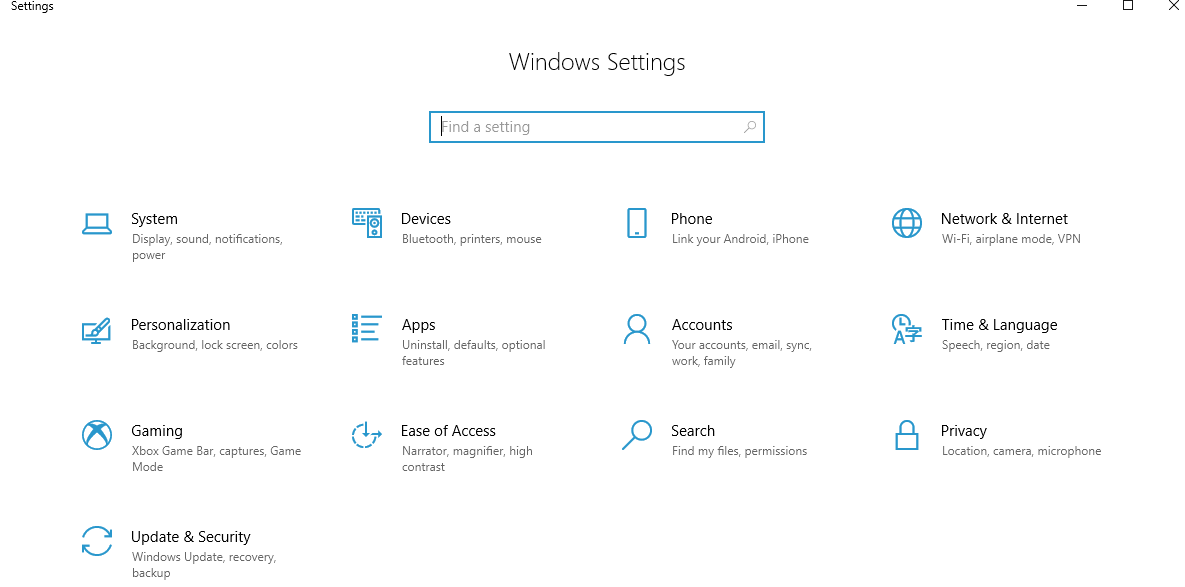


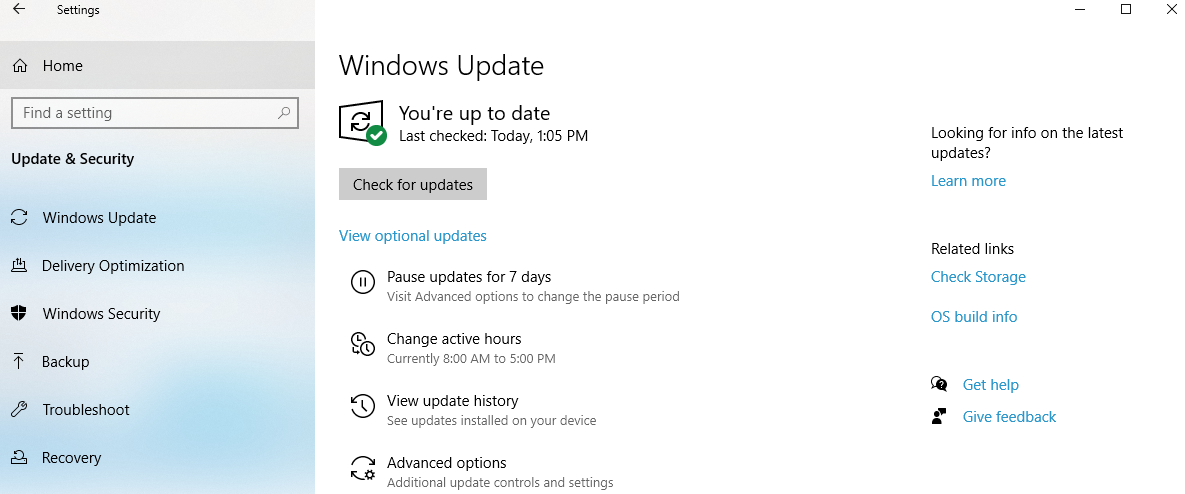




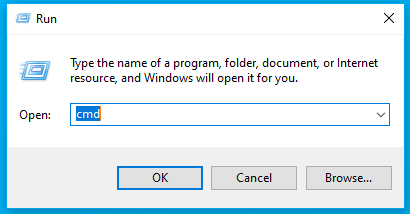
# Update Windows 10 version

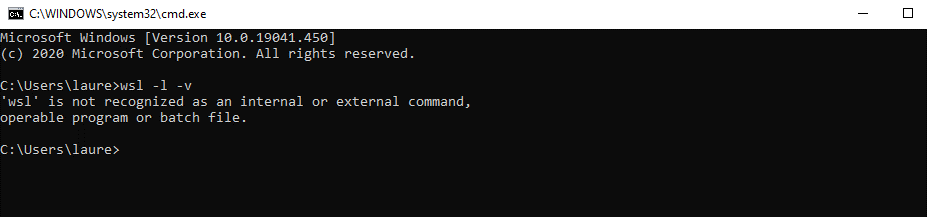




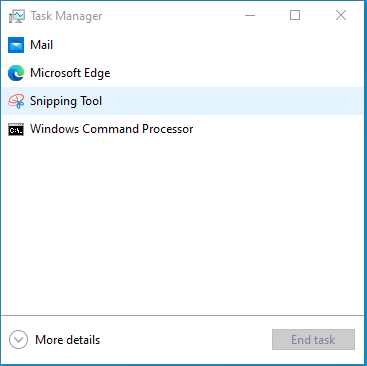


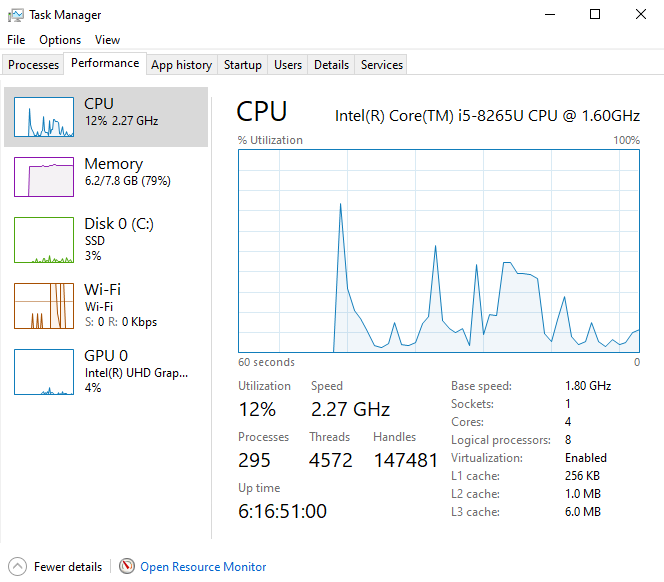
# Check if WSL is installed





# Check if virtualization is enabled





The easiest way to check is by opening up Task Manager->Performance Tab. You should see Virtualization as shown in the below screenshot. If it is enabled, it means that your CPU supports Virtualization and is currently enabled in BIOS. If it shows disabled, you need to enable it in BIOS. If you don’t see virtualization, it means that your CPU does not support virtualization.

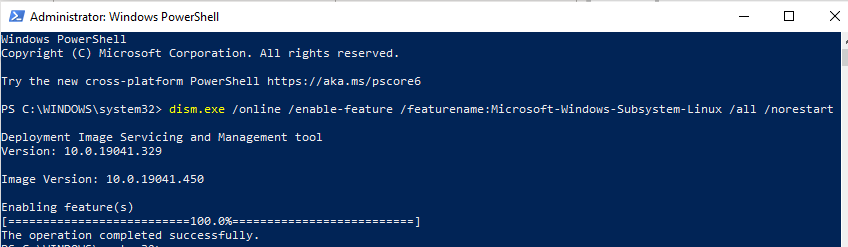
<https://www.laptopmag.com/articles/access-bios-windows-10>

# Install Windows Subsystem for Linux

Before installing any Linux distributions on Windows, you must enable the "Windows Subsystem for Linux" optional feature.

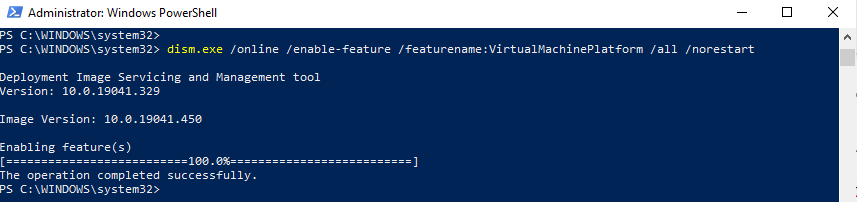
Open PowerShell as Administrator (from start menu, accept elevation prompt)

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart



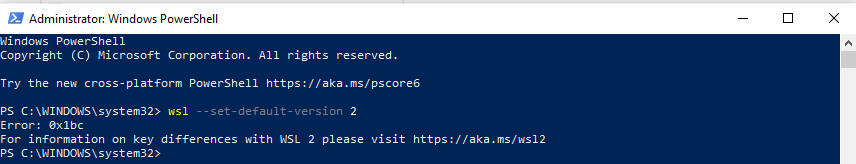
Before installing WSL 2, you must enable the "Virtual Machine Platform" optional feature.

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart



**Restart** your machine to complete the WSL install and update to WSL 2.

Open PowerShell as Administrator and run this command to set WSL 2 as the default version when installing a new Linux distribution:



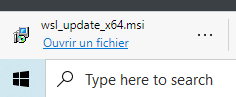
# Update the Linux kernel

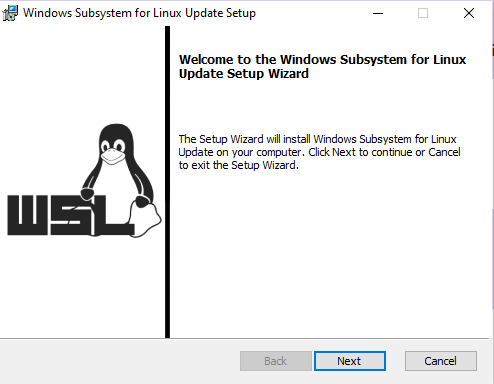
Please follow the link (<https://aka.ms/wsl2kernel>) and install the MSI from that page on our documentation to install a Linux kernel on your machine for WSL 2 to use. Once you have the kernel installed, please run the command again and it should complete successfully without showing the message.

<https://docs.microsoft.com/fr-fr/windows/wsl/wsl2-kernel>

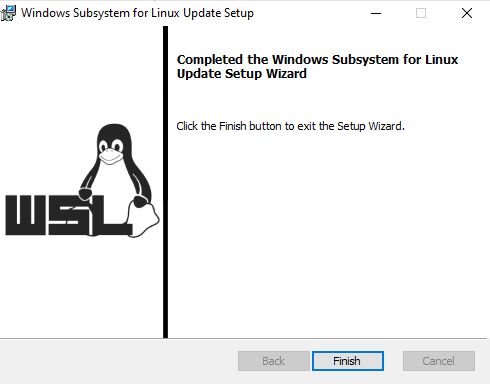
Please [download the latest WSL2 Linux kernel](https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi) update package for x64 machines.

<https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi>





Accept elevation prompt



To install the Linux kernel update package:

1. Run the update package downloaded in the previous step.
2. You will be prompted for elevated permissions, select ‘yes’ to approve this installation.
3. Once the installation is complete, you are ready to begin using WSL2!

<https://devblogs.microsoft.com/commandline/wsl2-will-be-generally-available-in-windows-10-version-2004/>

 We’ve removed the Linux kernel from the Windows OS image and instead will be delivering it to your machine via Windows Update, the same way that 3rd party drivers (like graphics, or touchpad drivers) are installed and updated on your machine today. This change will give you more agility and flexibility over Linux kernel updates in WSL 2. Read on to learn more about how you’ll see this in the user experience.

How will I notice this change?

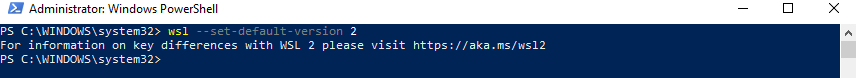
Our end goal is for this change to be seamless, where your Linux kernel is kept up to date without you needing to think about it. By default this will be handled entirely by Windows, just like regular updates on your machine. Inside of the initial release of Windows 10, version 2004, and in the latest Windows Insiders slow ring preview build you will temporarily need to manually install the Linux kernel, and will receive an update in a few months that will add automatic install and servicing capabilities. We made this change now and will have a patch later to ensure that all users in the initial general release of WSL 2 will be serviced via this dynamic model, and no one will be left in a middle state using the prior system.

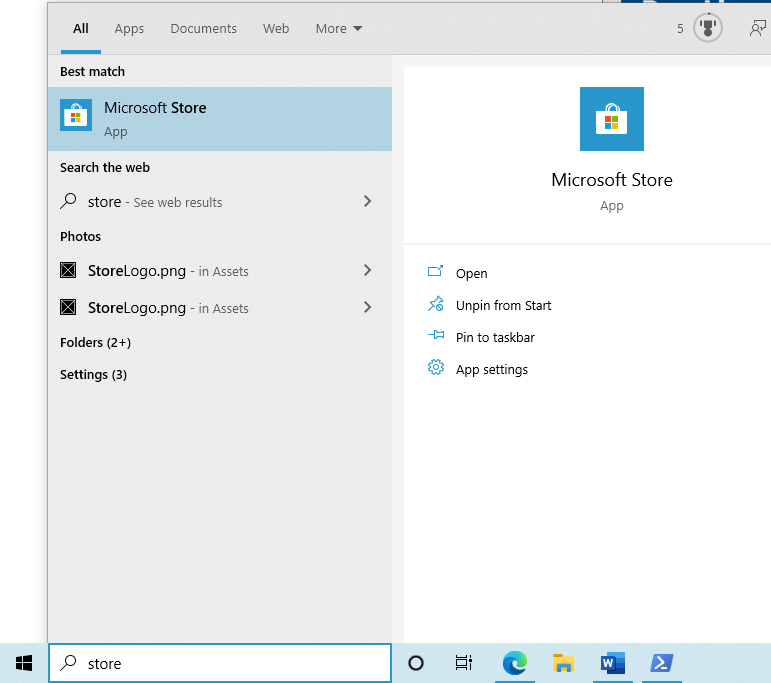
Automatic install and updates

If you’ve ever gone to your Windows settings, and clicked ‘Check for Updates’ you might have seen some other items being updated like Windows Defender malware definitions, or a new touchpad driver, etc. The Linux kernel in WSL 2 will now be serviced in this same method, which means you’ll get the latest kernel version independently of consuming an update to your Windows image. You can manually check for new kernel updates by clicking the ‘Check for Updates’ button, or you can let Windows keep you up to date just like normal.

 We’ve removed the Linux kernel from the Windows OS image and instead will be delivering it to your machine via Windows Update, the same way that 3rd party drivers (like graphics, or touchpad drivers) are installed and updated on your machine today. This change will give you more agility and flexibility over Linux kernel updates in WSL 2. Read on to learn more about how you’ll see this in the user experience.

# Install a Linux distribution

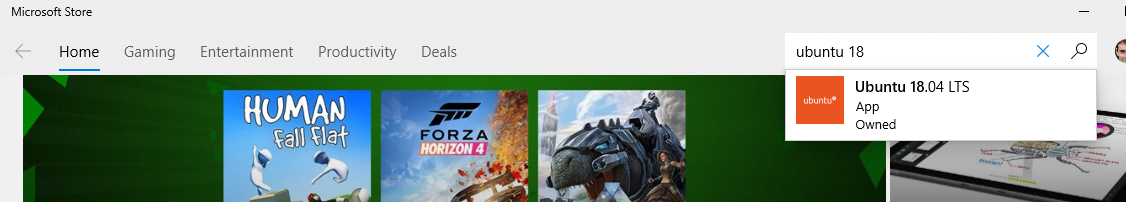


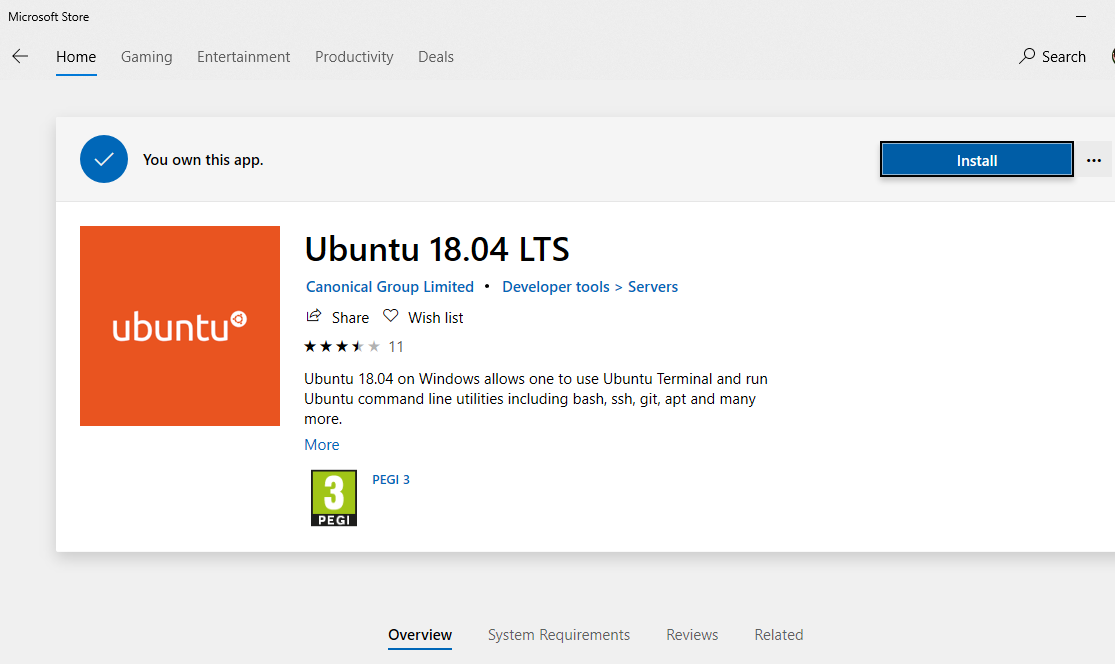


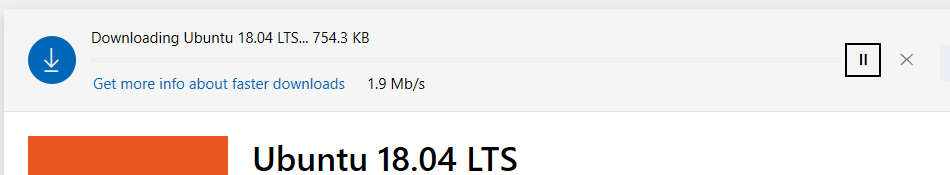
Install your Linux distribution of choice

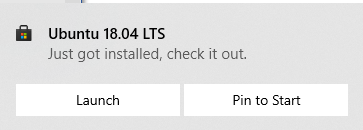
1. Open the [Microsoft Store](https://aka.ms/wslstore) and select your favorite Linux distribution.

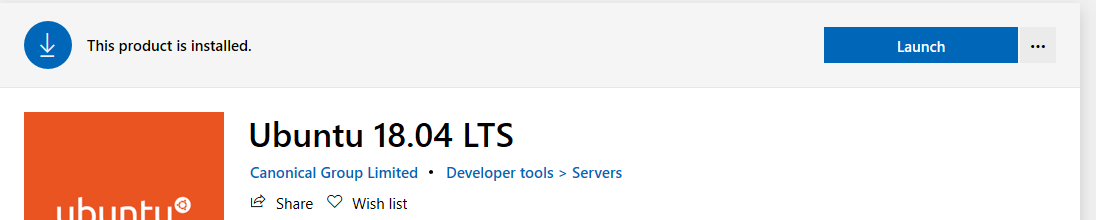
[Ubuntu 18.04 LTS](https://www.microsoft.com/store/apps/9N9TNGVNDL3Q)



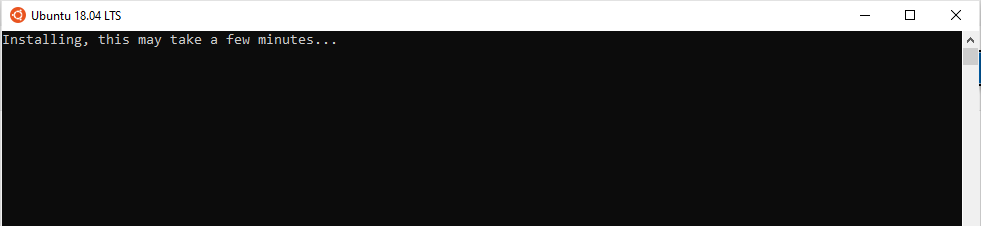


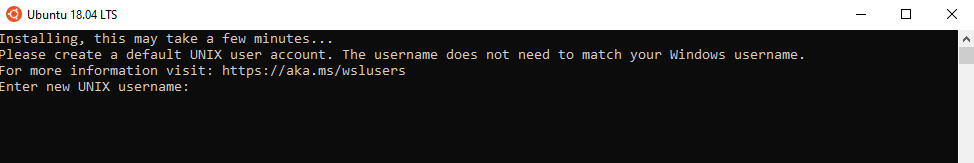


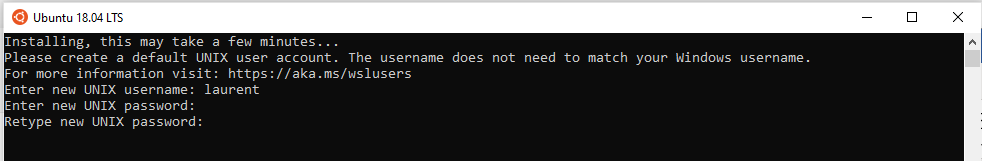


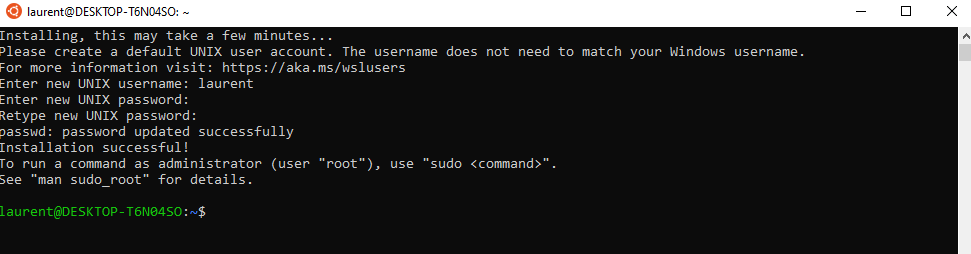


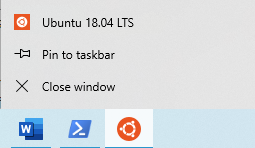
Launch

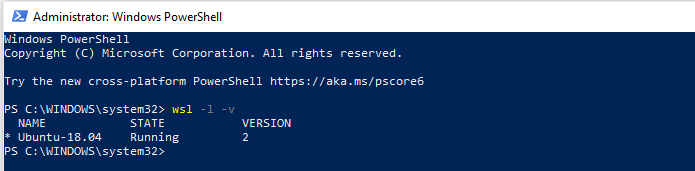








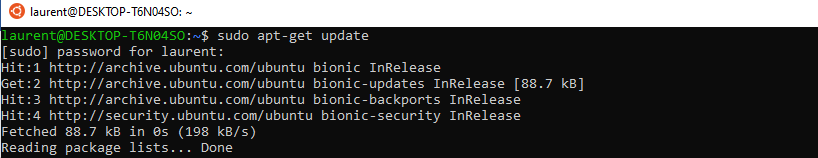




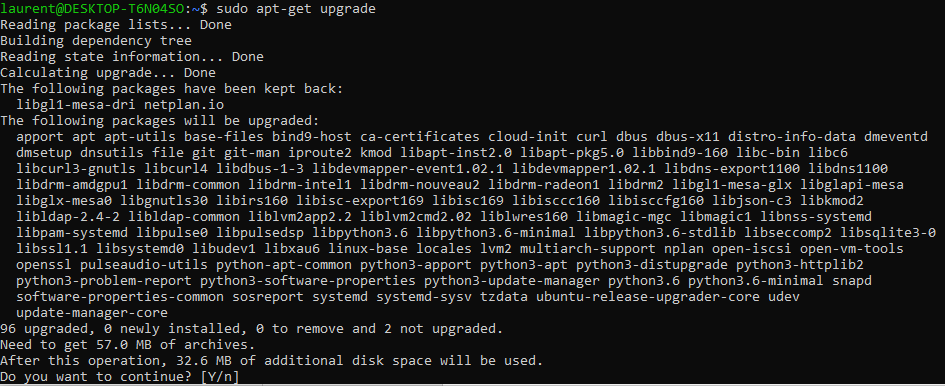
# Update Ubuntu install

1. Fetch **update** software list by running sudo apt-get **update** command.
2. **Update Ubuntu** software by running sudo apt-get **upgrade** command.
3. Reboot the **Ubuntu** box if required by running sudo reboot.

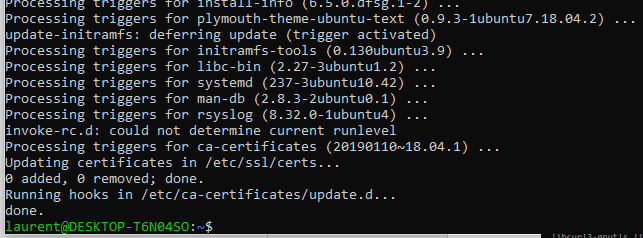
sudo apt-get **update**



sudo apt-get **upgrade**



Y



# Locate the Linux filesystem

C:\Users\laure\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu18.04onWindows\_79rhkp1fndgsc\LocalState

ext4.vhdx : 1.5 Go

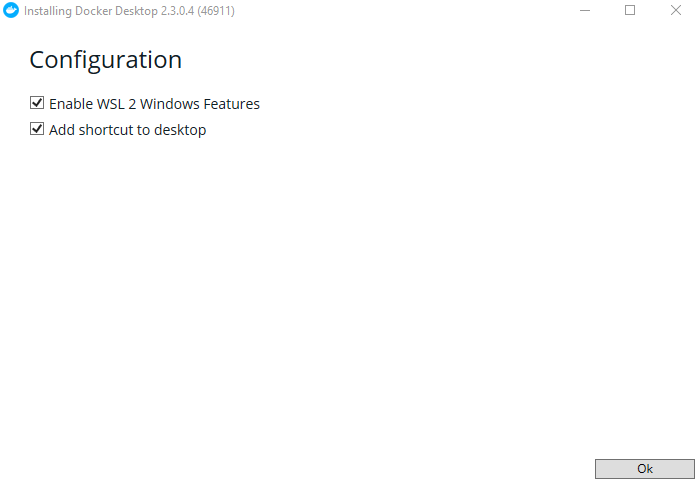
C:\Users\laure\AppData\Local\Temp

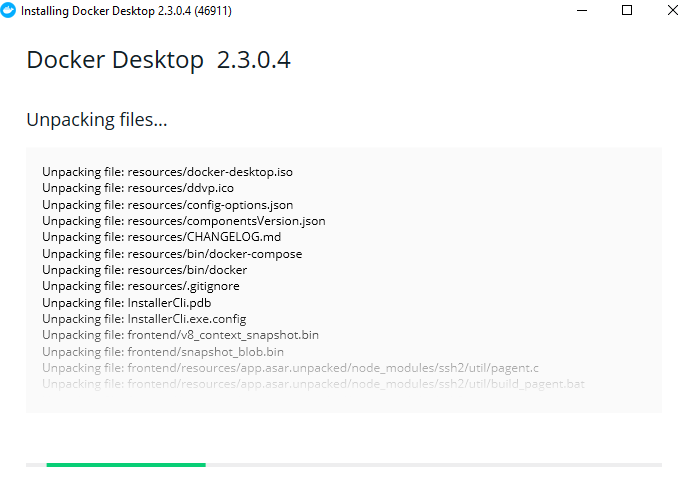
swap.vhdx : 66 Mo (only while VM launched)

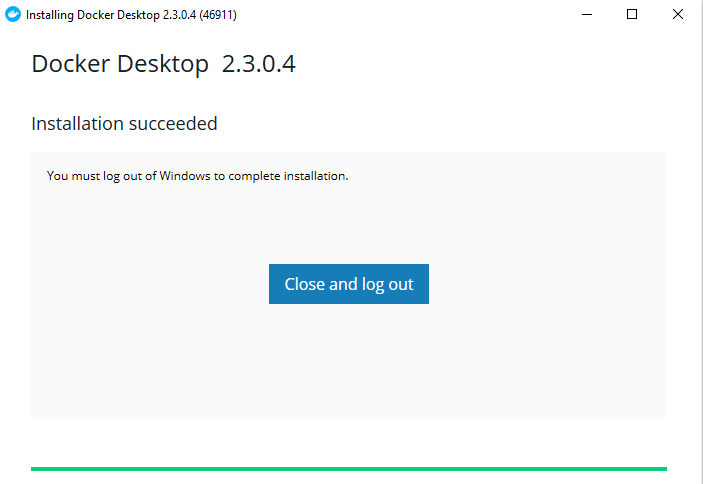
# Install Docker Desktop

Download [Docker Desktop Stable 2.3.0.2](https://hub.docker.com/editions/community/docker-ce-desktop-windows/) or a later release

Accept elevation







Warning it will log you out immediately …

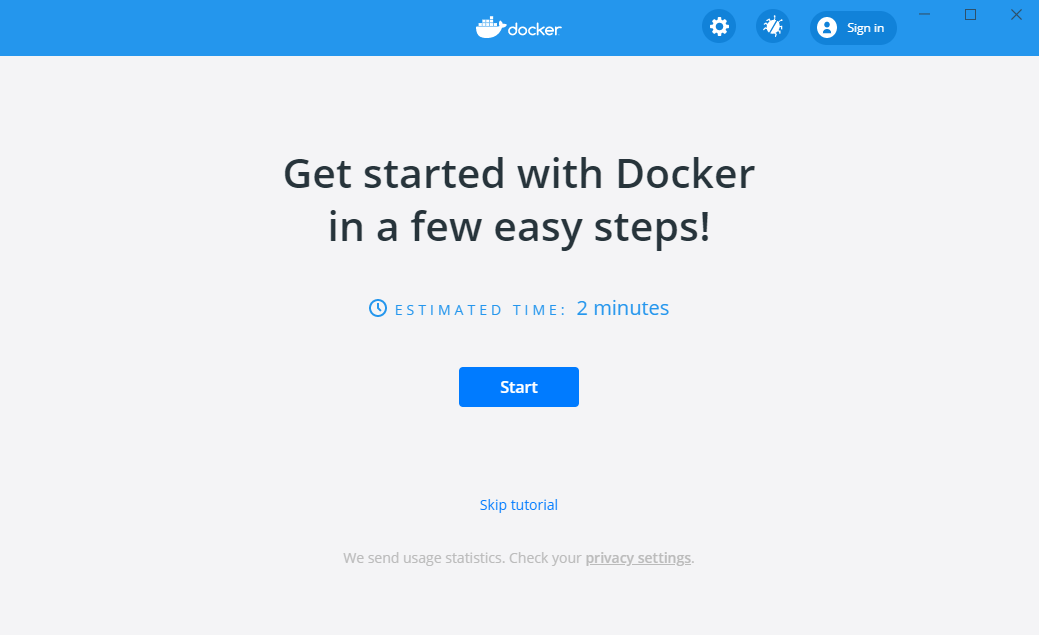
On next logon, wait a couple of minutes

Toast Docker is starting

Then toast Docker is running



The UI starts automatically



Skip tutorial

# Locate Docker Desktop files

Disk space :

Client : C:\Program Files\Docker => 2.2 Go

Wsl and data : C:\Users\laure\AppData\Local\Docker => 1 Go

C:\Users\laure\AppData\Local\Docker\wsl\data

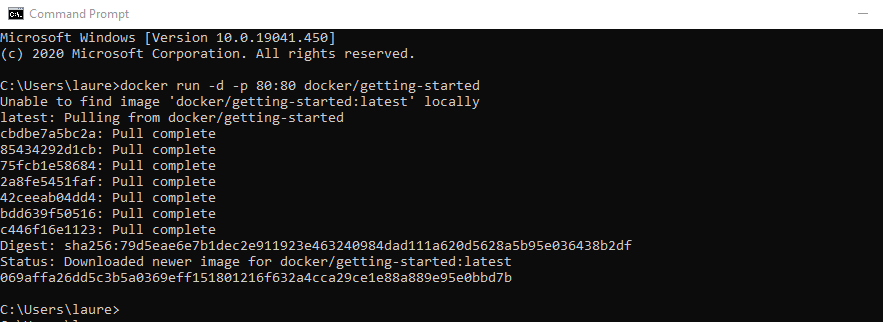
ext4.vhdx : 890 Mo

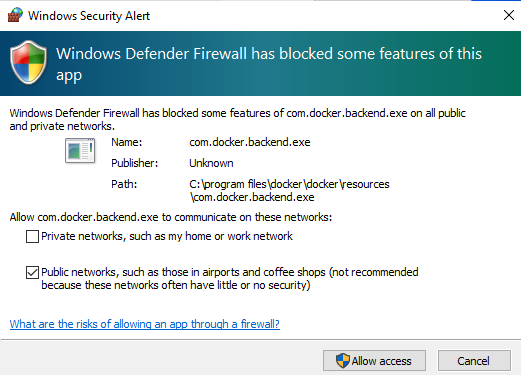
C:\Users\laure\AppData\Local\Docker\wsl\distro

ext4.vhdx : 105 Mo

# Test Docker install on Windows

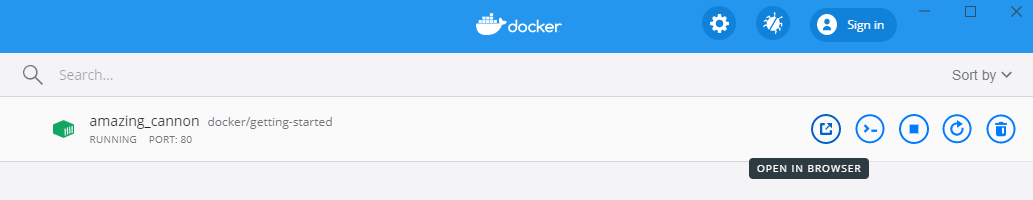
docker run -d -p 80:80 docker/getting-started

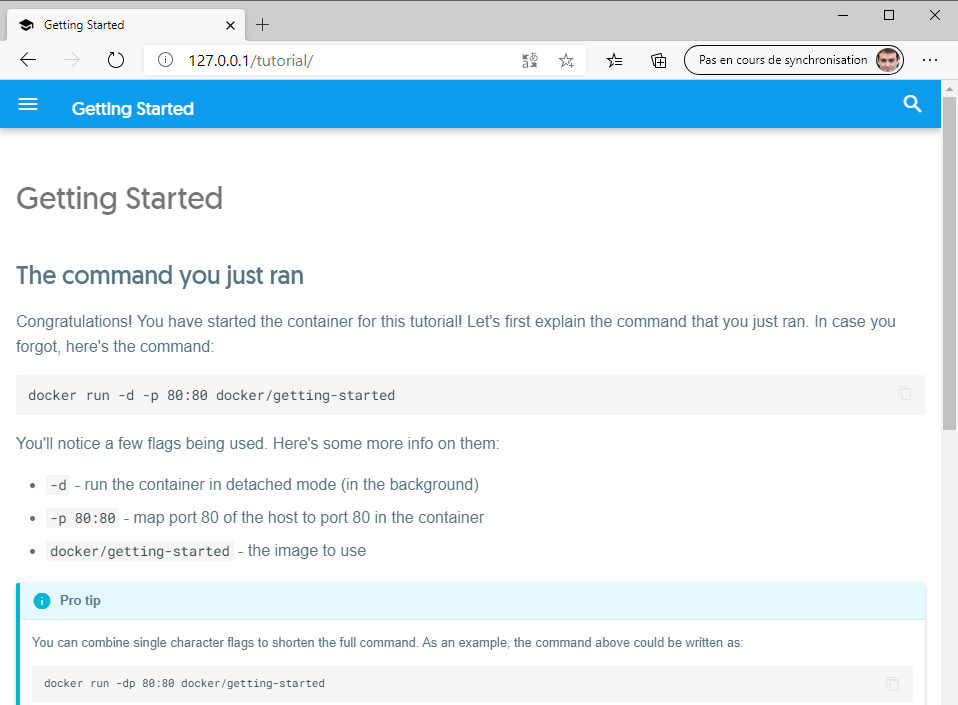




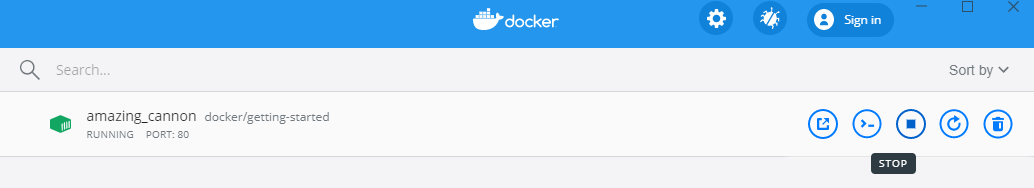
Cancel

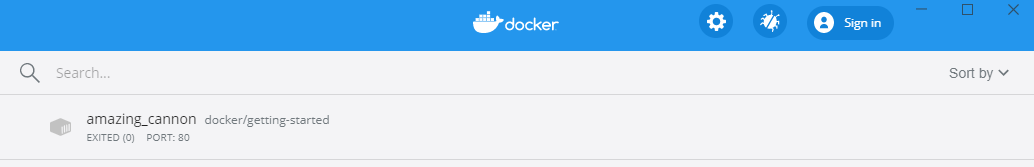
Docker dashboard > Open in Browser



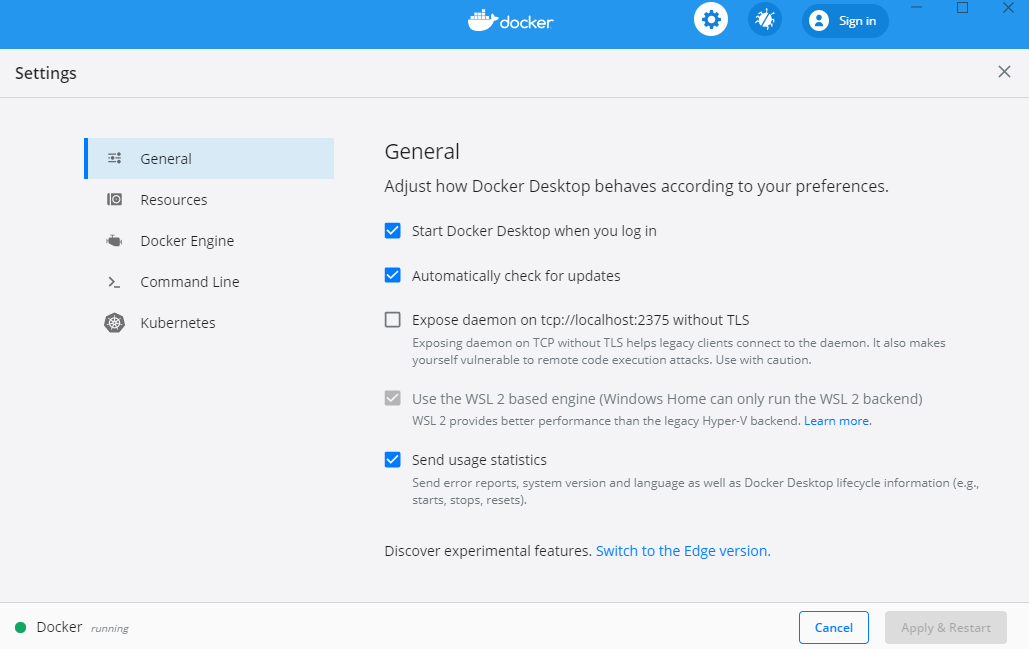


Docker Dashboard > Stop





Docker will start on login by default

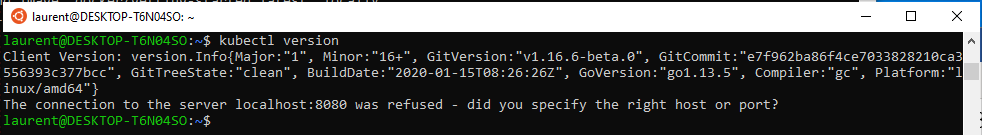


# Test docker install on Linux

Docker v19.03.12 was installed in the Linux filesystem automatically :



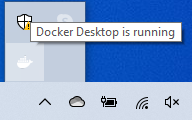
Kubectl v1.16.6 is also present in the filesystem :

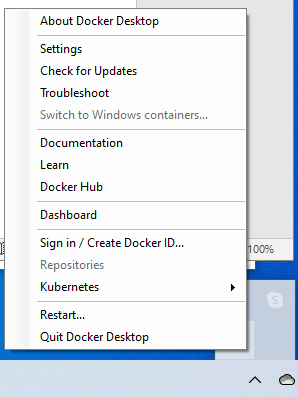


But the Docker Kubernetes is not installed by default

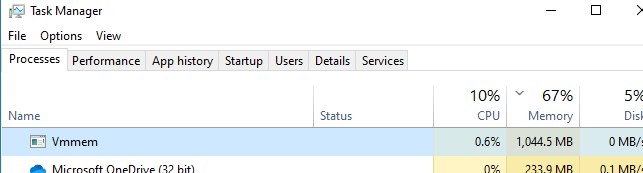
# Start and Stop Docker and WSL memory consumption

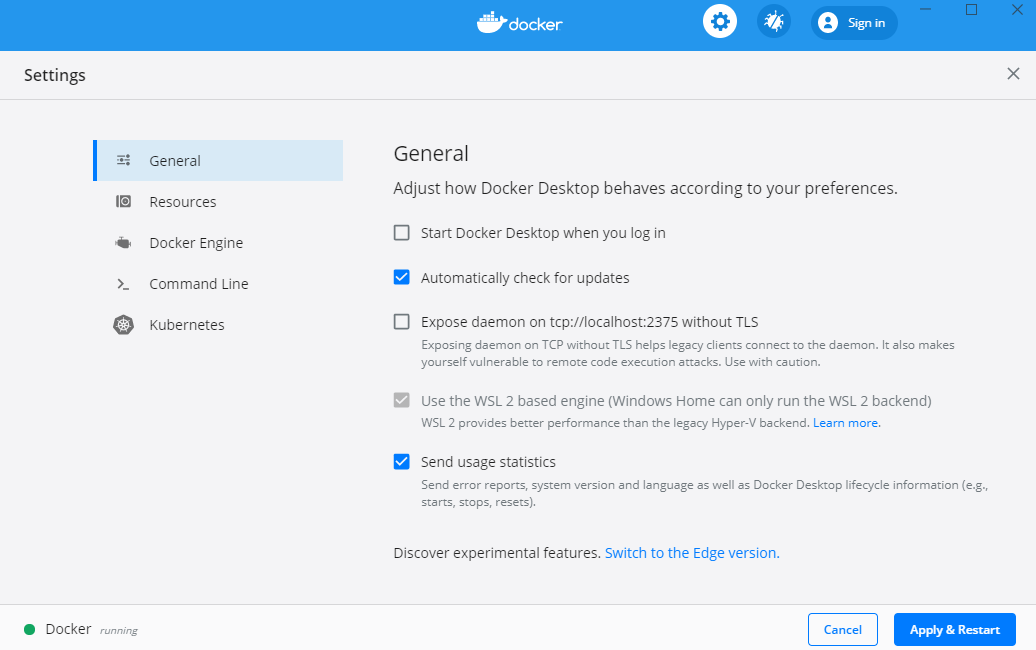
Close the dashboard UI

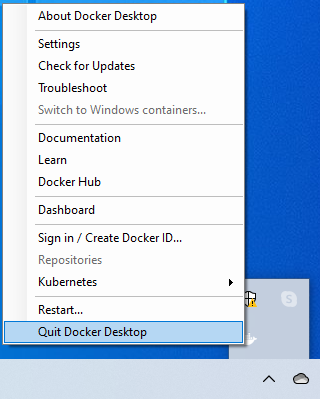




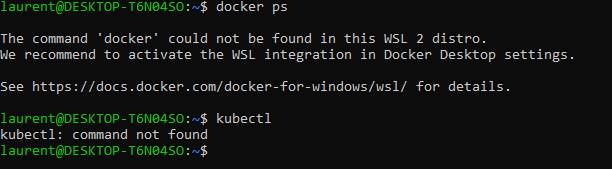
* Relaunch Dashboard



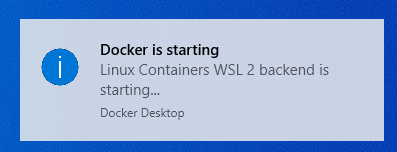


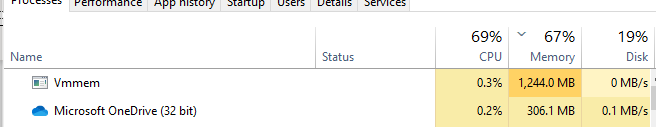




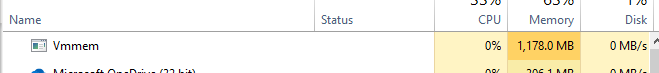




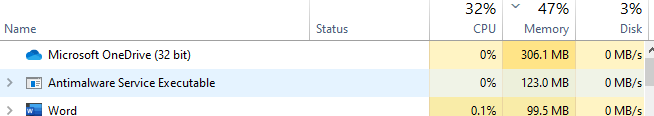




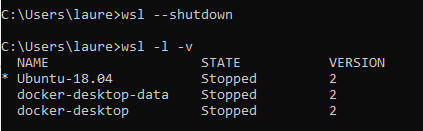
STOP



WAIT exactly one minute



wsl –shutdown => immediate

\*

# WSL 2 Configuration file

<https://docs.microsoft.com/fr-fr/windows/wsl/wsl-config>

You can configure global WSL options by placing a .wslconfig file into the root directory of your users folder: C:\Users\<yourUserName>\.wslconfig. Many of these files are related to WSL 2, please keep in mind you may need to run wsl --shutdown to shut down the WSL 2 VM and then restart your WSL instance for these changes to take effect.

going to c:\users\\*your your profile name\* and creating a .wslconfig file

|  |
| --- |
| [wsl2] |
|  | memory=4GB # Limits VM memory in WSL 2 to 4 GB |
|  | processors=5 # Makes the WSL 2 VM use two virtual processors |
|  |  |



And then, from Powershell with admin rights, restart WSL2 by typing:

Restart-Service LxssManager

**WSL 2 Settings**

Section label: [wsl2]

These settings affect the VM that powers any WSL 2 distribution.

| **PARAMÈTRES WSL 2** | | | |
| --- | --- | --- | --- |
| **key** | **value** | **default** | **notes** |
| kernel | string | The Microsoft built kernel provided inbox | An absolute Windows path to a custom Linux kernel. |
| memory | size | 80% of your total memory on Windows \* | How much memory to assign to the WSL 2 VM. |
| processors | number | The same number of processors on Windows | How many processors to assign to the WSL 2 VM. |
| localhostForwarding | boolean | true | Boolean specifying if ports bound to wildcard or localhost in the WSL 2 VM should be connectable from the host via localhost:port. |
| kernelCommandLine | string | Blank | Additional kernel command line arguments. |
| swap | size | 25% of memory size on Windows rounded up to the nearest GB | How much swap space to add to the WSL 2 VM, 0 for no swap file. |
| swapFile | string | %USERPROFILE%\AppData\Local\Temp\swap.vhdx | An absolute Windows path to the swap virtual hard disk. |

* Note: This value is true for Windows Build 19041 and may be different in Windows builds in the Insiders program

Entries with the path value must be Windows paths with escaped backslashes, e.g: C:\\Temp\\myCustomKernel

Entries with the size value must be a size followed by a unit, for example 8GB or 512MB.

All WSL 2 distros are run on the same virtual machine (VM), and therefore any configuration options to this VM will be applied globally to all WSL 2 distros. In this new update we’ve added the ability to start using global config options for WSL. These options are targeted towards power users who want to further customize their WSL experience.

To get started, create a new file called .wslconfig in your Users folder ( C:\Users\<yourUsername>\ where <yourUsername> is your Windows login name). The .wslconfig file is modelled after an [INI file](https://en.wikipedia.org/wiki/INI_file), just like .gitconfig.

# WLS 2 memory management

<https://devblogs.microsoft.com/commandline/memory-reclaim-in-the-windows-subsystem-for-linux-2/>

---

# Docker Desktop – Deploy local registry

<https://docs.docker.com/registry/deploying/>

docker run -d -p 5000:5000 --restart=always --name registry -v /mnt/registry:/var/lib/registry registry:2

# Rancher k3d – Create a local k3s cluster in containers

<https://k3d.io/>

laurent@XPS8700:~$ k3d help cluster create

Create a new k3s cluster with containerized nodes (k3s in docker).

Every cluster will consist of one or more containers:

- 1 (or more) server node container (k3s)

- (optionally) 1 loadbalancer container as the entrypoint to the cluster (nginx)

- (optionally) 1 (or more) agent node containers (k3s)

Usage:

k3d cluster create NAME [flags]

Flags:

-a, --agents int Specify how many agents you want to create

--api-port [HOST:]HOSTPORT Specify the Kubernetes API server port exposed on the LoadBalancer (Format: [HOST:]HOSTPORT)

- Example: `k3d cluster create --servers 3 --api-port 0.0.0.0:6550` (default "random")

-h, --help help for create

-i, --image string Specify k3s image that you want to use for the nodes (default "docker.io/rancher/k3s:v1.18.6-k3s1")

--k3s-agent-arg k3s agent Additional args passed to the k3s agent command on agent nodes (new flag per arg)

--k3s-server-arg k3s server Additional args passed to the k3s server command on server nodes (new flag per arg)

--network string Join an existing network

--no-image-volume Disable the creation of a volume for importing images

--no-lb Disable the creation of a LoadBalancer in front of the server nodes

-p, --port [HOST:][HOSTPORT:]CONTAINERPORT[/PROTOCOL][@NODEFILTER] Map ports from the node containers to the host (Format: [HOST:][HOSTPORT:]CONTAINERPORT[/PROTOCOL][@NODEFILTER])

- Example: `k3d cluster create --agents 2 -p 8080:80@agent[0] -p 8081@agent[1]`

-s, --servers int Specify how many servers you want to create (default 1)

--switch-context Directly switch the default kubeconfig's current-context to the new cluster's context (requires --update-default-kubeconfig) (default true)

--timeout duration Rollback changes if cluster couldn't be created in specified duration.

--token string Specify a cluster token. By default, we generate one.

--update-default-kubeconfig Directly update the default kubeconfig with the new cluster's context (default true)

-v, --volume [SOURCE:]DEST[@NODEFILTER[;NODEFILTER...]] Mount volumes into the nodes (Format: [SOURCE:]DEST[@NODEFILTER[;NODEFILTER...]]

- Example: `k3d cluster create --agents 2 -v /my/path@agent[0,1] -v /tmp/test:/tmp/other@server[0]`

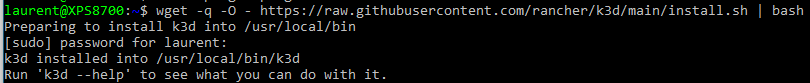
--wait Wait for the server(s) to be ready before returning. Use '--timeout DURATION' to not wait forever. (default true)

Global Flags:

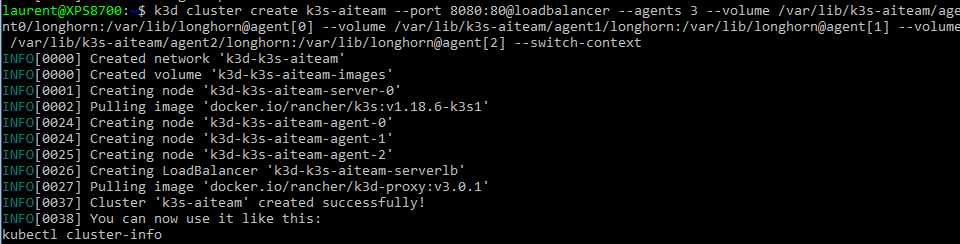
--verbose Enable verbose output (debug logging)

1. Start Docker Desktop
2. When started, open Unbuntu bash

wget -q -O - https://raw.githubusercontent.com/rancher/k3d/main/install.sh | bash



k3d cluster create k3s-aiteam --port 8080:80@loadbalancer --agents 3 --volume /var/lib/rancher/k3s/storage:/var/lib/rancher/k3s/storage --switch-context



k3d cluster start k3s-aiteam

# Set up k3s local-path provisioner

<https://rancher.com/docs/k3s/latest/en/storage/>

<https://github.com/rancher/local-path-provisioner/blob/master/README.md#usage>

One file with pvc name is created in /var/lib/rancher/k3s/storage in node local storage and grows as needed !

With k3d, you can map this path inside the container to a path in your linux distrib, per node :

**--volume /var/lib/cogfactory/storage/agent0:/var/lib/rancher/k3s/storage@agent[0]**

Here is a PVC :

apiVersion: v1--volume /var/lib/cogfactory/storage/agent0:/var/lib/rancher/k3s/storage@agent[0]e

kind: PersistentVolumeClaim

metadata:

name: test-pvc

namespace: default

spec:

accessModes:

- ReadWriteOnce

storageClassName: local-path

resources:

requests:

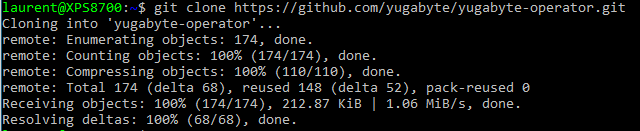
storage: 1Gi

* See “Delete k3d cluster – Delete persistent storage” to manually delete the local storage after the cluster is deleted

# Deploy YugabyteDB with operator

<https://docs.yugabyte.com/latest/deploy/kubernetes/single-zone/oss/yugabyte-operator/>

git clone <https://github.com/yugabyte/yugabyte-operator.git>

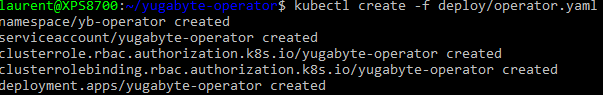


cd yugabyte-operator/

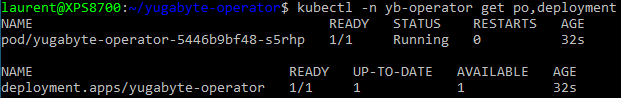
kubectl create -f deploy/crds/yugabyte\_v1alpha1\_ybcluster\_crd.yaml



kubectl create -f deploy/operator.yaml



kubectl -n yb-operator get po,deployment



cp deploy/crds/yugabyte\_v1alpha1\_ybcluster\_full\_cr.yaml deploy/crds/yugabyte\_v1alpha1\_ybcluster\_aiteam.yaml

vi deploy/crds/yugabyte\_v1alpha1\_ybcluster\_aiteam.yaml

* metadata:
* name: yugabyte-cluster
* namespace: yugabytedb
* Select latest image version : 2.2.2.0-b15 (<https://hub.docker.com/r/yugabytedb/yugabyte/tags>)
* Disable tls
* Pour master et server :
* Storageclass: local-path
* Use node to node encryption : false

# Set up YugabyteDB authentication

Optional – authentication (gflags section) :

<https://docs.yugabyte.com/latest/secure/authentication/ysql-authentication/>

<https://docs.yugabyte.com/latest/deploy/kubernetes/single-zone/oss/yugabyte-operator/#flags>

<https://docs.yugabyte.com/latest/reference/configuration/yb-tserver/#ysql-enable-auth>

<https://docs.yugabyte.com/latest/reference/configuration/yb-tserver/#use-cassandra-authentication>

# Set up YugabyteDB ports

<https://docs.yugabyte.com/latest/deploy/kubernetes/single-zone/oss/yugabyte-operator/>

UIPortMaster 7000

RPCPortMaster 7100

tserverUIPort NA

tserverRPCPort 9100

ycqlPort 9042

yedisPort 6379

ysqlPort 5433

# Deploy YugabyteDB – detailed procedure

kubectl create namespace yugabytedb

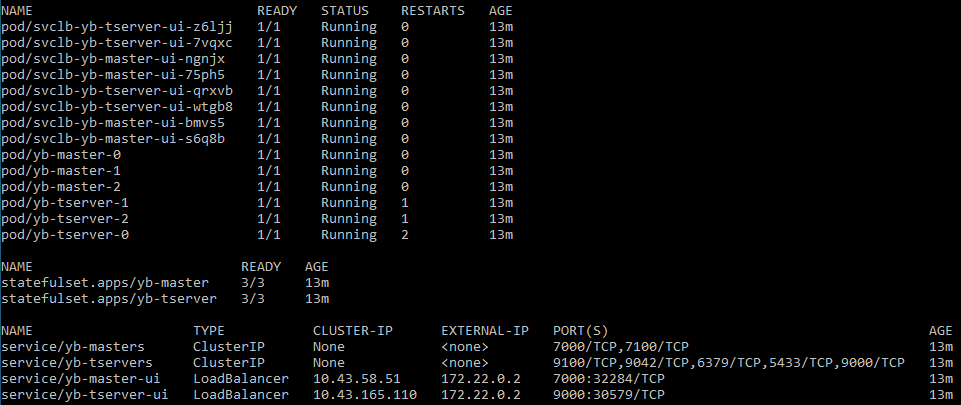


kubectl create -f deploy/crds/yugabyte\_v1alpha1\_ybcluster\_aiteam.yaml

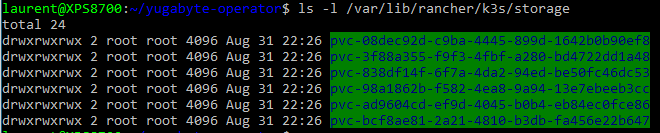


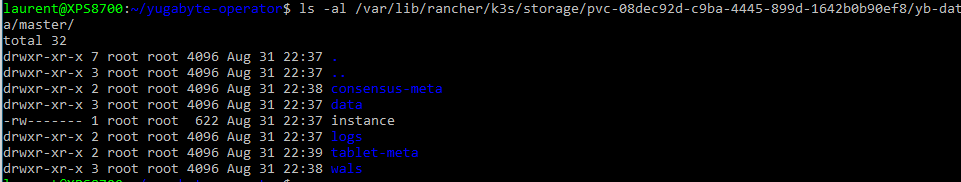
Warning : pulling image takes a while … 612.11 MB (10 min)

kubectl get pvc,po,sts,svc -n yugabytedb



kubectl get pods -n yugabytedb -w





Create additional LoadBalancer service :

cd..

vi sql-lb.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: yb-tserver

name: yb-tserver-sql

namespace: yugabytedb

spec:

selector:

app: yb-tserver

ports:

- name: sql

port: 5433

targetPort: 5433

type: LoadBalancer

kubectl apply -f sql-lb.yaml

kubectl get svc -n yugabytedb

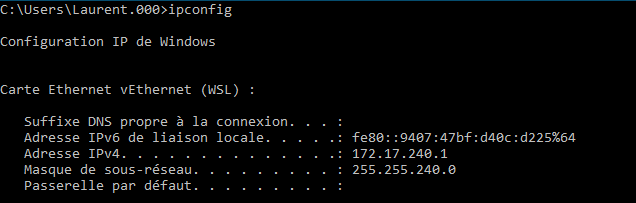
Yugabyte dashboard :

<http://localhost:7000/>



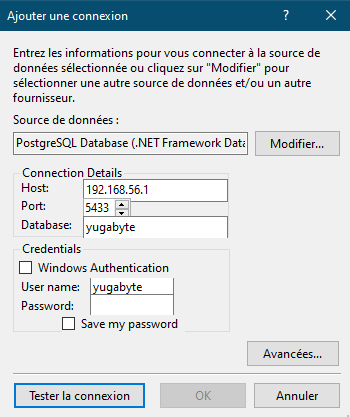
# Get machine and WSL IP addresses

Get wsl IP address : 172.17.240.1



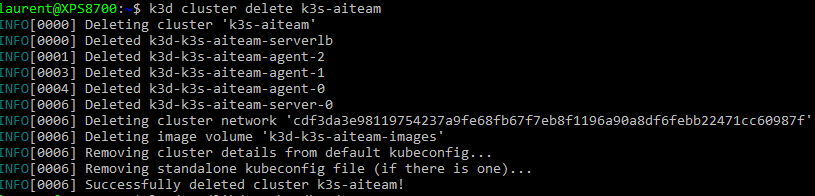
Get machine IP address

192.168.56.1



# Delete k3d cluster

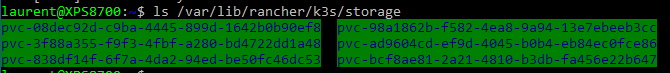
k3d cluster delete k3s-aiteam



# Delete k3d cluster – Delete persistent storage

Persistent storage still there … (new pvcs recreated when cluster recreated => must delete manually)

ls /var/lib/rancher/k3s/storage



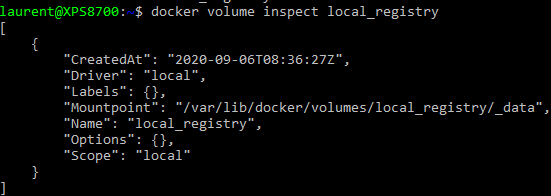
sudo rm -rf /var/lib/rancher/k3s/storage/pvc\*

ls /var/lib/rancher/k3s/storage

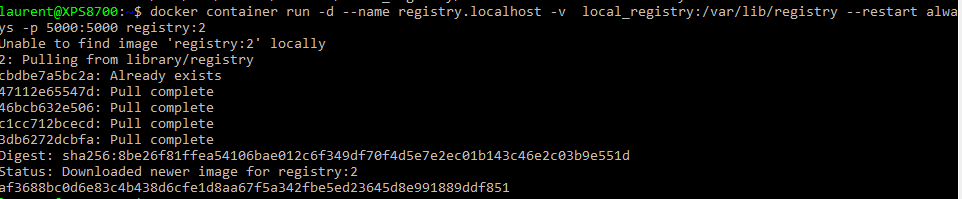
# Docker – Set up a local registry

<https://k3d.io/usage/guides/registries/>

docker volume create local\_registry



docker container run -d --name registry.localhost -v local\_registry:/var/lib/registry --restart always -p 5000:5000 registry:2



Launch notepad as an admin :

Open file : C:\Windows\System32\drivers\etc\

All files …

Open : Hosts

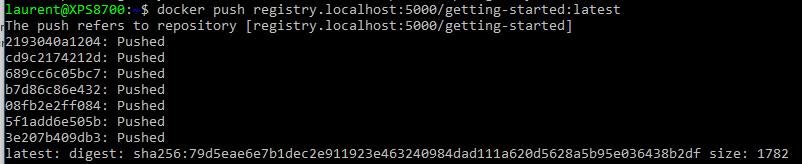
# Local registry

127.0.0.1 registry.localhost

>Test

docker tag docker/getting-started:latest registry.localhost:5000/getting-started:latest

docker push registry.localhost:5000/getting-started:latest



# k3d cluster – Configure local registry

sudo mkdir -p /etc/rancher/k3s

sudo vi /etc/rancher/k3s/registries.yaml

[warning – 2 spaces]

mirrors:

"registry.localhost:5000":

endpoint:

- http://registry.localhost:5000

docker network connect k3d-k3s-aiteam registry.localhost

# Create k3d cluster – Yugabyte & Postgresql versions

k3d cluster delete k3s-aiteam

sudo rm -rf /var/lib/rancher/k3s/storage/\*

* Yugabyte

k3d cluster create k3s-aiteam --port 8080:80@loadbalancer --port 7000:7000@loadbalancer --port 5433:5433@loadbalancer --agents 3 --volume /var/lib/rancher/k3s/storage:/var/lib/rancher/k3s/storage --volume /etc/rancher/k3s/registries.yaml:/etc/rancher/k3s/registries.yaml --switch-context

* Postgresql

k3d cluster create k3s-aiteam --port 8080:80@loadbalancer --port 5432:5432@loadbalancer --port 5050:5050@loadbalancer --agents 3 --volume /var/lib/rancher/k3s/storage:/var/lib/rancher/k3s/storage --volume /etc/rancher/k3s/registries.yaml:/etc/rancher/k3s/registries.yaml --switch-context

# k3d cluster - Test local registry + ingress

k3d cluster start k3s-aiteam

Inside Docker Dashboard, log into server node, test :

* curl http://registry.localhost:5000/v2/getting-started/manifests/latest

kubectl create deployment getting-started --image=registry.localhost:5000/getting-started:latest



kubectl create service clusterip getting-started --tcp=80:80

vi test-ing.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: getting-started

annotations:

ingress.kubernetes.io/ssl-redirect: "false"

spec:

rules:

- http:

paths:

- path: /tutorial

backend:

serviceName: getting-started

servicePort: 80

- path: /assets

backend:

serviceName: getting-started

servicePort: 80

kubectl apply -f test-ing.yaml



Dans un navigateur :

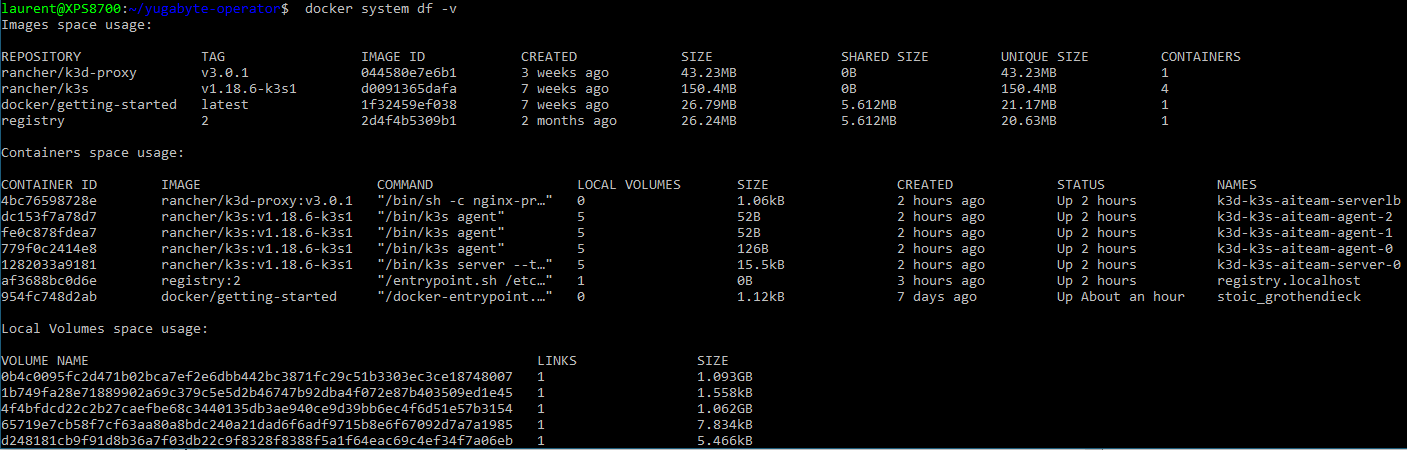
<http://localhost/tutorial/>

kubectl delete -f test-ing.yaml

kubectl delete deployment getting-started

# Docker - Size of images, containers, volumes

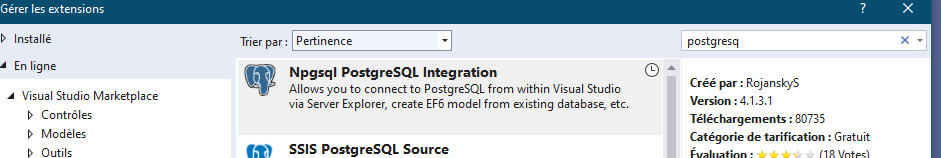
docker system df -v



# Login to yugabyteDB from Visual Studio

Extensions / manage extensions

* install “Npgsql PostgreSQL Integration”

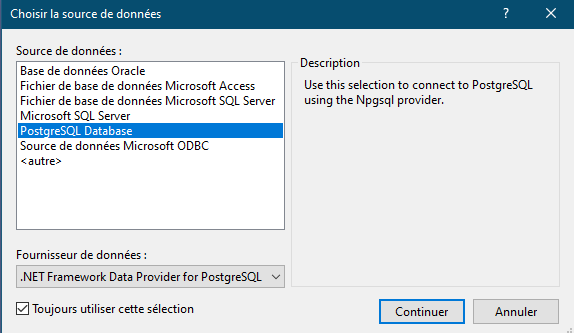


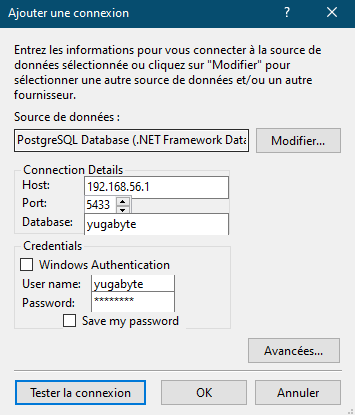
Close Visual Studio

Accept Modify

Restart Visual Studio

Tools / Connect to Database

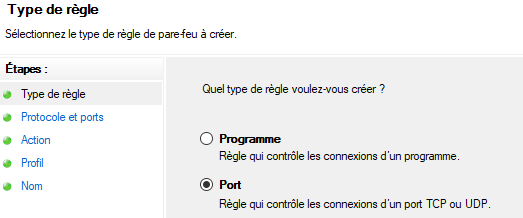


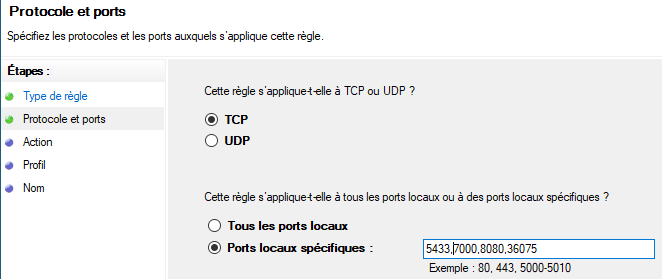


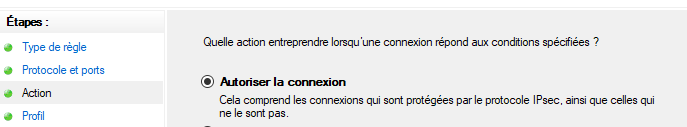
(default password yugabyte)

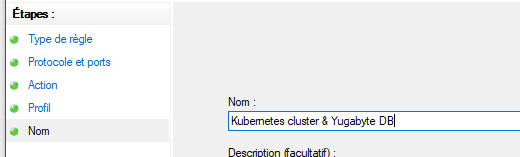
# Configure remote kubectl access

Open Windows firewall on server PC









On the client PC

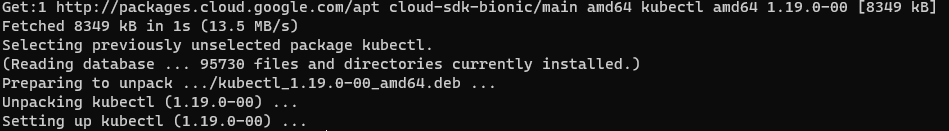
sudo apt-get update && sudo apt-get install -y apt-transport-https

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubectl



vi .kube/config

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https:// 192.168.1.14:36075

name: k3d-k3s-aiteam

contexts:

- context:

cluster: k3d-k3s-aiteam

user: admin@k3d-k3s-aiteam

name: k3d-k3s-aiteam

current-context: k3d-k3s-aiteam

kind: Config

preferences: {}

users:

- name: admin@k3d-k3s-aiteam

user:

password: 6eaddc27ea5852767c5d16489317b0dd

username: admin

# Install Postgresql with Crunchy Operator

<DOC>

<https://access.crunchydata.com/documentation/postgres-operator/latest/>

## pgo create cluster hacluster --replica-count=1 --metrics --pgbackrest-storage-type="local,s3" --pgbouncer --pgbadger

<https://access.crunchydata.com/documentation/postgres-operator/4.4.1/architecture/pgadmin4/>

## pgo create pgadmin hippo

## kubectl port-forward svc/hippo-pgadmin 5050:5050

http://localhost:5050

Ports : <https://access.crunchydata.com/documentation/postgres-operator/4.4.1/installation/prerequisites/>

<https://access.crunchydata.com/documentation/postgres-operator/4.4.1/installation/postgres-operator/>

<START HERE>

curl <https://raw.githubusercontent.com/CrunchyData/postgres-operator/v4.4.1/installers/kubectl/postgres-operator.yml> > postgres-operator.yml

vi postgres-operator.yml

* db\_name: "aiteam"
* grafana\_install: "true"
* grafana\_storage\_class\_name: "local-path"
* namespace\_mode: "disabled"
* pgo\_admin\_password: "lizard77"
* pgo\_installation\_name: "devtest"
* prometheus\_install: "true"
* prometheus\_storage\_class\_name: "local-path"
* service\_type: "LoadBalancer"
* backrest\_storage: "k3s-local-path"
* backup\_storage: "k3s-local-path"
* primary\_storage: "k3s-local-path"
* replica\_storage: "k3s-local-path"
* storage5\_name: "k3s-local-path"
* storage5\_access\_mode: "ReadWriteOnce"
* storage5\_size: "1G"
* storage5\_type: "dynamic"
* storage5\_class: "local-path"

kubectl create namespace pgo

kubectl apply -f postgres-operator.yml

kubectl -n pgo get pods

kubectl -n pgo describe pod postgres-operator-5fbcccf9b5-5pnrn

* Wait for all pods : Running
* Error for pgo-deploy is normal when operator deployment is finished

curl https://raw.githubusercontent.com/CrunchyData/postgres-operator/v4.4.1/installers/kubectl/client-setup.sh > client-setup.sh

NB : need to execute client-setup each time you reinstall the operator (cert generation)

chmod +x client-setup.sh

./client-setup.sh

vi ~/.bashrc

export PATH=/home/laurent/.pgo/pgo:$PATH

export PGOUSER=/home/laurent/.pgo/pgo/pgouser

export PGO\_CA\_CERT=/home/laurent/.pgo/pgo/client.crt

export PGO\_CLIENT\_CERT=/home/laurent/.pgo/pgo/client.crt

export PGO\_CLIENT\_KEY=/home/laurent/.pgo/pgo/client.key

export PGO\_APISERVER\_URL='https://127.0.0.1:8443'

export PGO\_NAMESPACE=pgo

source ~/.bashrc

kubectl -n pgo get deployments

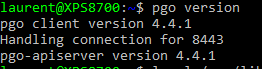


kubectl -n pgo get pods



kubectl -n pgo port-forward svc/postgres-operator 8443:8443 &

pgo version



pgo create cluster aiteam --replica-count 1 --metrics --show-system-accounts

database name: aiteam

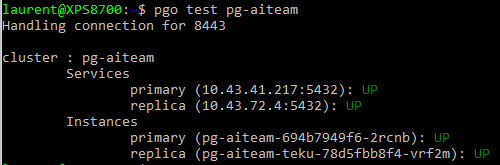
users:

username: postgres password: c7KI0c5OpP2omk1}=dmP02wq

username: primaryuser password: ,t>)?g,9kDWRhy|oj02;[Y}{

username: testuser password: Y9q\*,:mG\0<mLw\_x+bv?EGxe

pgo test aiteam



pgo create pgadmin pg-aiteam

kubectl -n pgo get service aiteam-pgadmin -o yaml > pgadmin-service.yaml

vi pgadmin-service.yaml

* type: LoadBalancer

kubectl apply -f pgadmin-service.yaml

## pgo show user pg-aiteam

pgo create user --username=laurent aiteam --managed

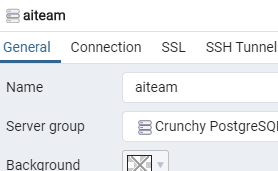
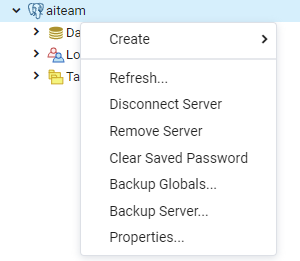
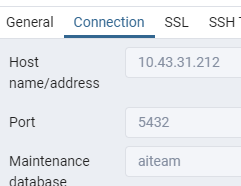
pgo update user --username=laurent aiteam --password=lizard77

## pgo delete user --username=testuser pg-aiteam

pgo show user aiteam

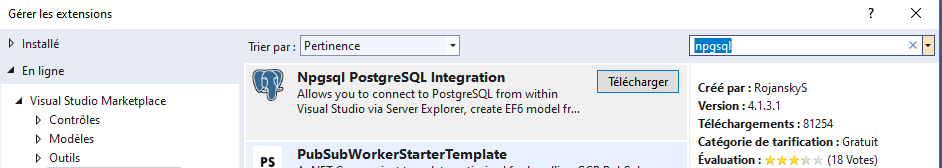
<http://localhost:5050>

(NOT USEFUL IF CLUSTER NAME MATCHES DATABASE NAME) change pg-aiteam database name to aiteam (right-click => Properties)

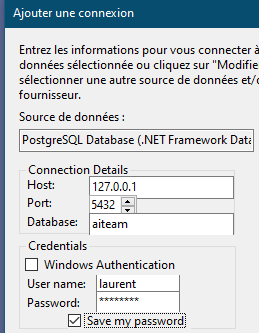
# Postgresql - Connect from Visual Studio

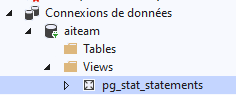
* Extensions



Close / Modify / reopen Visual Studio

* Server Explorer / Add database connexion

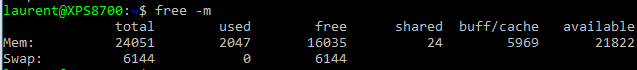




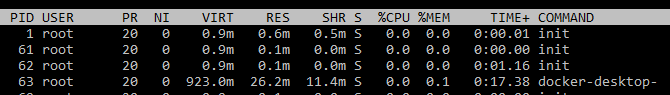
# Study of memory used

<https://www.howtogeek.com/659529/how-to-check-memory-usage-from-the-linux-terminal/>

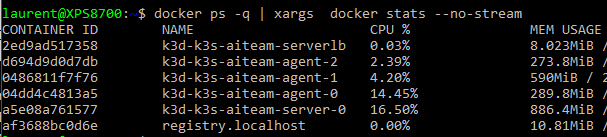
free -m



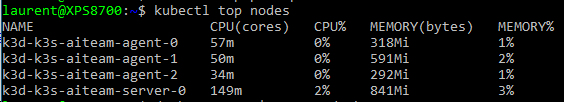
top (then press key e to switch to mb)



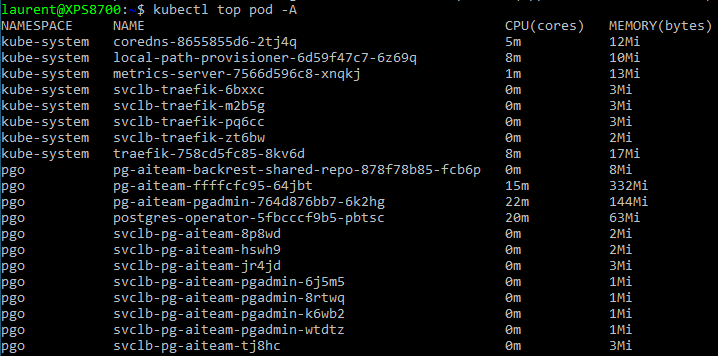
docker ps -q | xargs docker stats --no-stream



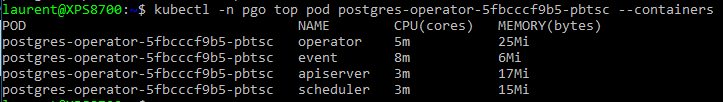
kubectl top nodes



kubectl top pod -A



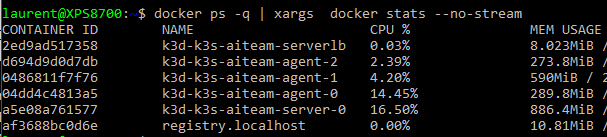
kubectl top pod postgres-operator-5fbcccf9b5-pbtsc --containers -n pgo



kubectl top pod postgres-operator-5fbcccf9b5-pbtsc --containers -n pgo

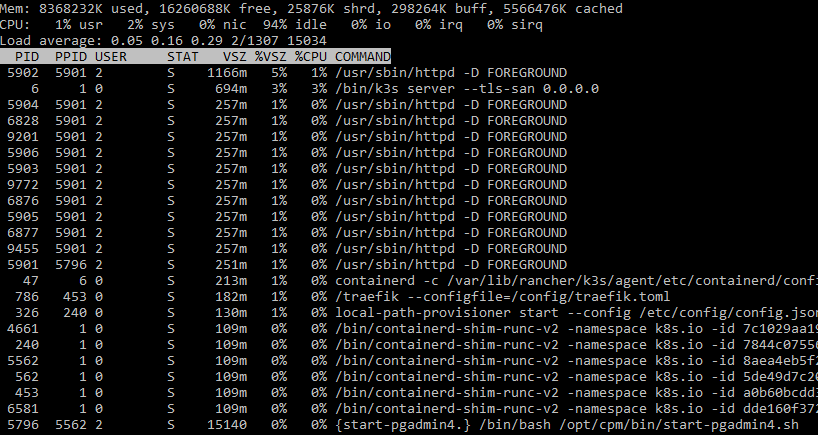


(copy of command above)



docker exec -it a5e08a761577 /bin/sh

top (then, press m to sort by memory)



# Set up SSH connection to WSL

This is pretty straightforward in Ubuntu.

# Remove the default install  
sudo apt-get remove --purge openssh-server# Update  
sudo apt-get update  
sudo apt-get upgrade -y# Install OpenSSH Server  
sudo apt-get install -y openssh-server

Set the following in ssh config:

# Configure OpenSSH  
sudo vi /etc/ssh/sshd\_config

Port 222  
#X11Forwarding yes  
#X11DisplayOffset 10

Finally, start the OpenSSH server.

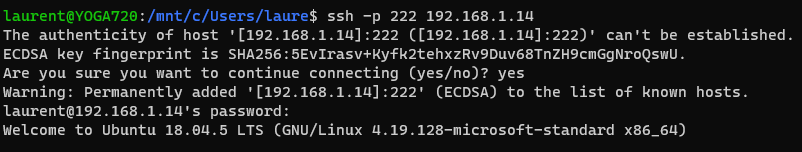
sudo service ssh start

Windows

netsh interface portproxy add v4tov4 listenport=222 listenaddress=0.0.0.0 connectport=222 connectaddress=172.25.51.41

On the target machine :

ssh -p 222 192.168.1.14



# Diagnose Windows firewall problem

<https://www.howtogeek.com/220204/how-to-track-firewall-activity-with-the-windows-firewall-log/>

<https://superuser.com/questions/1130078/how-to-tell-which-windows-firewall-rule-is-blocking-traffic>

date time action protocol src-ip dst-ip src-port dst-port size tcpflags t cpsyn tcpack tcpwin icmptype icmpcode info path

2020-09-13 11:38:10 DROP TCP 192.168.1.15 192.168.1.14 49170 5050 52 S 2836977185 0 64240 - - - RECEIVE

Powershell as Admin :

New-NetFirewallRule -Name Allow192.168.1.15 -DisplayName 'Allow from 192.168.1.15' -Enabled True -Direction Inbound -Protocol ANY -Action Allow -Profile ANY -RemoteAddress 192.168.1.15

<https://stackoverflow.com/questions/61002681/connecting-to-wsl2-server-via-local-network>

netsh interface portproxy add v4tov4 listenport=5050 listenaddress=0.0.0.0 connectport=5050 connectaddress=172.25.51.41

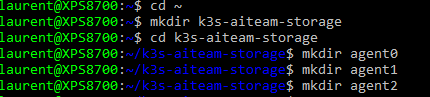
netsh interface portproxy add v4tov4 listenport=5432 listenaddress=0.0.0.0 connectport=5432 connectaddress=172.25.51.41

netsh interface portproxy add v4tov4 listenport=8080 listenaddress=0.0.0.0 connectport=8080 connectaddress=172.25.51.41

# Install Rancher Longhorn => KO

<https://rancher.com/docs/k3s/latest/en/storage/>

<https://longhorn.io/docs/1.0.2/deploy/install/install-with-kubectl/>



mkdir -p ~/k3s-aiteam-storage/agent0/var/run

mkdir -p ~/k3s-aiteam-storage/agent1/var/run

mkdir -p ~/k3s-aiteam-storage/agent2/var/run

k3d cluster create k3s-aiteam --port 8080:80@loadbalancer --agents 3 --volume ~/k3s-aiteam-storage/agent0/var/run:/var/run@agent[0] --volume ~/k3s-aiteam-storage/agent1/var/run:/var/run@agent[1] --volume ~/k3s-aiteam-storage/agent2/var/run:/var/run@agent[2] --switch-context

k3d cluster create k3s-aiteam --port 8080:80@loadbalancer --agents 3 --volume /var/run:/var/run:shared --switch-context

kubectl apply -f <https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/longhorn.yaml>

kubectl get pods \

--namespace longhorn-system \

--watch

kubectl describe pods -n longhorn-system longhorn-manager-krptq

Your error is because /var/lib/longhorn is not a shared/propagated mount but longhorn requires a bidirectional mount. You can workaround this by mounting some directory in shared mode, e.g. k3d create --enable-registry --workers 2 --auto-restart --api-port 0.0.0.0:6443 -v $HOME/tmp/longhorn:/var/lib/longhorn:shared.

kubectl -n longhorn-system get pod

Create an ingress with basic authentication

<https://longhorn.io/docs/1.0.2/deploy/accessing-the-ui/longhorn-ingress/>

Access the Loghorn UI

<https://longhorn.io/docs/1.0.2/deploy/accessing-the-ui/>

# Deploy ASP.Net core to Kubernetes

<https://docs.microsoft.com/fr-fr/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/build-aspnet-core-applications-linux-containers-aks-kubernetes>

**Execute Migrations**

cmd

dotnet tool install --global dotnet-ef

cd C:\Users\laure\source\repos\AspNetCoreTest\AspNetCoreTest

dotnet add package Microsoft.EntityFrameworkCore.Design -v 3.1.4

(!! warning, needs to be same version as Npgsql !!)

dotnet build

cd C:\Users\laure\source\repos\AspNetCoreTest\AspNetCoreTest.Data

dotnet ef migrations add InitialCreate --startup-project ..\AspNetCoreTest

dotnet ef database update --startup-project ..\AspNetCoreTest

# Deploy Kubernetes dashboard

<https://rancher.com/docs/k3s/latest/en/installation/kube-dashboard/>