paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Version : 04/08/2024

Prérequis

- Connaissances de base de l’environnement Unix/Linux.

Objectifs

- Comprendre le positionnement de Docker et des conteneurs
- Manipuler l'interface en ligne de commande de Docker pour créer des conteneurs
- Mettre en œuvre et déployer des applications dans des conteneurs
- Administrer des conteneurs
- Déployer rapidement des applications à l'aide de conteneurs
- Identifier les risques et challenges inhérents à Docker afin d’anticiper les bonnes solutions

Tour de table

- Présentation des membres
- Vos attentes vis-à-vis de la formation

Programme de la formation

1. De la virtualisation à Docker
2. Présentation de Docker
3. Mise en œuvre en ligne de commande
4. Création de conteneur personnalisé
5. Mettre en œuvre une application multiconteneur
6. Interfaces d'administration
7. Administrer des conteneurs en production
8. Orchestration et clustérisation
9. Ressources des TP
10. Annexe & Ressources connexes

paul.millet@lameduse.fr

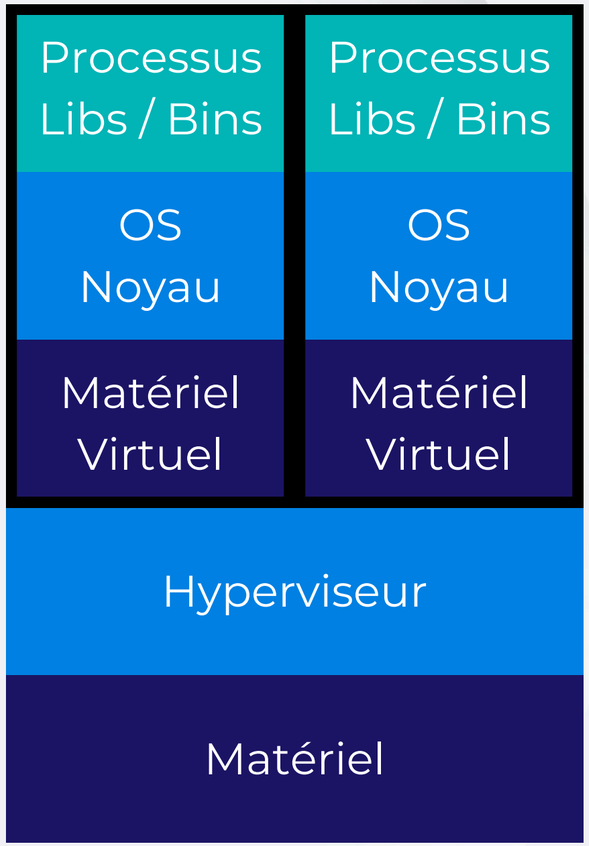
LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

De la virtualisation à Docker

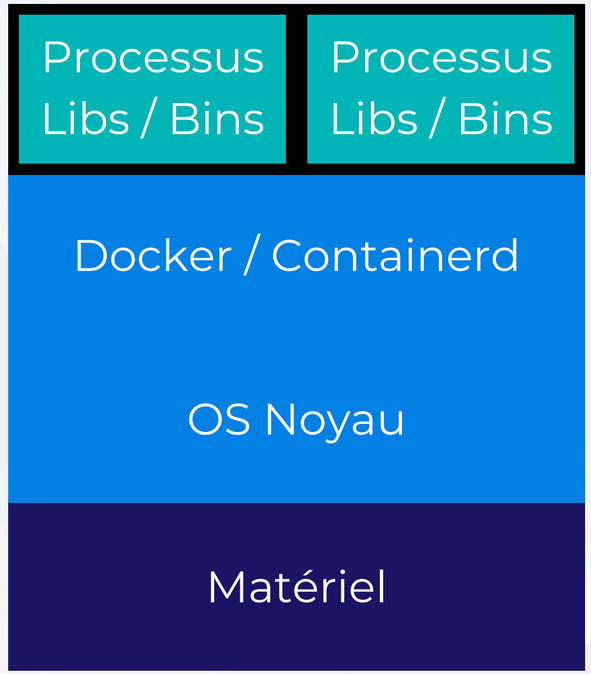
1. Les différents types de virtualisation.
2. La conteneurisation : LXC, namespaces, control-groups.
3. Le positionnement de Docker.
4. Docker versus virtualisation.

Les différents types de virtualisation



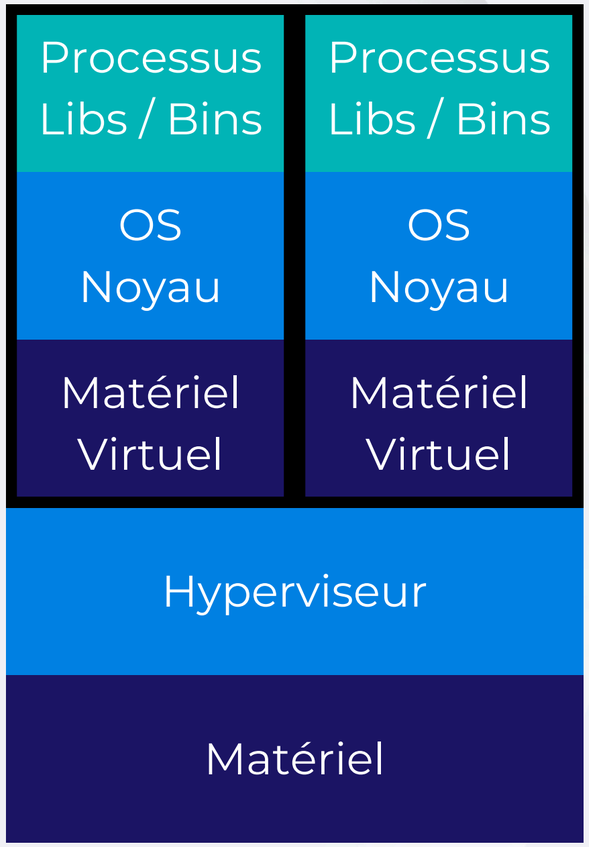
Machine virtuelle

Stockage persistant (Statefull)	Stockage non persistant (Stateless) OU Stockage persistant
Séparation des fichiers et librairie	Séparation des fichiers et librairie
Système d'exploitation indépendant (as son propre kernel)	Utilise les ressources du système d'exploitation (kernel)
Matériel virtualisé	Pas de virtualisation matérielle
Consommation importante Isolation totale	Consommation réduite Isolation partielle



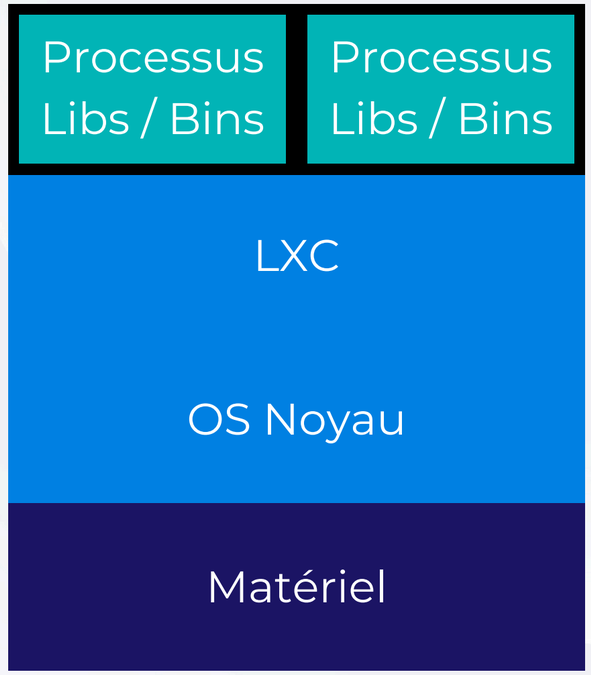
Conteneurisation

La contenerisation : Machine virtuelle VS LXC



Machine virtuelle

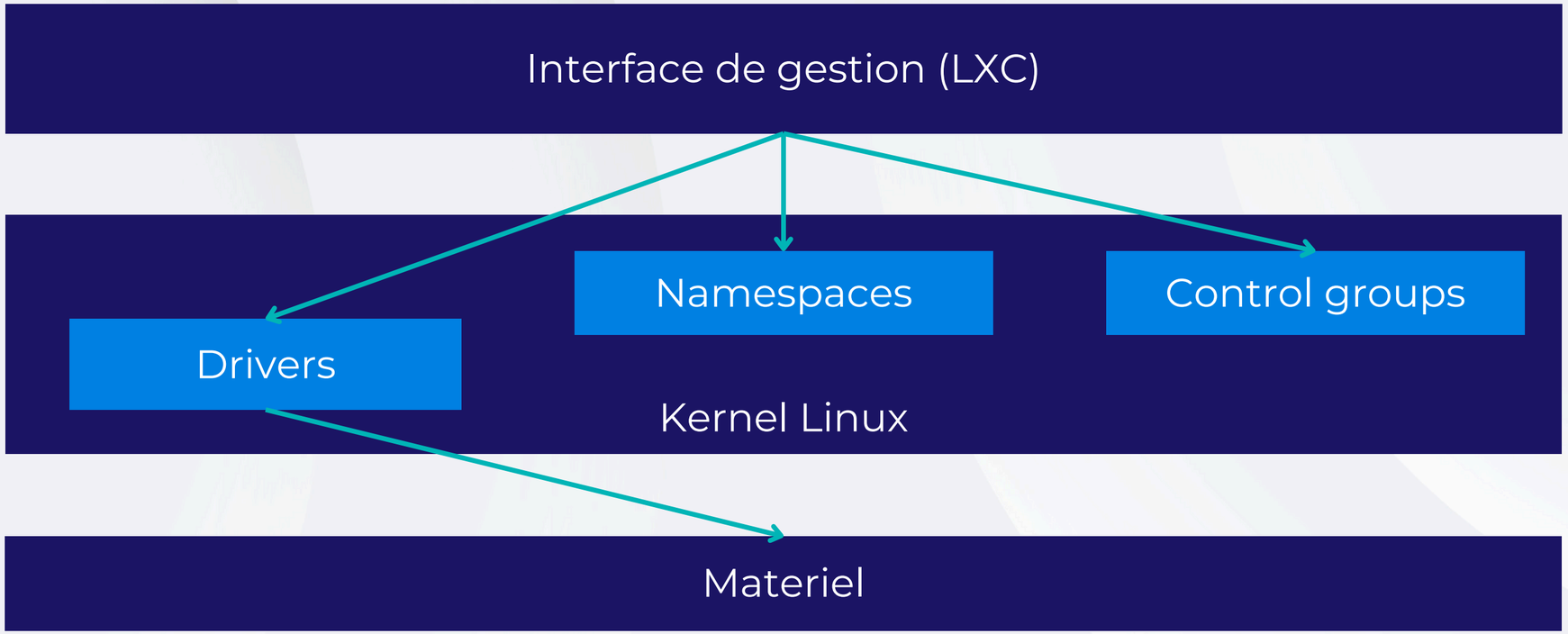
Stockage persistant (Statefull)	Stockage persistant (Statefull)
Séparation des fichiers et librairie	Séparation des fichiers et librairie
Système d'exploitation indépendant (as son propre kernel)	Utilise les ressources du système d'exploitation (kernel)
Matériel virtualisé	Pas de virtualisation matérielle
Consommation importante Isolation totale	Consommation réduite Isolation partielle



Conteneurisation LXC

LaMeDuSe

La contenerisation : namespaces, control-groups et drivers



Qu’est ce que les namespaces 1/3

Définition : permettent l’isolation de ressources pour un ou plusieurs processus.

Types :

- PID : Isolation de l’identifiant de processus. Deux processus peuvent avoir un même identifiant dans deux espaces séparé.
- UTS : Isolation du nom d’hôte et de domaine.
- Mount: Isolation du système de fichier, point de montage, permettant à chaque espace d’avoir sa propre vu du système de fichier.

Qu’est ce que les namespaces 2/3

Types :

- IPC: Isolation de la communication inter-processus (mémoire partagée, semaphore, message queues).
- Network: Isolation des ressources réseaux (Interface, adresse IP, table de routage, règle de pare feu).
- User: Isolation de l’utilisateur et des groupes. Les processus de différent espace ont une vue différente des utilisateurs et groupes. Permets d’exécuter un processus avec des permissions administrateur, alors que le container a des privilèges limités sur le système hôte.

Qu’est ce que les namespaces 3/3

- Cgroup: Isolation de la vue des controlgroups permettant que les processus ne voient que le controlgroup aux quel ils appartiennent et donc sans avoir connaissances des autres présents sur le système.
- Time: Isole le temps du système permettant au processus de différent namespace d’avoir une notion du temps différente (e.g. différentes horloges)

Qu’est ce que les controlgroups

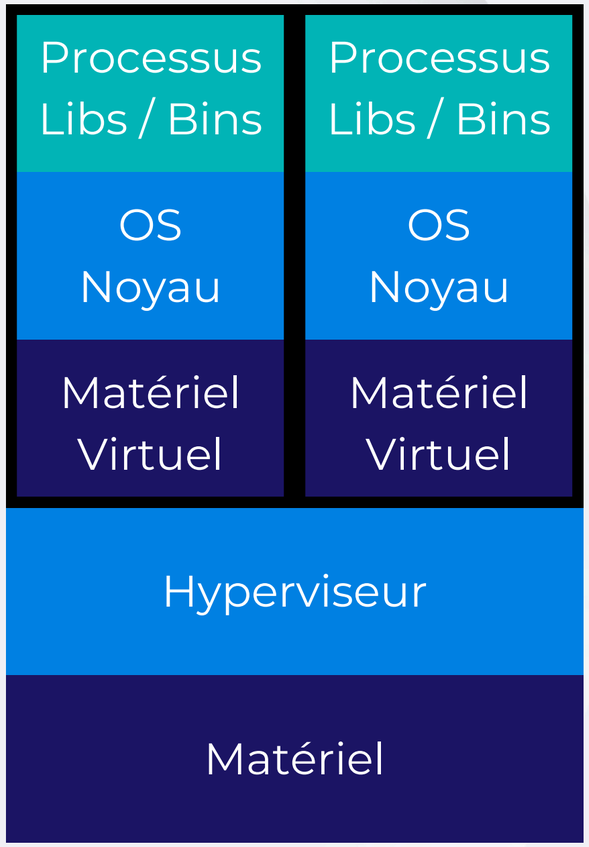
- Permet une limitation de l’usage des ressources par controlgroup.
- CPU: Limitation de l’usage du temps processeurs.
- Memory: Limitation de l’usage de la mémoire. En cas d’excès, le kernel peut exécuter des actions telles que le swapping ou tuer les processus dans le groupe.
- Block I/O: Limitation de l’usage de l’I/O du disque. Empêchant ainsi la saturation du disque.
- Network: Limitation de la bande passante réseau.
- Devices: Autorise ou restreint l’accès à des ressources matérielles (e.g. disques, interfaces réseau).
- PIDs: Limitation du nombre de processus par groupe.

Le positionnement de Docker

- Simplicité de déploiement et de distribution (images docker, dockerfile)
- Abstraction des drivers (stockage, réseau, IPAM, journalisation)
- Simplicité d'utilisation (utilitaire docker)
- Simplicité de déploiement multi-containers (docker compose)
- Optimisation de la gestion du stockage (système de fichier par couche)
- Éco-système d'image faite par la communauté (docker hub)

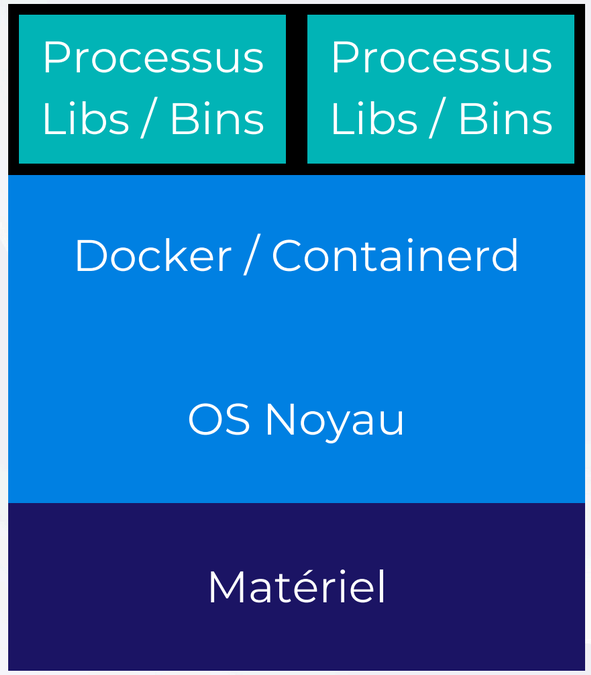
Docker : Standardiser un environnement applicatif pour faciliter son déploiement et sa distribution.

La contenerisation : Virtualisation VS Docker



Machine virtuelle

Stockage persistant (Statefull)	Stockage non persistant par défaut (Stateless)
Séparation des fichiers et librairie	Séparation des fichiers et librairie
Système d'exploitation indépendant (as son propre kernel)	Utilise les ressources du système d'exploitation (kernel)
Matériel virtualisé	Pas de virtualisation matérielle
Consommation importante Isolation totale	Consommation réduite Isolation partielle



Conteneurisation
Docker / Containerd

LaMeDuSe

paul.millet@lameduse.fr

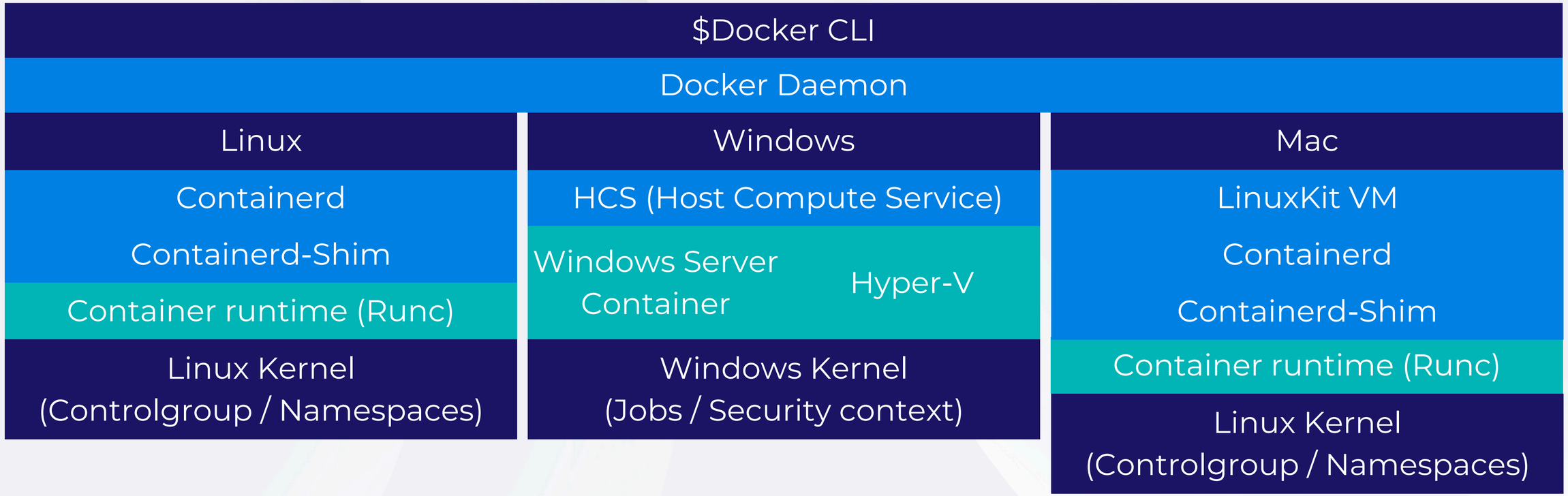
LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Présentation de Docker

- L'architecture de Docker.
- Disponibilité et installation de Docker sur différentes plateformes (Windows, Mac et Linux).
- Création d'une machine virtuelle pour maquettage.
- La ligne de commande et l'environnement.
- TP: Créer une machine virtuelle pour réaliser un maquettage.

L’architecture de Docker



Installation de Docker (Linux)

1. Activer la virtualisation dans le bios
2. Télécharger et exécuter le script Docker https://get.docker.com
3. Activer et lancer le service docker
4. Ajouter l'utilisateur au groupe docker (permet à celui-ci d'interagir avec le service docker en utilisant l'utilitaire docker)

Installation de Docker (Windows)

1. Activer la virtualisation dans le bios
2. Activer Hyper-V et la plateforme de container dans les fonctionnalités de windows
3. (optionnel) Activer la plateforme Windows Subsystem for Linux (WSL)
4. Télécharger et installer Docker Desktop https://docs.docker.com/desktop/release-notes/

Notes :

- Sur les environnements Windows avec WSL vous pouvez exécuter des containers Docker pour Linux.

Installation de Docker (Mac)

1. Télécharger et installer Docker Desktop
   https://docs.docker.com/desktop/release-notes/

Notes :

- Docker supporte officiellement les deux dernières versions de MacOS
- Pour les Mac avec processeur Apple il est recommandé d'installer Rossetta 2 (softwareupdate --install-rosetta)

Création d'une machine virtuelle pour maquettage (Etapes)

1. Activer la virtualisation matérielle (Intel VT-x / AMD-V)
2. Installer l’outil de virtualisation (VMware, VirtualBox, …)
3. Configurer la machine virtuelle
    a. Activer la virtualisation (Intel VT-x / AMD-V)
4. Installer le système d’exploitation pour accueillir docker

La ligne de commande et l'environnement

Gestion du container

- run [option] image ⇒ exécuter une image
- start [option] nom_du_container ⇒ démarre un container
- stop [option] nom_du_container ⇒ stoppe un container
- restart [option] nom_du_container ⇒ rédémarre un container
- kill [option] nom_du_container ⇒ tue un container
- exec [option] nom_du_container commande ⇒ execute une commande dans un container
- attach [option] nom_du_container ⇒ attache la console à l'entrée, sortie et sortie d'erreur d'un container
- container list [option] => liste les containers en cours d'exécution
  - (--all : listes tout les containers, y compris ceux stoppé)

La ligne de commande et l'environnement

gestion des images

- build [option] chemin_dockerfile ⇒ créer une image docker a partir d’un dockerfile
- push [option] nom_image:[tag] ⇒ envoie une image docker a un registre
- image ls ⇒ liste les images docker
- image prune ⇒ retire toute les images non utilisée
- image rm [option] nom_image ⇒ supprime l’image

La ligne de commande et l'environnement

gestion des réseaux

- network create [option] nom_reseau ⇒ créer un réseau
- network rm [option] nom_reseau ⇒ supprime un réseau
- network prune ⇒ supprime les réseaux non utilisé
- network ls ⇒ list les réseaux
- network connect [option] nom_reseau nom_container ⇒ connecte le container a un réseau
- network disconnect [option] nom_reseau nom_container ⇒ déconnecte le container d’un réseau

La ligne de commande et l'environnement

gestion des volumes

- volume create [option] [nom_du_volume]
- volume inspect [option] nom|id
- volume ls
- volume prune
- volume rm [option] nom_de_volume

TP : Créer une machine virtuelle pour réaliser un maquettage

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Mise en œuvre en ligne de commande

- Mise en place d'un premier conteneur.
- Le Docker hub : ressources centralisées.
- Mise en commun de stockage interconteneur.
- Mise en commun de port TCP interconteneur.
- Publication de ports réseau.
- Le mode interactif.
- TP : Configurer un conteneur en ligne de commande.

Mise en place d'un premier conteneur.

On execute la commande $docker run hello-world

Containerd vérifie si l’image existe localement

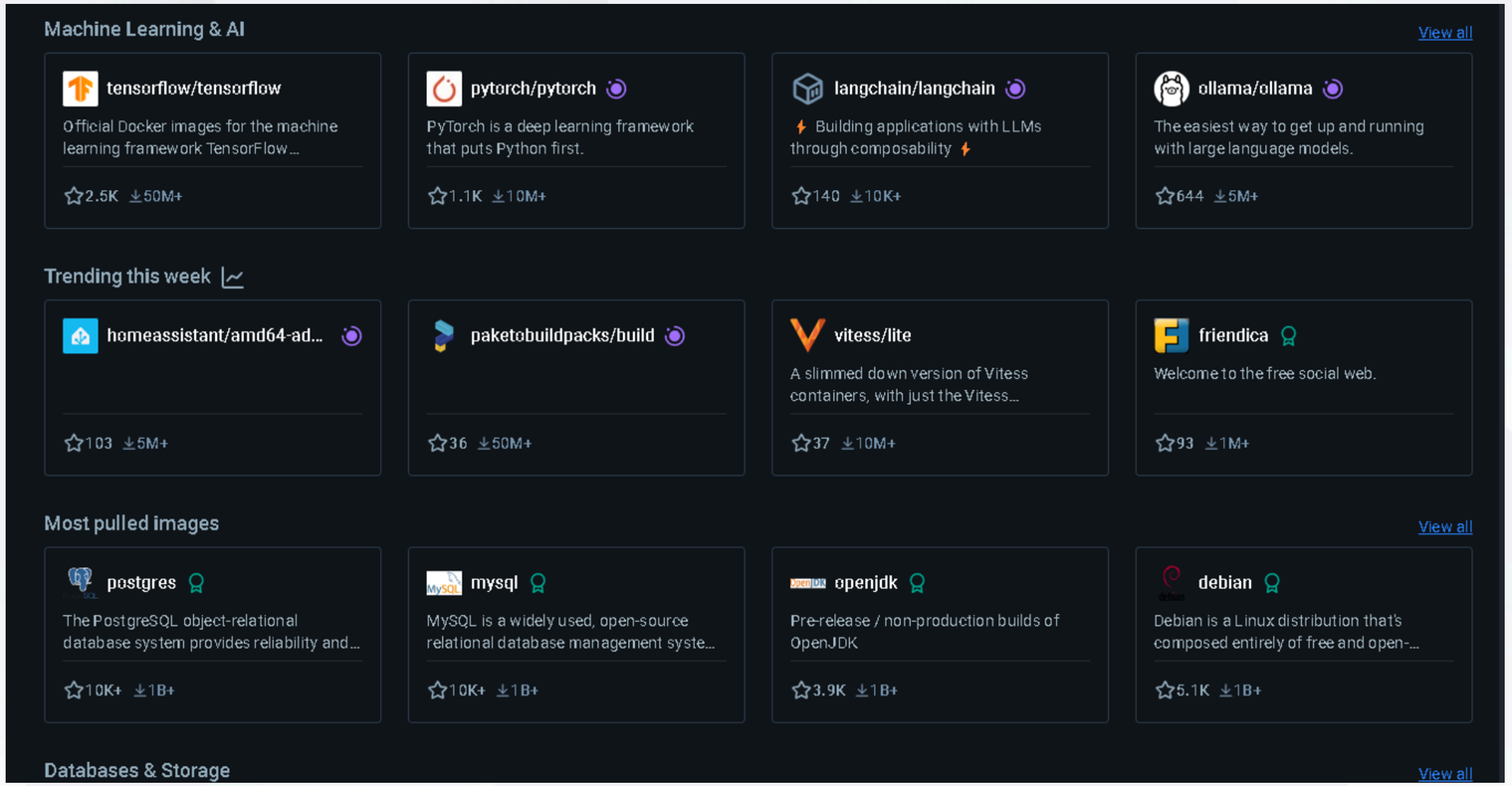
Containerd récupère l’image du container demandée nommée “hello-world”

Lancement du container

```
warstrolo@dev-xubuntu-01:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53cc4d415d839c98be39331c948609b659ed725170ad2ca8eb36951288f81b75
Status: Downloaded newer image for hello-world:latest

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

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

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

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

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

Le Docker hub : ressources centralisées.

- url : https://hub.docker.com
- Permet l'accès à une collection d'image faite par la communauté
- Peut héberger vos images sans avoir à déployer un registre soi-même



Mise en commun de stockage interconteneur.

- Création du volume docker : “random_volume”
  - docker volume create random_volume
- Lancement des premiers container dans deux terminaux séparé
  - docker run --name writer --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_filewrite
  - docker run --name reader --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_fileread
- options
  - --name : assigne un nom au container, permet de le spécifier après dans les commandes (rm, stop, start)
  - --volume : assigne le volume au container et le monte sur “/tmp”
  - registry.lameduse.....filewrite : nom de l’image que nous utilisons

Différence entre un volume et un “bind-mount”

- Volume Docker :
  - Stockage géré par docker
  - Chemin (sous linux): /var/lib/docker/volumes/
- Bindmount:
  - Stockage géré par le système hôte
  - Le chemin est celui que vous avez choisi

Mise en commun de stockage interconteneur.

```
warstrolo@dev-xubuntu-01:~$ docker volume create random_volume
random_volume
warstrolo@dev-xubuntu-01:~$ docker run --name writer --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_filew
rite
Unable to find image 'registry.lameduse.fr/public/lameduse-ressources-dok_filewrite:latest' locally
latest: Pulling from public/lameduse-ressources-dok_filewrite
c6a83fedfae6: Already exists
9ff7791c907d: Pull complete
c9f58c058851: Pull complete
ac275217c340: Pull complete
Digest: sha256:3f449e1c927d2dbf80e37e7d36bb4ed4bfb95c6dc9ded224a51b005aae812f43
Status: Downloaded newer image for registry.lameduse.fr/public/lameduse-ressources-dok_filewrite:latest
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:39:58 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:00 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:02 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:04 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:06 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:08 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:10 UTC 2024
Writing date and time to random_file:
Contents of random_file:
Fri Aug 16 13:40:12 UTC 2024
Writing date and time to random_file:
Contents of random_file:
```

```
warstrolo@dev-xubuntu-01:~$ docker run --name reader --volume random_volume:/tmp registry.lameduse.fr/public/lameduse-ressources-dok_filer
ead
Unable to find image 'registry.lameduse.fr/public/lameduse-ressources-dok_fileread:latest' locally
latest: Pulling from public/lameduse-ressources-dok_fileread
c6a83fedfae6: Already exists
e68bca6f1f98: Pull complete
9473657d6aaa: Pull complete
46a9cdf9721f: Pull complete
Digest: sha256:63d4985104dac08edd07e2dc64e5aa84cbbf9010f55d1dc3156e6964cbd71e24
Status: Downloaded newer image for registry.lameduse.fr/public/lameduse-ressources-dok_fileread:latest
Reading from random_file:
Fri Aug 16 13:40:28 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:33 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:35 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:37 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:39 UTC 2024
Reading from random_file:
Fri Aug 16 13:40:41 UTC 2024
```

Mise en commun de port TCP interconteneur

- Création du réseau docker : “random_network”
  - docker network create random_network
- Lancement des premiers containers dans deux terminaux séparé
  - docker run --name reader --network random_network registry.lameduse.fr/public/lameduse-ressources-dok_netread
  - docker run --name writer -it --network random_network -e TARGET_HOST=reader registry.lameduse.fr/public/lameduse-ressources-dok_netwrite
- options
  - --network: assigne le réseau au container
  - -e : lance le container avec la variable d’environement
  - -it : mode intéractif

Mise en commun de port TCP interconteneur

```
warstrolo@dev-xubuntu-01:~$ docker network create random_network
4fcfc5b25e96b8f309047e58746d1141f4ce1c2e1453b12f23a54f87f27dcbc8
```

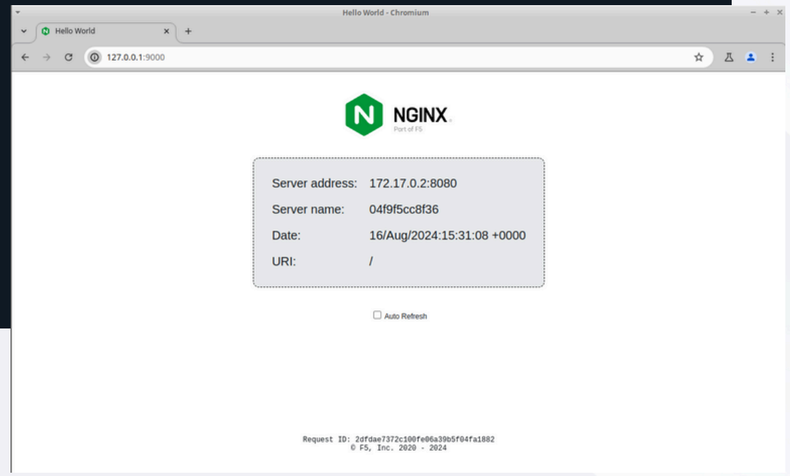
```
warstrolo@dev-xubuntu-01:~$ docker run --name reader --network random_network registry.lameduse.fr/public/lameduse-ressources-dok_netread
Server is listening on port 8080
Received message: test

Received message: hello

Received message: world

```

```
cwarstrolo@dev-xubuntu-01:~$ docker run --name writer -it --network random_network -e TARGET_HOST=reader registry.lameduse.fr/public/lameduse-ressources-dok_netwrite
test
Response: Message received
hello
Response: Message received
world
Response: Message received
```

Publication de port réseau

- docker run -p 8080:8080 --name nginx nginxdemos/nginx-hello
- -p port_hote:port_container : spécifie la redirection de port

```
warstrolo@dev-xubuntu-01:~$ docker run -p 8080:8080 nginxdemos/nginx-hello
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf is not a file or does not exist
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/08/16 15:28:11 [warn] 1#1: the "user" directive makes sense only if the master process runs with super-user privileges, ignored in /etc/nginx/nginx.conf
:2
nginx: [warn] the "user" directive makes sense only if the master process runs with super-user privileges, ignored in /etc/nginx/nginx.conf:2
2024/08/16 15:28:11 [notice] 1#1: using the "epoll" event method
2024/08/16 15:28:11 [notice] 1#1: nginx/1.27.1
2024/08/16 15:28:11 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
2024/08/16 15:28:11 [notice] 1#1: OS: Linux 6.8.0-40-generic
2024/08/16 15:28:11 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/08/16 15:28:11 [notice] 1#1: start worker processes
2024/08/16 15:28:11 [notice] 1#1: start worker process 20
2024/08/16 15:28:11 [notice] 1#1: start worker process 21
2024/08/16 15:28:11 [notice] 1#1: start worker process 22
2024/08/16 15:28:11 [notice] 1#1: start worker process 23
2024/08/16 15:28:11 [notice] 1#1: start worker process 24
2024/08/16 15:28:11 [notice] 1#1: start worker process 25
2024/08/16 15:28:11 [notice] 1#1: start worker process 26
2024/08/16 15:28:11 [notice] 1#1: start worker process 27
```



Le mode interactif

- Création du container
  - docker run -p 8080:8080 --name nginx nginxdemos/nginx-hello
- Utilisation du mode interactif pour interagir avec la console (sh)
  - docker exec -it nginx /bin/sh
- Options
  - -i : mode interactif permet d'interagir avec la console
  - -t : met en place l'interface du terminal (tty)

```
warstrolo@dev-xubuntu-01:~$ docker exec -it nginx /bin/sh
/ $ hostname
1ba71073f06d
/ $ ls
bin                  docker-entrypoint.sh  lib                  opt                  run                  sys                  var
dev                  etc                   media                proc                 sbin                 tmp
docker-entrypoint.d  home                  mnt                  root                 srv                  usr
/ $
```

Le mode détaché

- Création du container
  - docker run -d -p 8080:8080 --name nginx nginxdemos/nginx-hello
- Utilisation du mode détaché qui permet au container de s'exécuter en arrière-plan
- Options
  - -d : mode détaché

```
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$ docker run -d -p 8080:8080 --name nginx nginxdemos/nginx-hello
28319a0272d89728edfb0be74bee733481a18ba48fffad94d93e374dbcae7734
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$
```

TP : Configurer un conteneur en ligne de commande

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Création de conteneur personnalisé

- Produire l'image de l'état d'un conteneur.
- Qu'est-ce qu'un fichier Dockerfile ?
- Automatiser la création d'une image.
- Mise en œuvre d'un conteneur.
- Conteneur hébergeant plusieurs services : supervisor.
- TP : Créer un container personnalisé.

Produire l'image de l'état d'un conteneur

- La commande docker commit permet de créer une image à partir de l’état d’un container
- Usage
  - docker commit [options] nom_container [nom_image][:tag]

Qu'est-ce qu'un fichier Dockerfile ?

- Le fichier Dockerfile permet de spécifier les instructions de création d’une image
- Format :
  - # Commentaire
  - INSTRUCTION [argument]
- Les instructions par convention sont écrites en majuscule (bien que pas obligatoire)
- Les instructions sont exécuté dans l’ordre

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- ADD [options] <source...> <destination> : Ajoute un fichier local, distant ou un répertoire
  - La source est une archive tar : ADD décompresse l’archive et ajoute son contenu à la destination
  - La source est un répertoire git : clone le répertoire et l’ajoute à la destination
  - La source est une url : télécharge le fichier et l’ajoute a la destination
- ARG <argument> [=valeur_par_default] : Permet de passer un argument lors de la création de l’image
  - L’option “--build-arg <argument=valeur>” doit être passé dans les options lors de la commande “docker build”

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- CMD : Définie la commande par défaut utilisée pour lancer le container
  - Particularité : Peut-être réécrite lors de l’exécution du container
    - docker run image [CMD]
  - Formes
    - CMD [”executable”, “argument1”, “argumentN”…] : spécifie l’executable
    - CMD [”argument1", “argumentN”] : utilise l’exécutable par défaut avec les arguments spécifiés
- COPY : Copie la / les source(s) vers la destination
  - Forme : COPY [options] <source…> <destination>

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- ENTRYPOINT : Spécifie l'exécutable par défaut
  - Forme : ENTRYPOINT ["executable", "argument1", "argumentN"]
  - Différence avec CMD : Les instructions passées par CMD sont ajoutés après celles spécifiées par ENTRYPOINT
- ENV : Définie une variable d'environnement
  - Forme : ENV <nom>=<valeur>
- EXPOSE : Définie les ports utilisés par le container
  - Note :
    - Ne publie pas les ports par défaut
    - Sans options au moment de l'exécution ne sert à rien d'autre qu'à documenter
    - Avec l'option -P : Docker publie automatiquement les ports

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- FROM : Créer une étape de construction à partir d’une image existante
  - Forme : FROM <nom_image>[:tag] [AS nom_utilisation]
  - Note :
    - Première instruction donnée lors de la définition du Dockerfile
    - le nom d’utilisation est utilisé lorsque l’image a besoin de plusieurs images différentes pour se construire (par exemple une image qui permet la compilation et une autre image pour l’exécution)

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- RUN : Exécute une commande lors de la construction de l'image
  - Formes :
    - RUN [option] <commande>
    - RUN [option] [<commande1>, <commandeN>]
- SHELL : Change la console utilisée par default
  - Forme : SHELL [<executable>, [paramètre]]
  - Note : Cette console est utilisée par l'instruction RUN
- USER : Change l'utilisateur qui exécute les commandes
  - Forme : USER <utilisateur>[:groupe]

Qu'est-ce qu'un fichier Dockerfile ? (Liste des instructions)

- WORKDIR : Change le répertoire d’exécution
  - Forme : WORKDIR <répertoire>
  - Note :
    - Si le répertoire n’existe pas, il est automatiquement créé
    - Définie le répertoire par défaut pour les commandes
      - RUN
      - CMD
      - ENTRYPOINT
      - COPY
      - ADD
    - Si le chemin est relatif, il sera relatif au précédent répertoire d’execution

Automatiser la création d’une image

```
# On utilise l’image de base alpine:latest
FROM alpine:latest

# On installe bash
RUN apk add --no-cache bash

# On copie le script “write-file.sh”
COPY write-file.sh /write-file.sh

# On authorise l’execution du script
RUN chmod +x /write-file.sh

# On définit le script comme la commande d’execution par default
CMD ["/write-file.sh"]
```

Mise en œuvre d'un conteneur (construction de l’image)

On exécute la commande “docker build”

1. “docker build <contexte>” : Créer l’image
2. “docker build -t <nom_image>[:tag] <contexte>” : Créer l’image et la nomme
3. “docker build -f <chemin_dockerfile> <contexte>” : Créer l’image en utilisant un Dockerfile extérieur à la racine du contexte

Note :

- le contexte est le chemin ou se situe le contexte
  - sans l’option -f celui-ci doit contenir le fichier Dockerfile a la racine

```
warstrolo@dev-xubuntu-01:/mnt/hgfs/WORK_DATA/lameduse/lameduse-ressources/dok_netwrite$ docker image list
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
<none>                                                    <none>    2871c9ff59bb   10 seconds ago   128MB
toto                                                      latest    9649558da5ac   3 minutes ago    128MB
registry.lameduse.fr/public/lameduse-ressources-dok_filewrite   latest    5b1c0e33ea6c   47 hours ago     9.83MB
```

Mise en œuvre d'un conteneur (publication de l’image)

On exécute la commande “docker push”

1. “docker push <nom_image>[:tag]” : envoie l’image vers le registre
    a. si l’image est nommée sans le registre (e.g. image ou repo/image) elle est envoyée par défaut sur le docker hub (requiert une authentification)
    b. si l’image est nommée avec le registre (e.g. registry.organisation.com/repo/image) elle est envoyé sur le registre spécifié
2. “docker save -o <chemin_archive> <nom_image>[:tag]”
    a. exporte l’image dans l’archive spécifié

Mise en œuvre d'un conteneur (récupération de l’image)

On exécute la commande “docker push”

1. “docker pull <nom_image>[:tag]”
    a. récupère l’image et l’envoie sur le registre interne de l’hôte docker
2. “docker load -i <chemin_archive>”
    a. importe l’image depuis l’archive spécifiée
    b. garde le nom spécifié lors de l’export

Conteneur hébergeant plusieurs services : supervisor

Le gestionnaire de processus

- Permet la mise en place de plusieurs processus en utilisant une configuration
- Il faut le définir comme point d'entrée (entrypoint) dans le Dockerfile

Note :

- Avoir plusieurs services dans le même container n'est pas recommandée

Conteneur hébergeant plusieurs services : supervisor

Exemple : Supervisord

- Lance un processus: “supervisorctl start <application>”
- Arrête un processus: “supervisorctl stop <application>”
- Vérification du status: “supervisorctl status”

```
[supervisord]
logfile=/var/log/supervisord.log   ; Localisation du fichier de journal
pidfile=/var/run/supervisord.pid   ; Localisation du fichier PID

[program:application]
command=/chemin/executable --option    ; Commande qui execute l’application
autostart=true                  ; Démarer le processus au démarage de supervisord
autorestart=true                ; Redémarage automatique du processus
stderr_logfile=/var/log/myapp.err  ; Localisation de la sortie d’erreur
stdout_logfile=/var/log/myapp.out  ; Localisation de la sortie standard
```

Une image docker en production

Objectif :

- Toute la compilation a été faite avant
- Temps de démarrage très court

En production (avec une plateforme de type k8s) :

- Besoin de plus de puissance, création automatique de plusieurs containers pour supporter le besoin
- Besoin de moins de puissance, suppression de containers automatique pour réduire la consommation

TP : Créer un conteneur personnalisé

5 - Mettre en œuvre une application multiconteneur

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Mettre en œuvre une application multiconteneur

- Utilisation Docker Compose
- Création d'un fichier YAML de configuration
- Déployer plusieurs conteneurs simultanément
- Lier tous les conteneurs de l'application
- TP : Mettre en œuvre une application multiconteneur

Présentation Docker Compose

- Docker Compose permet de déployer un environnement complet
- Docker Compose configure :
  - Les services : container unique faisant partie de l'environnement
  - Les volumes : volume utilisé par l'environnement et assignable à un ou plusieurs services
  - Les réseaux : réseaux utilisés par l'environnement assignable a un ou plusieurs services
- Les services sont accessibles par leur nom entre eux

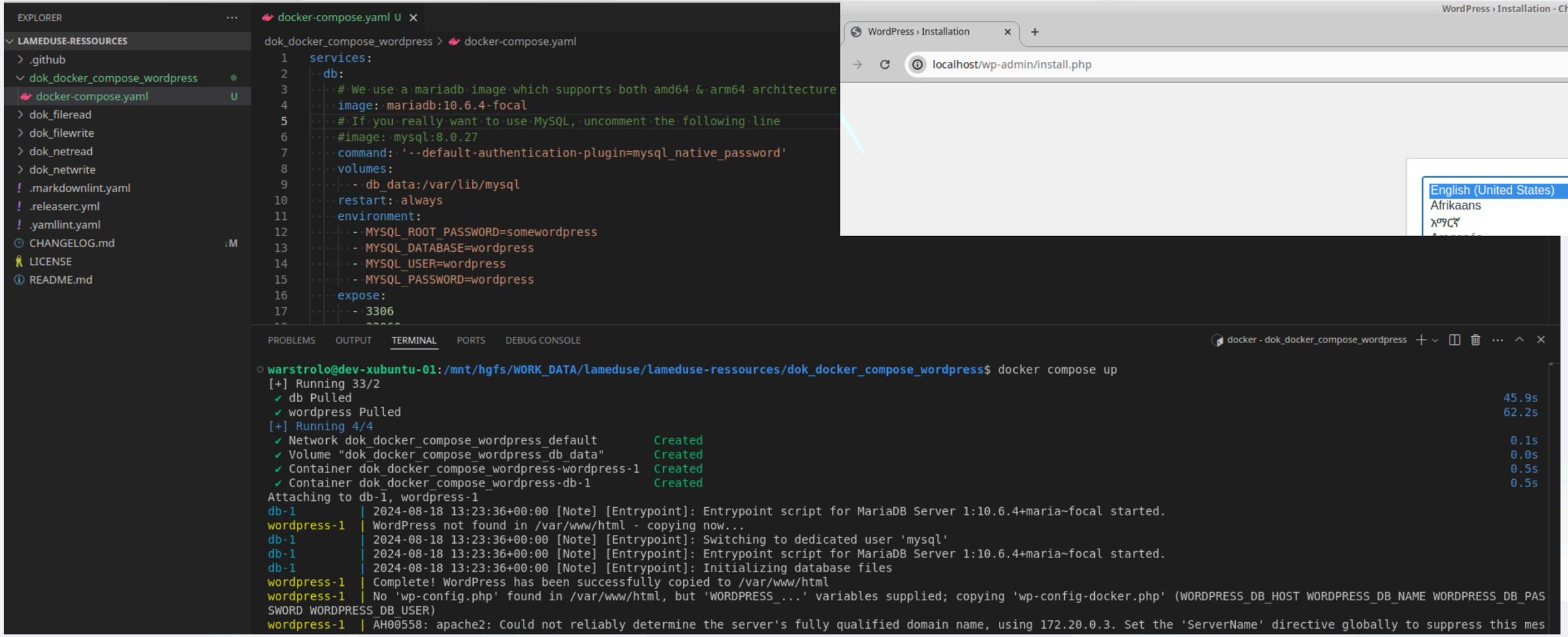
Utilisation Docker Compose

- docker compose up [option]
  - Met en place l'environnement (ressources, containers)
  - Lance l'environnement
- docker compose down [option]
  - Arrête l'environnement
  - Supprime l'environnement (containers, ressources)
    - Sauf les volumes (il faut l'option -v pour retirer les volumes)
- docker compose start [service…]
  - Lance tous les services existants ou seulement ceux spécifiés
- docker compose stop [service…]
  - Arrête tous les services existant ou seulement ceux spécifiés

Création d'un fichier YAML de configuration (exemple)

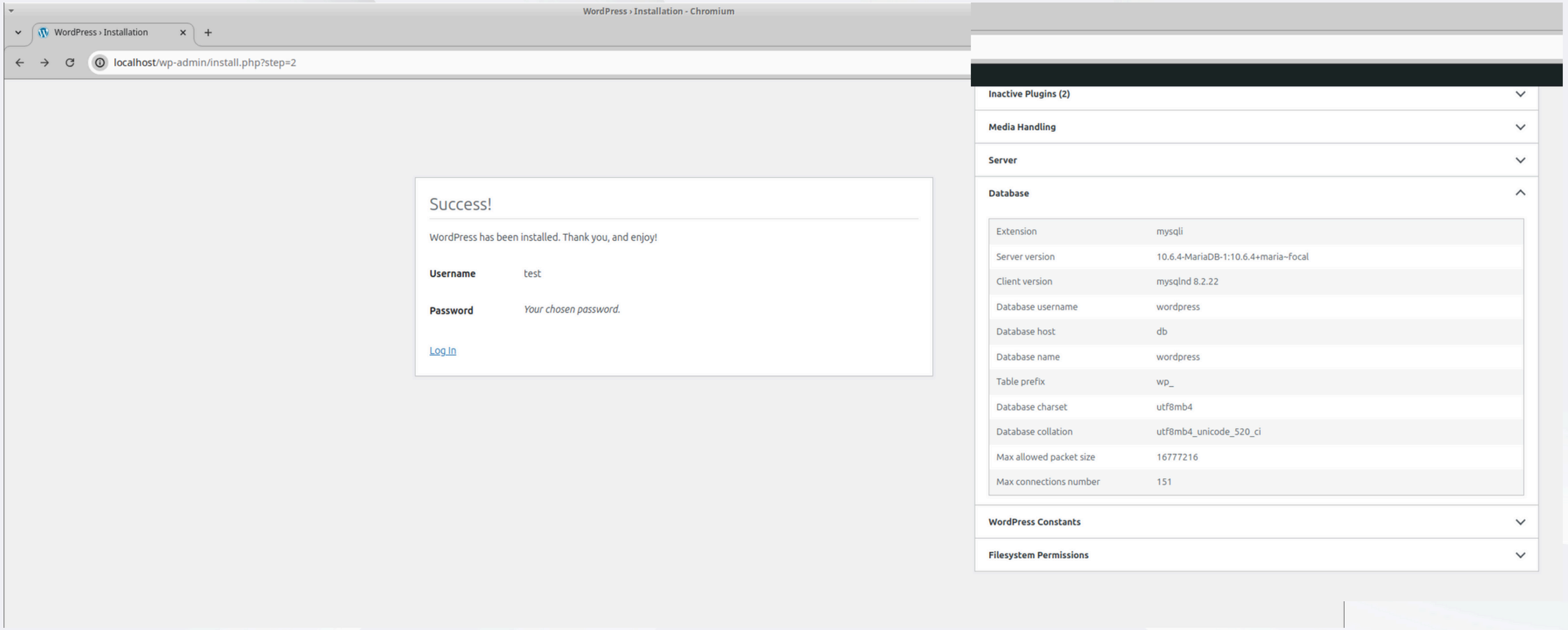
```
services:
  db:
    image: mariadb:10.6.4-focal #définie l'image
    command: '--default-authentication-plugin=mysql_native_password' #Option CMD dans docker run
    volumes:
      - db_data:/var/lib/mysql #definie l'utilisation et le montage du volume db_data
    restart: always #stratégie de redémarrage
    environment:  #variable d'environnement
      - MYSQL_ROOT_PASSWORD=somewordpress
      ...
    expose: #quel port sont exposés pour les autres services
      - 3306
  wordpress:
    image: wordpress:latest
    ports: #quel port sont exposés publiquement
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      ...
volumes: #définition du volume
  db_data:
```

Déployer plusieurs conteneurs simultanément



LaMeDuSe

Lier tous les conteneurs de l'application



TP : Mettre en œuvre une application multiconteneur

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Mettre en œuvre une application multiconteneur

- L'API Docker et les Web Services
- Interface d'administration en mode Web
- Héberger son propre registre : Docker Registry, Gitlab-CE...
- TP : Construire et utiliser son propre registre

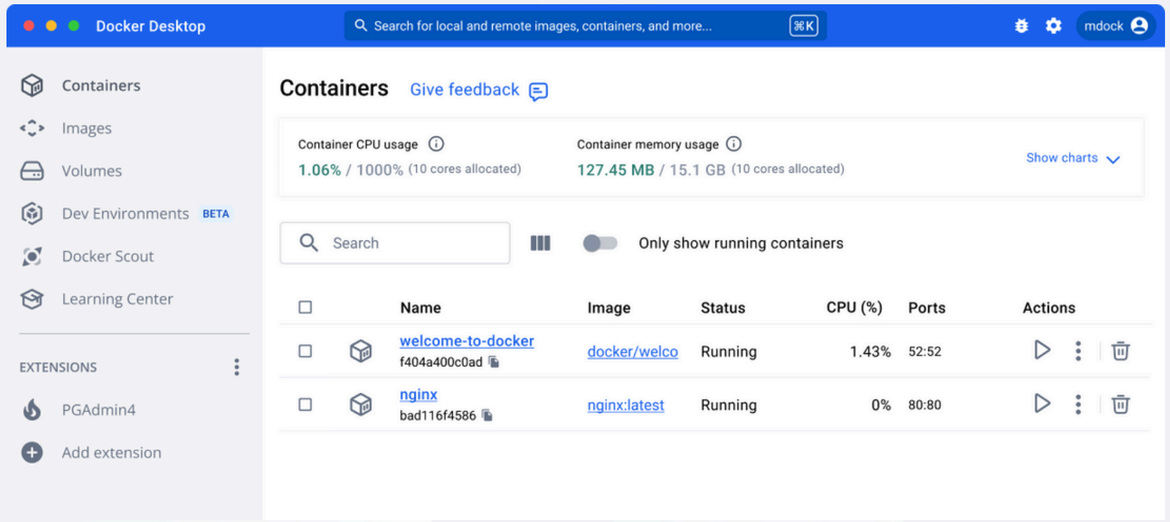
L'API Docker et les Web Services

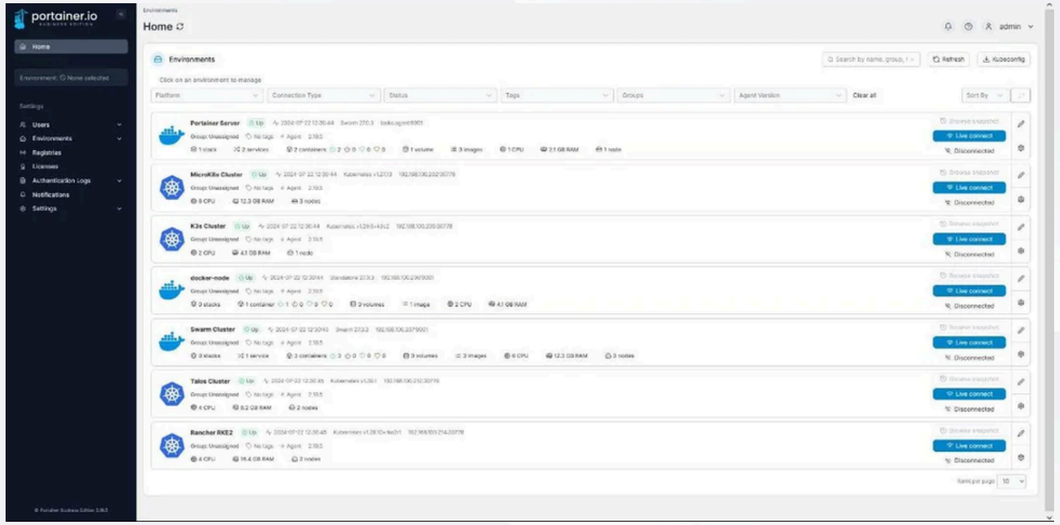
- Sous linux Docker rend accessible le socket unix “/var/run/docker.sock”
- Ce socket permet l’exécution des commandes “docker …”
- Il est également utilisé par des applications tierces qui permettent d’effectuer des opérations sur le daemon de docker

Interface d'administration en mode Web

Il existe plusieurs solutions pour administrer Docker via une application graphique

- Docker Desktop : Permet de gérer ses containers, d'explorer les logs
- Portainer : Permet de gérer des nœuds Docker (possible de le déployer en production)





Héberger son propre registre : Docker Registry, Gitlab-CE...

Pour rendre héberger une image docker de façon à la rendre accessible, il existe plusieurs solutions

- Docker Hub (freemium/cloud) : Registre officiel de docker
- Docker Registry (oss/selfhosted) : Solution de registre de Docker open-source (pas de dashboard)
- Harbor (oss/selfhosted) : Solution de registre faite par VMware (dashboard + option de sécurité)
- ghcr.io (freemium/cloud) : Registre propulsé par GitHub
- GitLab container registry (freemium/cloud & hosted) :
    - Registre propulsé par GitLab

TP : Construire et utiliser son propre registre

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Administrer des conteneurs en production

- Automatiser le démarrage des conteneurs au boot.
- Gérer les ressources affectées aux conteneurs.
- Gestion des logs des conteneurs.
- Sauvegardes : quels outils et quelle stratégie ?
- Administrer les conteneurs.

Automatiser le démarrage des conteneurs au boot

- Avec Docker run
  - l'option "--restart" permet de definir une politique de redémarage
    - unless-stopped : redémarre le container sauf si explicitement stoppé par l'utilisateur
      - redémarre après un arrêt du système
    - on-failure : redémarre si le container rencontre une erreur
      - ne démarrera pas après un après du système
    - always : redémarre toujours le containers

Automatiser le démarrage des conteneurs au boot

- Avec Docker Compose

service:
    web:
        image: <image>
        restart: <stratégie de redémarage (même que celle de docker run)>

Gérer les ressources affectées aux conteneurs

Pourquoi ?

- Pour éviter de saturer le système, il est important de limiter l'usage des containers (mémoire, temps processeur)
- Par défaut, les containers n'ont aucune limite, ils peuvent donc utiliser toutes les ressources disponibles sur le noyau

Comment ?

- Les limites sont données lors de la création du container et ne sont pas changeable sans la destruction et recréation du container

Gérer les ressources affectées aux conteneurs

Limiter la mémoire

- “--memory <limite>” : définie la limite mémoire
- “--memory-swap <limite>” : définie la limite du swap

Note :

- Les unités de mémoire (bit, kilo, mega, giga) sont respectivement (b, k, m, g)
- e.g. pour limiter un container à un giga de mémoire vive (--memory 1g)

Gérer les ressources affectées aux conteneurs

Limiter le CPU

- “--cpus <limite>” : définie la limite CPU (e.g. 2.5 = 2 cœurs et demi)
- “--cpuset-cpus <cores>” : définie les cœurs attribués au container
  - 0 = 1er cœur
  - 0-3 : cœurs 1 2 3 4
  - 1,3 : 2e et 4e cœurs

Gestion des logs des conteneurs

- Si le container est lancé sans l’option de détachement, les logs apparaissent directement dans la console
- Sinon
  - “docker logs [option] <nom_du_container>”
    - --since : pour afficher le log depuis un temps donné
    - --until : pour afficher le log jusqu'à un temps donné
    - --tail : affiche les x dernières lignes
- note
  - pour l’option de temps, on peut spécifier une date ou un temps relatif
    - e.g. Nh ou Nm ou Ns avec N étant un nombre

Sauvegardes : quels outils et quelle stratégie ?

Stratégies

- Les containers sont par convention non persistant donc toutes les données présentes avec eux sont censées pouvoir être supprimés
  - Les données persistantes doivent être dans un volume ou un bind-mount
- “docker commit” : permet de créer une image à partir d’un container existant
- “docker export” : créer une archive à partir d’un container
- “docker import” : charge une archive et créer une image à partir de l’archive

TP : Administrer les conteneurs

paul.millet@lameduse.fr

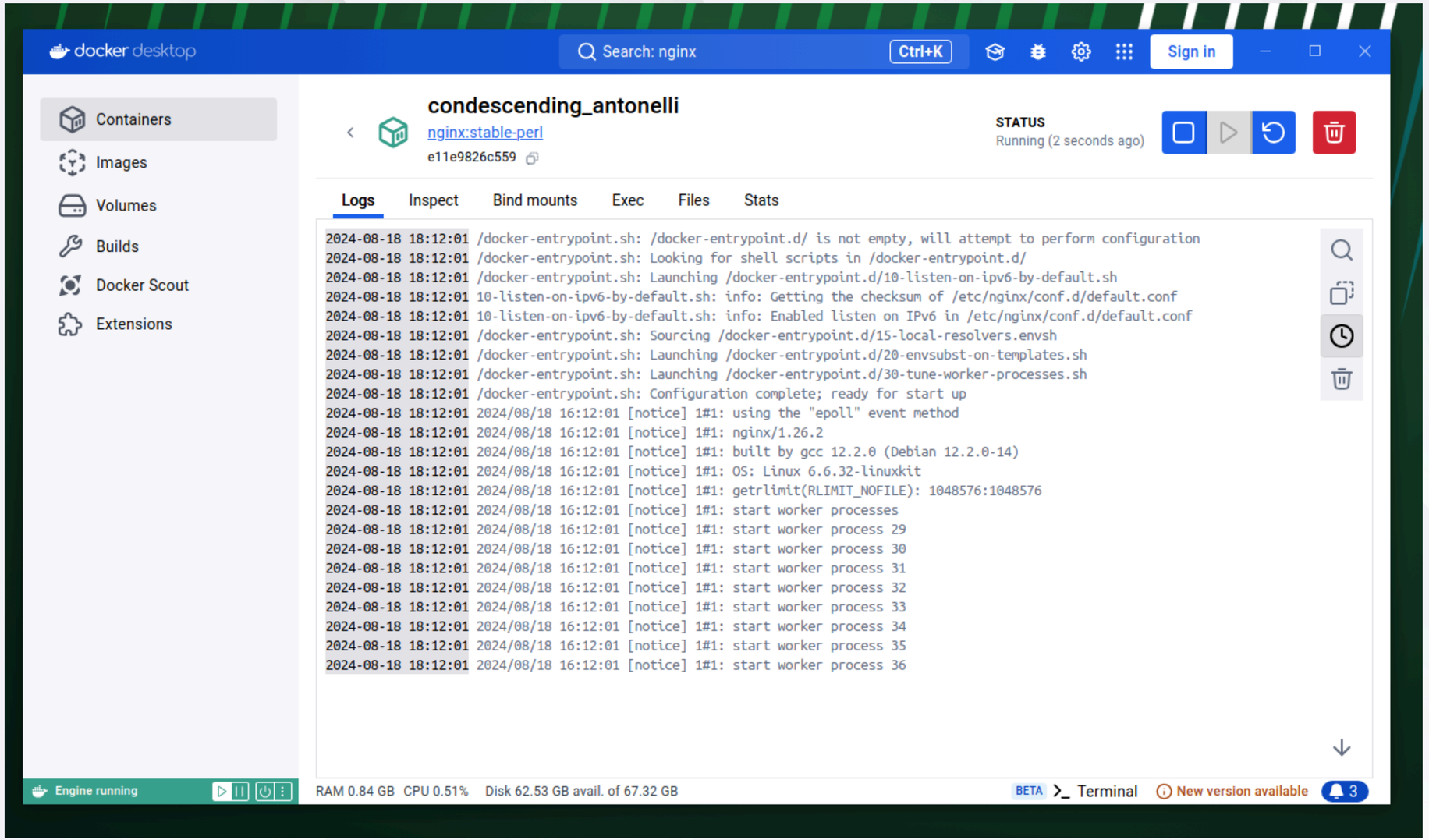
LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

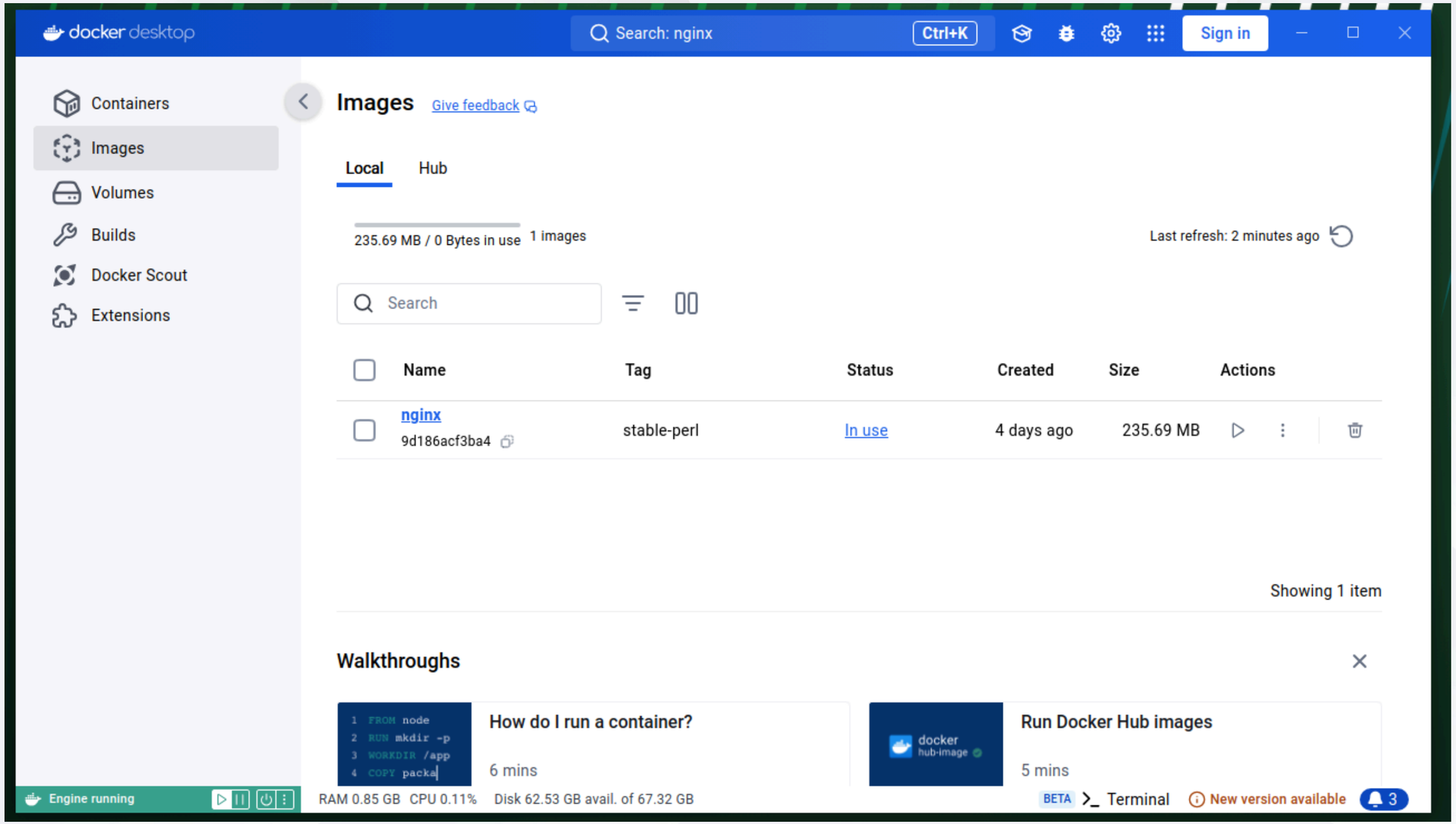
Orchestration et clusterisation

- Présentation de Docker Desktop.
- L'orchestrateur Swarm : nodes, services, secrets, configs.
- Déploiement de services et stacks dans un Swarm.
- Reverse-proxy et load-balancer pour Web Services en cluster (Traefik...).
- TP : Création d'un cluster Swarm.
- TP : Gestion des nœuds dans le cluster.
- TP : Publication et mise à l'échelle d'un service dans le cluster.

Présentation de Docker Desktop

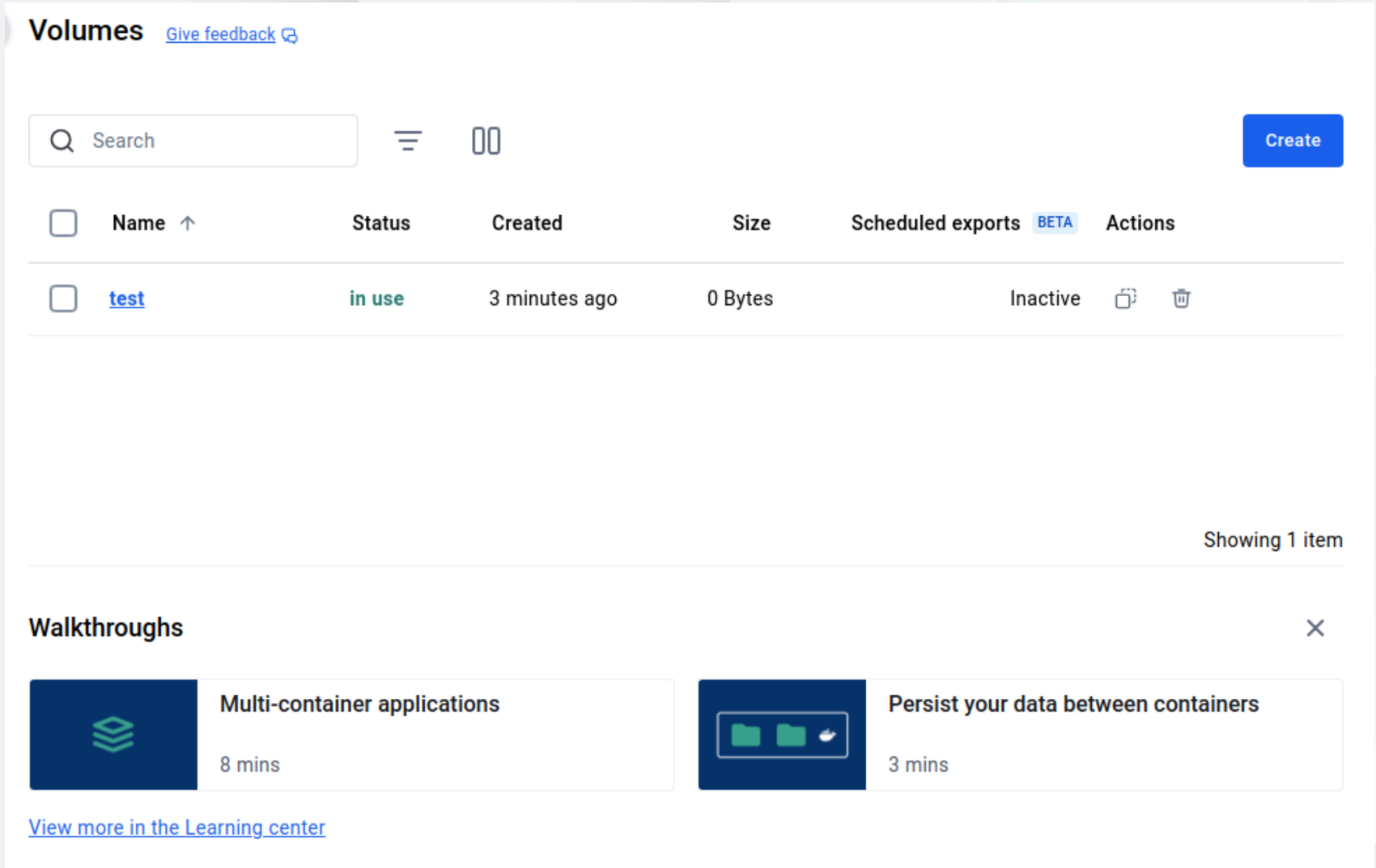


Présentation de Docker Desktop

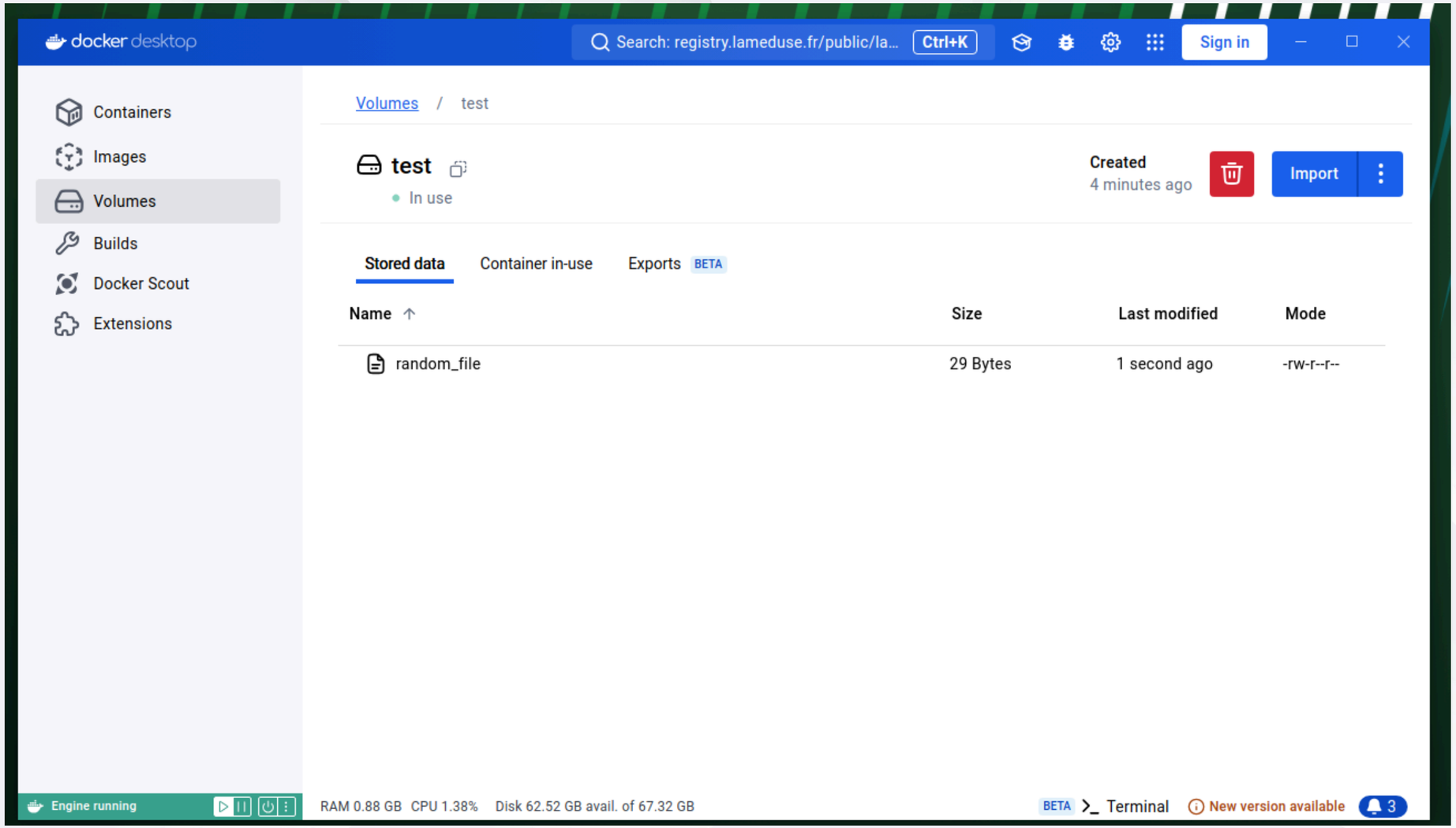


LaMeDuSe

Présentation de Docker Desktop

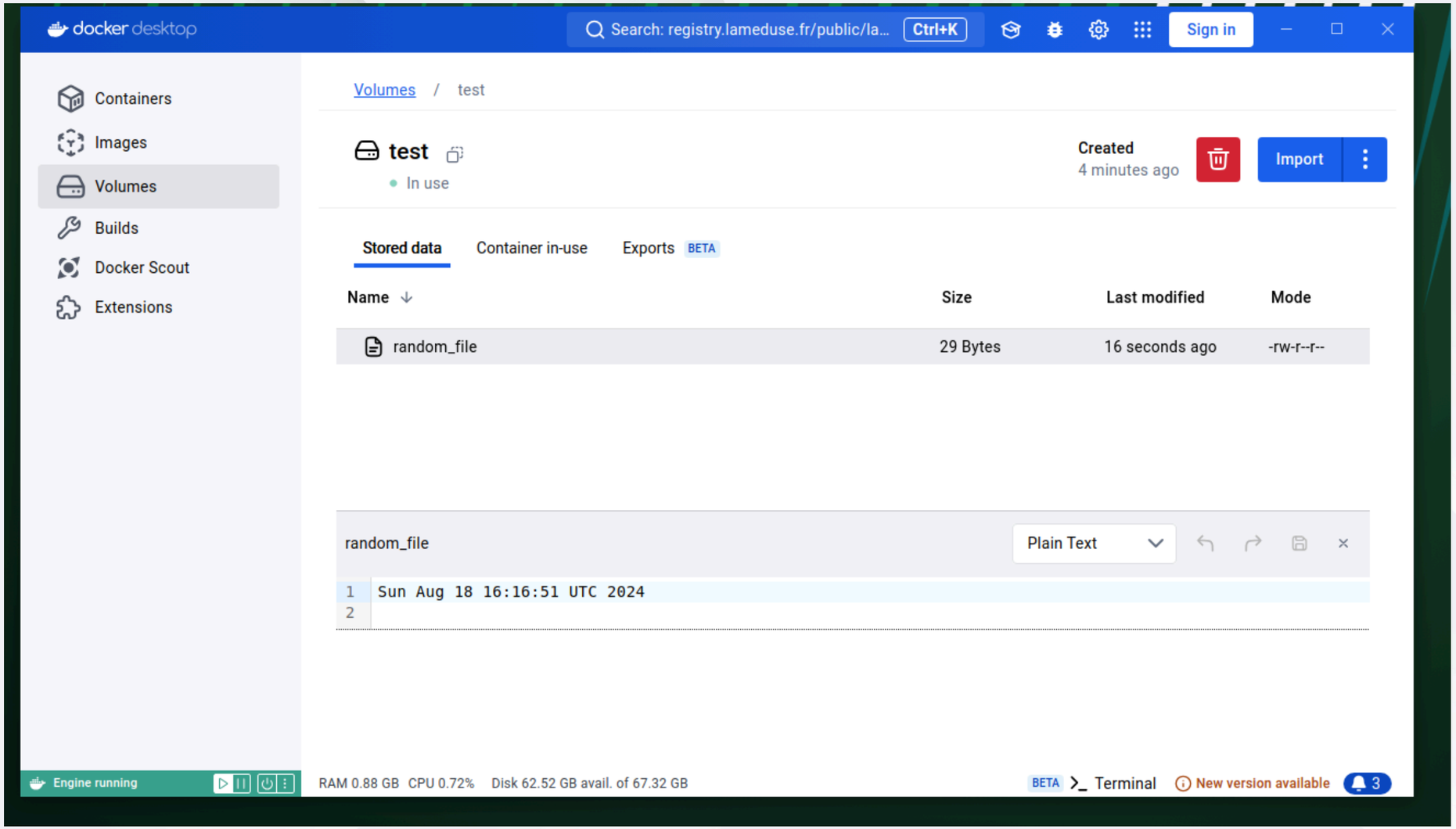


Présentation de Docker Desktop



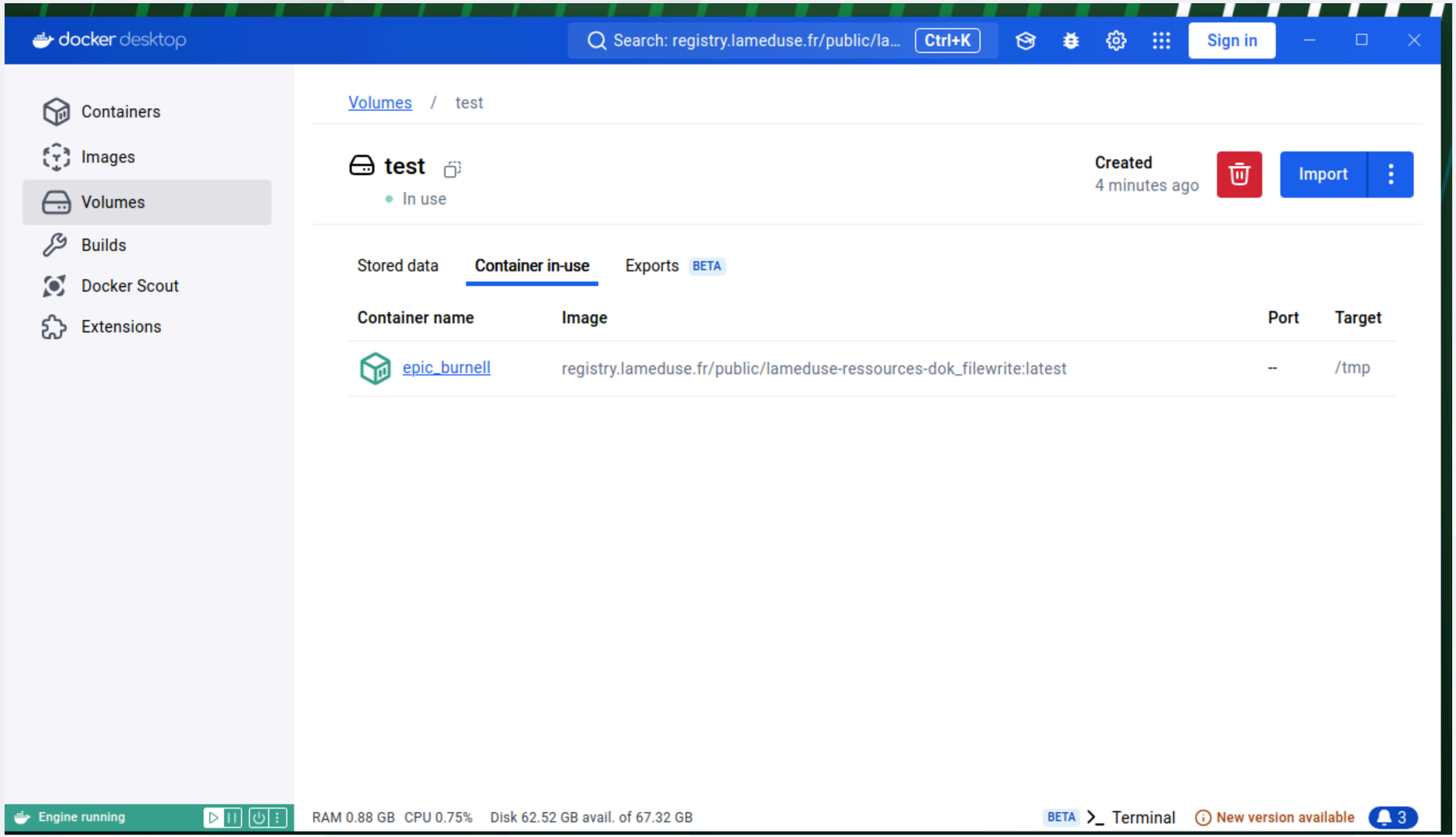
LaMeDuSe

Présentation de Docker Desktop



LaMeDuSe

Présentation de Docker Desktop



LaMeDuSe

L'orchestrateur Swarm : nodes, services, secrets, configs.

Termes

- Node
  - Représente un serveur (noeud)
  - Master
    - Gère le déploiement sur les nœuds worker
    - Initialisation avec “docker swarm init”
  - Worker
    - Exécute le travail donné par le nœud maitre
    - Initialisation avec “docker swarm join <token> <hote>
      - “docker swarm join-token worker” permet de récupérer les informations

L'orchestrateur Swarm : nodes, services, secrets, configs.

Termes

- Service
  - Correspond à un container
  - Les services peuvent être répliqués avec la commande “docker service scale”

L'orchestrateur Swarm : nodes, services, secrets, configs.

Termes

- Secret
  - Création avec la commande “docker secret create”
  - valeur encryptée
- Config
  - Création avec la commande “docker config create”
  - Fichier qui peut être montés dans le container
  - e.g. une configuration Nginx

Déploiement de services et stacks dans un Swarm

- docker service create [option] <image> [commande] [arguments]
  - --replicas <nb_entier> : définie le nombre de réplications du service
- docker stack deploy [option] <nom de la stack>
  - (requis) --compose-file <docker-compose.yaml>

```
warstrolo@dev-xubuntu-01:~/Documents/WORK_DATA/lameduse/lameduse-ressources/dok_docker_compose_wordpress$ docker stack deploy --compose-file ./docker-compose.yaml wordpress
Ignoring unsupported options: restart

Ignoring deprecated options:

expose: Exposing ports is unnecessary - services on the same network can access each other's containers on any port.

Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network wordpress_default
Creating service wordpress_db
Creating service wordpress_wordpress
warstrolo@dev-xubuntu-01:~/Documents/WORK_DATA/lameduse/lameduse-ressources/dok_docker_compose_wordpress$ S
```

Reverse-proxy et load-balancer pour Web Services en cluster

Déploiement de Traefik avec docker swarm

```
version: '3.8'
services:
  traefik:
    image: traefik:v2.10
    ports:
      - "80:80"
      - "8080:8080"  # Dashboard
    command:
      - "--api.insecure=true"  # Enable dashboard at /:8080
      - "--providers.docker=true"
      - "--entrypoints.web.address=:80"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
    deploy:
      replicas: 1
```

Reverse-proxy et load-balancer pour Web Services en cluster

Déploiement de l’application

```
version: '3.8'
services:
  web:
    image: nginx:alpine
    deploy:
      replicas: 3
      labels:
        - "traefik.http.routers.web.rule=Host(`example.com`)"
        - "traefik.http.services.web.loadbalancer.server.port=80"
```

TP : Création d'un cluster Swarm.

TP : Gestion des nœuds dans le cluster.

TP : Publication et mise à l'échelle d'un service dans le cluster.

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Ressources des TP

1. TP : Créer une machine virtuelle pour réaliser un maquettage
2. TP : Configurer un conteneur en ligne de commande
3. TP : Créer un conteneur personnalisé
4. TP : Mettre en œuvre une application multiconteneur
5. TP : Construire et utiliser son propre registre
6. TP : Administrer les conteneurs
7. TP : Création d'un cluster Swarm.
8. TP : Gestion des nœuds dans le cluster.
9. TP : Publication et mise à l'échelle d'un service dans le cluster.

1- TP: Créer une machine virtuelle pour réaliser un maquettage

Sujet 1 : Mise en place d'une machine virtuelle pour accueillir docker

1. Création de la VM sous VMware Workstation
    a. Utiliser l'ISO Alma Linux
    b. Allouer 20 go de disque
2. Paramétrage de la machine virtuelle
    a. Allouer 4 go de ram
    b. Activer la virtualisation Intel VT-x / AMD-V
3. Installation de Docker
    a. Télécharger le script d'installation
    b. Démarer et activer le service docker

2- TP: Configurer un conteneur en ligne de commande

Sujet 1 : Exécuter les deux containers (filewriter & filereader)

1. Création du volume partagé
2. Exécution du container fileread
3. Exécution du container filewrite

Notes :

- registry.lameduse.fr/public/lameduse-ressources-dok_filewrite
- registry.lameduse.fr/public/lameduse-ressources-dok_fileread

2- TP: Configurer un conteneur en ligne de commande

Sujet 1bis : Exécuter les deux containers (netwriter & netreader)

1. Création du réseau partagé
2. Exécution du container netread
3. Exécution du container netwrite

Notes :

- registry.lameduse.fr/public/lameduse-ressources-dok_netwrite
- registry.lameduse.fr/public/lameduse-ressources-dok_netread

2- TP: Configurer un conteneur en ligne de commande

Sujet 2 : Exécuter nginxdemos/hello

Objectifs

- Le container doit être accessible sur http://localhost:9000
- Port interne du container : 80

2- TP: Configurer un conteneur en ligne de commande

Sujet 3 : Exécuter nginxdemos/hello (avancé)

Objectifs

- Le container doit être accessible sur http://localhost:9000
- Interdiction d'utiliser la publication de port "-p"

2- TP : Configurer un conteneur en ligne de commande

Sujet 3 Solution : Exécuter nginxdemos/hello (avancé)

Objectifs

- Le container doit être accessible sur http://localhost:9000
- Interdiction d'utiliser la publication de port "-p"

Solutions

- Solution 1 :
  - L'option --network contient des réseaux prédéfinis comme "host" qui permet au container d'utiliser directement le réseau de l'hôte
- Solution 2 :
  - Utilisation de l'option -P pour automatiquement publier les ports exposé

3- TP : Créer un conteneur personnalisé

Sujet 1 : Créer un container d'une application ReactJS

Objectifs

- Cloner le repo git (https://github.com/LaMeDuSe/lameduse-ressources.git)
- Aller dans le répertoire "dok_cra_example"
- Créer le fichier Dockerfile (Ou utiliser celui existant)
  - Lire le readme dans le répertoire (il contient des informations pour le faire)
- Construire l'image sous le nom de "application_react" avec le tag "demo"
- Lancer le container et publier le port 8080 (port interne : 80)

3- TP : Créer un conteneur personnalisé

Sujet 2 : Créer le Dockerfile d'une application Reactjs

Objectifs

- Créer le fichier Dockerfile du sujet 1
  - image de build : "node:18"
  - image d'exécution : "nginx:alpine"
  - COPY --from=nom_image <source> <destination> : copie des fichiers d'une image à une autre
  - Commande nginx : "nginx -g daemon off;"

4- TP : Mettre en œuvre une application multiconteneur

Sujet 1 : Créer un docker compose pour wordpress

Notes

- variable pour la base de donnée
  - MYSQL_ROOT_PASSWORD=somewordpress
  - MYSQL_DATABASE=wordpress
  - MYSQL_USER=wordpress
  - MYSQL_PASSWORD=wordpress
- variable pour le serveur wordpress
  - WORDPRESS_DB_HOST=db
  - WORDPRESS_DB_USER=wordpress
  - WORDPRESS_DB_PASSWORD=wordpress
  - WORDPRESS_DB_NAME=wordpress
- Images
  - mariadb:10.6.4-focal
  - wordpress:6.6.1-php8.2-apache

4- TP : Mettre en œuvre une application multiconteneur

Sujet 1 (solution) : Créer un docker compose pour wordpress

Solution :

- https://github.com/LaMeDuSe/lameduse-ressources/tree/main/dok_docker_compose_wordpress

5- TP : Construire et utiliser son propre registre

Sujet 1 : Mettre en place son propre registre

Objectif :

- Exécuter l’image registry:2
- Publier le port 5000
- Construire et envoyer une image sur le registre

6- TP : Administrer les conteneurs

Sujet 1 : Mettre en place un environnement wordpress en utilisant un bindmount

Objectif :

- Créer le docker-compose
- Utiliser un bind-mount pour stocker les données de la base de donnée
- Utiliser la commande docker compose down -v

6- TP : Administrer les conteneurs

Sujet 2 : Déployer fileread et filewrite avec Docker Desktop

Objectif :

- Re-déployer Fileread et Filewrite en utilisant Docker Desktop

6- TP : Administrer les conteneurs

Sujet 3 : Déployer netread et netwrite avec Docker Desktop

Objectif :

- Re-déployer netread et netwrite en utilisant Docker Desktop

7- TP : Création d'un cluster Swarm.

Sujet 1 : Créer un cluster Docker Swarm

Objectif :

- Créer un nœud maitre docker swarm
- Ajouter un nœud de travail
- Profiter de votre nouveau cluster !

7- TP : Création d'un cluster Swarm.

Sujet 1 (solution) : Créer un cluster Docker Swarm

Objectif :

- Créer un nœud maitre docker swarm
  - docker swarm init
- Ajouter un noeud de travail
  - docker swarm join-token worker
  - (sur le nœud à ajouter) “docker swarm join <token> <hote>”
- Profiter de votre nouveau cluster !

8- TP : Gestion des nœuds dans le cluster

Sujet 1 : Mettre en pause un nœud de travail

Objectif :

- Mettre en pause un nœud avec la commande
  - “docker node update <node>”

8- TP : Gestion des nœuds dans le cluster

Sujet 1 (solution) : Mettre en pause un nœud de travail

Objectif :

- Mettre en pause un nœud avec la commande
  - “docker node update --availability pause <node>”

Note :

- Si vous avez regardé la documentation, vous avez vu les options suivantes
  - Avaliable : nœud disponible, prêt à recevoir de la charge
  - Pause : le nœud ne prend plus de nouveau service
  - Drain : le noeud est vidé de tout les services

9- TP : Publication et mise à l'échelle d'un service dans le cluster

Sujet 1 : Créer un déploiement nginx et lui ajouter 2 répliquas

Objectif :

- Déployer un service nginxdemos/hello
  - publier le port 9000
- Ajouter 2 répliquas au service

Question :

- Avez vous rencontrer une erreur ? Si oui pourquoi ?

9- TP : Publication et mise à l'échelle d'un service dans le cluster

Sujet 1 (solution) : Créer un déploiement nginx et lui ajouter 2 répliquas

Objectif :

- Déployer un service nginxdemos/hello
  - publier le port 9000
  - commande : “docker service create [--name <nom>] <image>”
- Ajouter 2 répliquas au service
  - commande : “docker service scale <nom=NB_replicas>”

Question :

- Avez vous rencontrer une erreur ? Si oui pourquoi ?

0.4- TP : Additions

Sujet 1 : Déployer apache + php

Objectif :

- Déployer apache + php dans une machine virtuelle debian
- Déployer apache + php dans un container docker

```
<!DOCTYPE html>
<html>
 <head>
 <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

0.7- TP : Additions

Sujet 2 : Déployer portainer + une base de donnée

Objectif :

- déployer une instance portainer
- deployer une base de données en utilisant portainer

0.8- TP : Additions

Sujet 3 : Déployer deux instance WordPress avec docker swarm

Objectif :

- déployer une instance swarm
- deployer une instance traefik
- deployer une instance WordPress dont le titre sera blog
- deployer une instance WordPress dont le titre sera site principal
- Configurer l'utilisation de traefik sur les deux instance

paul.millet@lameduse.fr

LaMeDuSe

DOK : Docker, créer et administrer ses conteneurs virtuels d'applications

Sources

Sources :

- https://github.com/containerd/containerd/blob/main/core/runtime/v2/README.md

Q&A

Q: A quoi corresponds le contexte ?
A: Le contexte est l'endroit où se situent tous les fichiers de votre application nécessaires à la construction de votre container.
Le contexte est un répertoire qui souvent contient à sa racine un fichier dockerfile.
Q: Où se situes les volumes ?
A: /var/lib/docker/volumes

Docker Desktop + Ubuntu 24.04

Fix: sudo sysctl -w kernel.apparmor_restrict_unprivileged_userns=0

Instruction Docker avancée

- HEALTHCHECK Check a container's health on startup.
- LABEL Add metadata to an image.
- MAINTAINER Specify the author of an image.
- ONBUILD Specify instructions for when the image is used in a build.
- STOPSIGNAL Specify the system call signal for exiting a container.
- VOLUME Create volume mounts.

Création d'un fichier YAML de configuration (Options)

```yaml
version: '3.8'  # Version du format de fichier Docker Compose.
services:
  app:
    image: nginx:latest  # Spécifie l'image Docker à utiliser pour le conteneur.
    build:
      context: ./app  # Répertoire pour la construction de l'image.
    command: "nginx -g 'daemon off;'"  # Commande à exécuter dans le conteneur.
    environment:
      - DEBUG=true  # Variable d'environnement pour le conteneur.
    ports:
      - "8080:80"  # Mappage des ports (hôte:conteneur).
    volumes:
      - app-data:/usr/share/nginx/html  # Volume monté dans le conteneur.
    networks:
      - webnet  # Réseau auquel se connecter.
    restart: always  # Politique de redémarrage du conteneur.
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost/"]  # Test de la santé du service.
    logging:
      driver: json-file  # Pilote de journalisation.
    user: "1000:1000"  # Utilisateur pour exécuter le conteneur.
    working_dir: /usr/src/app  # Répertoire de travail dans le conteneur.
  db:
    image: postgres:13  # Image Docker pour le service de base de données.
    environment:
      POSTGRES_PASSWORD: example  # Mot de passe pour PostgreSQL.
    volumes:
      - db-data:/var/lib/postgresql/data  # Volume pour les données de la base de données.
    networks:
      - webnet  # Réseau auquel se connecter.
# Définition des volumes utilisés par les services
volumes:
  app-data:  # Volume pour l'application.
  db-data:   # Volume pour la base de données.
# Définition des réseaux personnalisés
networks:
  webnet:  # Réseau pour la communication entre services.
    driver: bridge  # Pilote de réseau.
```