

Mise en production et déploiement - MS2D / ERIS

Christophe Brun

Campus Saint-Michel IT

17 avril 2024



Table des matières

- ① Table des matières
- ② Programme du module
- ③ Introduction
- ④ Le matériel
- ⑤ Le logiciel
- ⑥ Le ressources partagées
- ⑦ Les services
- ⑧ Ansible
- ⑨ Docker

Mise en production et déploiement

Compétence acquise au cours des 5 jours du module

Compétences :

- “Préparer l'environnement et déployer le progiciel ou la solution.”



Mise en Production et déploiement

Le programme officiel des 5 jours du module

① Mise en exploitation des ressources matérielles et logicielles

- Vérification des configurations
- Déploiement des applications
- Automatisation des procédures de déploiement
- Élaborer les bilans de l'exploitation
- Prévoir les évolutions de l'infrastructure

② Indicateurs et mesure de performance – Systèmes / Réseau et web

- Centralisation des journaux et exploitation des logs avec syslogd
- Analyse du trafic réseau avec MRTG
- Analyse des journaux de type d'Apache Web Server avec Analog
- Consolidation d'indicateur de qualité avec rrdtool
- Création de page HTML de type tableau de bord avec rrdtool – Tableau de bord
- Gestion d'incidents et actions correctives

Evaluation

- Bons et mauvais points tout au long du module.
- 80 % x évaluation continue avec les résultats des exercices.
- 20 % sur une évaluation écrite finale

Intervenant sur le module Architecture d'Application

Christophe Brun, conseil en développement informatique

- 1^{ère} année d'intervenant à Saint-Michel 😊.
- 7 ans de conseil en développement au sein d'SSII .
- 7 ans de conseil en développement à mon compte [PapIT](#).
- Passionné !



Les ressources matérielles

Les machines du web

On peut résumer un serveur web à :

- Un CPU (Architectures x86, ARM, RISC-V, etc.)
- De la mémoire RAM
- De l'IO
 - Disque dur (NVMe)
 - Réseau (Ethernet, optique)
 - Disque DVD

A l'image de n'importe quel ordinateur, mais souvent avec plus de capacités et de redondances.

Les ressources logicielles

Les stacks du web

Une stack web est faite en général de :

- Un OS, qui est une couche d'abstraction du hardware présenté précédemment (Ubuntu, Red Hat, Windows, BSD).
- Un reverse Proxy (Apache, Nginx, etc.)
- Un serveur web (Apache, Nginx, Gunicorn, NextJS, etc.)
- L'application web un ensemble de “Business rules” implémentées grâce à des bases de données, des IHM, des connections aux API (en PHP, Python, NodeJS, etc.)
- Une base de données (MySQL, PostgreSQL, MongoDB, etc.)

Les ressources partagées

Les machines du web

Dans un environnement de production, on retrouve une forte variabilité des ressources CPU mais surtout des ressources logicielles.

A cette variabilité des environnements de production, il faut ajouter la variabilité des environnements de développement.

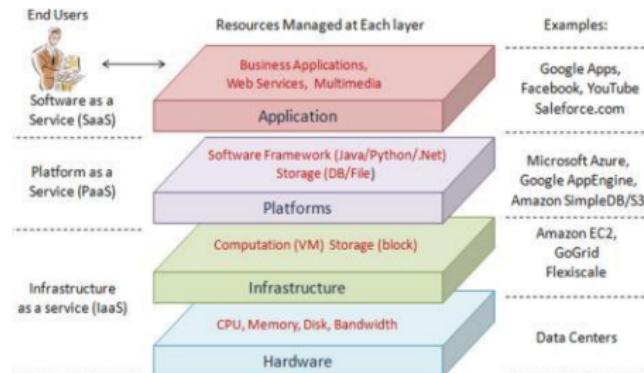
Beaucoup de développeurs travaillent sous Windows ou Mac OS qui sont rarement des environnements de production. Les capacités hardware sont souvent très différentes, la production a plus de capacité que le développement en général.

C'est cette variabilité des environnements qui est la difficulté majeur du déploiement et de la mise en production.

Les ressources partagées

Les environnements cloud

Pour palier à cette variabilité, diverses architectures en nuage ont vu le jour. Certaines ont abstraient le hardware voir même une partie du software pour ne laisser que l'application.



IAAS, PAAS et SAAS

On voit que SAAS et Paas ont fait abstraction du hardware¹.

¹Patel et al., https://www.researchgate.net/publication/324692035_Arcus_Cloud_A_Private_Cloud_Establishment

Les ressources partagées

Les environnements cloud

Abstraction du hardware ne veut plus dire qu'on a plus du tout besoin de s'en soucier. Même dans le cloud public il faut adapter les capacités des ressources aux besoins et aux coûts.

Essential	
Nombr e de nœuds	1
RAM par nœud	De 4 à 30 Go
Stockage total utile	De 80 à 640 Go
Engagement de niveau de service (SLA)	Non
Passage à une offre supérieure en un clic	Oui
Chiffrement des données au repos et en transit (SSL)	Oui
Graphique de performance	Oui
Sauvegarde en temps réel vers une localisation distante	Oui
Période de rétention des sauvegardes	2 jours
Point de restauration antérieur (Point in Time Recovery)	Oui
Database forking	Oui
Connection pooling	Non
Support de Terraform (en savoir plus)	Oui
Connexivité au réseau privé (vRack)	Oui
Haute-disponibilité	Non
Accès en lecture seule aux nœuds de replication	

Resource Allocation
Select an initial resource allocation preset, or customize your deployments resources

Templates

Small	Medium	
RAM	Disk	IOPS
12GB	120GB	1200
Cores	Members	
3	3	

MySQL sur le cloud public IBM

MySQL sur le cloud public OVH

Qu'est-ce qui vous étonne dans ces deux captures d'écran ?

Les ressources partagées

Les environnements cloud

Le cloud public donne souvent l'impression que les ressources sont infinies, mais c'est faux.

Ils ne communiquent même pas les métriques requises pour comprendre les capacités de leur cloud.

Le cloud privé à l'avantage de présenter de manière plus claire les ressources hardware.

Les ressources partagées

Les environnements cloud

Même dans le cloud privé, les vCPU ne sont pas des CPU physiques, mais des CPU partagés.

Ce n'est indiqué nul part, mais c'est probablement le cas chez OVH . Cela implique que OS qui héberge l'hyperviseur équilibre en permanence les ressources CPU entre les VM . Cela a un coût en performance qui est inconnu.

Techniquement ce n'est pas obligatoire de partager les CPU, mais c'est souvent le cas pour des raisons de scalabilité. Au besoin un coeur physique peut être dédié à une seule VM .

Linode, un cloud provider, apporte des informations claires sur ce sujet
<https://www.linode.com/docs/products/compute/compute-instances/plans/comparing-shared-and-dedicated-cpus/>.

Les ressources partagées

La virtualisation

Développée par IBM dès les années 60, la virtualisation permet de créer plusieurs machines virtuelles sur un seul hardware. Elle fut d'abord commercialement disponible sur les mainframes avant les plus petites plateformes AS400, iSeries et Power. La technologie TIMI d'IBM permet depuis les années 80 de faire tourner des programmes qui restent compatibles avec des hardwares de plus en plus puissants (changement d'ISA, d'*endianness* A.K.A boutisme en français).

Cela permet de mutualiser les ressources hardware, de les partager en fonction des besoins et même de changer de hardware.

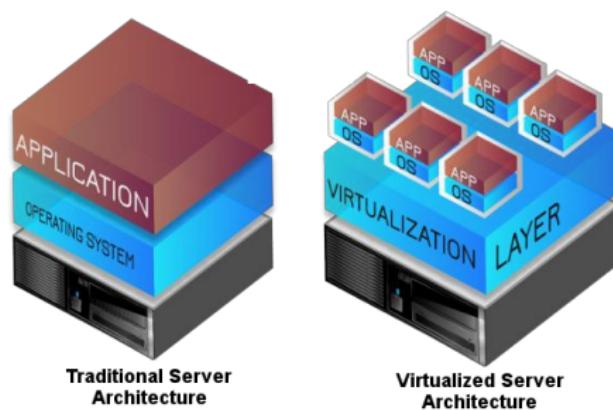
Elle apporte un gain de sécurité en isolant les VM les unes des autres.

Quid des gros bugs /failles de sécurité comme Meltdown et Spectre ?

Les ressources partagées

La virtualisation¹

Aujourd’hui la virtualisation est principalement utilisée pour le partage des ressources et isoler les applications dans une VM pour raison de sécurité. Un OS hôte fait tourner un hyperviseur qui lui fait tourner des VM et plus juste un soft compilé.



Les ressources partagées

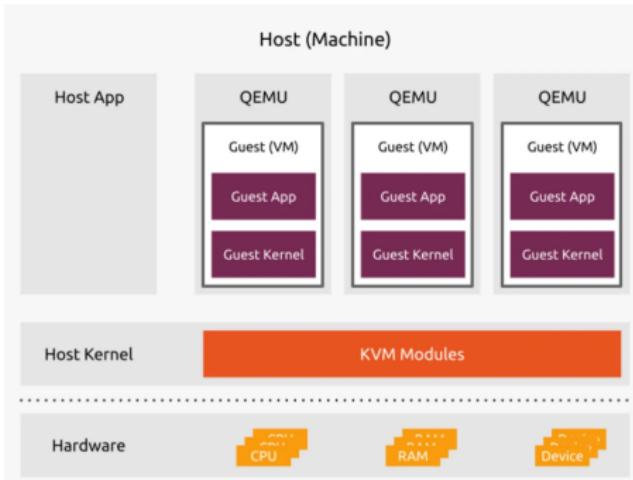
La virtualisation avec KVM et Qemu²

KVM (Kernel-based Virtual Machine) est une technologie de virtualisation ouverte intégrée à Linux.

C'est un hyperviseur de type 1, qui négocie directement avec le hardware et offre des performances proches de la machine hôte. Contrairement à un hyperviseur de type 2, qui doit passer par l'OS hôte pour accéder au hardware mais ces derniers ont donc une meilleure portabilité et compatibilité.

Qemu lui peut faire tourner des VM avec ou sans KVM en fonction des besoins.

²KVM hypervisor: a beginners' guide,
<https://ubuntu.com/blog/kvm-hypervisor>



Les ressources partagées

Exercice, avec ou sans KVM ?

- Un Windows guest sur un Mac OS hôte?
- Un Alpine Linux sur un Raspberry³ ?
- VM Ubuntu sur un Ubuntu hôte?
- VM Ubuntu sur un Windows hôte?



³<https://unix.stackexchange.com/questions/340912/qemu-with-kvm-with-differaing-host-guest-architectures>

Les services

Définition

Dans les systèmes Windows et Linux, le terme service désigne un programme qui s'exécute en arrière-plan, sans intervention de l'utilisateur. Il faut le configurer pour qu'il démarre automatiquement, dans un certain ordre et avec les bonnes permissions.

La configuration principale consiste à définir une ligne de commande qui sera exécutée au démarrage du service.

Avec `systemd`⁴ sous Linux, on peut définir des dépendances entre les services. Limiter les ressources CPU, mémoire, IO, etc. Définir ce qu'il faut faire en cas de crash. Exécuter des scripts avant et après le démarrage du service.

Leur chemin par défaut est `/etc/systemd/system/`.

Ne pas confondre `systemd` qui exécute des tâches en arrière plan et l'ordonnanceur `cron`.

⁴System and Service Manager, <https://systemd.io/>

Les services

Exemples

`systemd` est donc l'outil de choix pour lancer en production des applications, des progiciels, etc.

Encore mieux, on peut configurer des services pour qu'ils lancent des VM avec Qemu et les applications tourneraient dans les VM . On profite des avantages de la virtualisation pour isoler les applications et les sécuriser.

Que fait cette configuration ?

```
[Unit]
Description=IN DATA development application
[Service]
RuntimeMaxSec=3600s
Restart=always
WorkingDirectory=/home/debian/dev/IN FRANCE
ExecStart="/home/debian/dev/IN FRANCE/python3-dev/bin/python3" -m gunicorn -w 1
          -b unix:/tmp/in-france-dev-indata-gunicorn.sock application_indata
[Install]
WantedBy=multi-user.target
```

Le chemin de l'exécutable dans `ExecStart` doit être absolu!

Les services

Les commandes de base

- `systemctl start <service>` pour démarrer un service.
- `systemctl stop <service>` pour arrêter un service.
- `systemctl restart <service>` pour redémarrer un service.
- `systemctl status <service>` pour afficher le statut d'un service.
- `systemctl enable <service>` pour activer un service au démarrage.
- `systemctl disable <service>` pour désactiver un service au démarrage.
- `systemctl list-units --type=service` pour lister les services.
- `systemctl daemon-reload` pour recharger la configuration de tous les services.
- `systemctl reload <service>` pour recharger la configuration d'un service.

Les services

Les commandes pour monitorer

Pour monitorer les logs d'un service, la commande `journalctl` est très utile.

Quelques exemples de commandes utiles :

- `journalctl --unit<service>` pour afficher les logs d'un service.
- `journalctl --unit<service> -f` pour afficher les logs en temps réel.
- `journalctl --unit<service> --since "2024-04-17 00:00:00"` pour afficher les logs depuis une date.
- `journalctl --unit<service> --since "2024-04-17 00:00:00" --until "2024-04-17 23:59:59"` pour afficher les logs entre deux dates.
- `journalctl --unit=<service> -n 100 --no-pager` pour afficher les 100 dernières lignes.

Exercice pratique

Créer une VM avec Qemu et lancer une application avec systemd

Les exigences :

- Créer une VM Linux avec Qemu à l'image d'un VPS OH d'entrée de gamme (1 vCPU, 2 Go de RAM et 10 Go de disque).
- Configurer la machine hôte pour démarrer la VM automatiquement.
- Configurer la VM et Qemu avec une sécurité en accord les bonnes pratiques de sécurité (Clé SSH uniquement, pas de connexion SSH avec le user root, gestion des ports forwardés).
- Démarrer l'application Gunicorn “Hello World”
<https://gunicorn.org/#quickstart> avec systemd.
- Accès à la VM et à l'application depuis la machine hôte.

Exercice pratique

Une des solutions

```
qemu-img create -f qcow2 linux.qcow2 10G # Create a disk image
# Download a minimal Ubuntu ISO
wget http://archive.ubuntu.com/ubuntu/dists/bionic/main/installer-amd64/current/
    images/netboot/mini.iso
# Install the OS using a virtual CDROM
qemu-system-x86_64 -m 2G -smp 1 -nic user -boot d -cdrom mini.iso -hda linux.
    qcow2 -k fr -enable-kvm
# Run the VM to configure and test SSH
qemu-system-x86_64 -m 2G -smp 1 -nic user,hostfwd=tcp::5022-:22,hostfwd=tcp
    ::5080-:80 -display none -hda linux.qcow2 -k fr -enable-kvm
```

Expliquez chacune des lignes de commande ci-dessus.

Ressources utiles :

- Similaire mais avec Alpine Linux
- Page d'installation Ubuntu du “MinimalCD”
- Configuration des login SSH sur Ubuntu

Exercice pratique

Une des solutions

Le service de la machine hôte :

```
[Unit]
Description=Qemu Ubuntu VM for St-Michel classes
After=network.target
StartLimitIntervalSec=0

[Service]
Restart=always
WorkingDirectory=/home/chrichri/Documents/Campus-St-Michel-IT/production-
    deployment
ExecStart=/usr/bin/qemu-system-x86_64 -m 2G -smp 1,maxcpus=1 -nic user,hostfwd=
    tcp::5022-:22,hostfwd=tcp::5080-:8080 -display none -hda linux.qcow2 -k fr
    -enable-kvm

[Install]
WantedBy=multi-user.target
```

Expliquez chaque option de la configuration du service.

Exercice pratique

Une des solutions

Le service de la machine hôte :

```
[Unit]
Description=Qemu Ubuntu VM for St-Michel classes
After=network.target
StartLimitIntervalSec=0

[Service]
Restart=always
WorkingDirectory=/home/chrichri/Documents/Campus-St-Michel-IT/production-
    deployment
ExecStart=/usr/bin/qemu-system-x86_64 -m 2G -smp 1,maxcpus=1 -nic user,hostfwd=
    tcp::5022-:22,hostfwd=tcp::5080-:8080 -display none -hda linux.qcow2 -k fr
    -enable-kvm

[Install]
WantedBy=multi-user.target
```

Expliquez chaque option de la configuration du service.

-enable-kvm si l'hôte est un Linux également.

-display none pour ne pas afficher la console graphique.

Exercice pratique

Une des solutions

Le service dans la VM :

```
[Unit]
Description=St-Michel Hello World
After=network.target
StartLimitIntervalSec=0

[Service]
Restart=always
WorkingDirectory=/home/chrichri
ExecStart=/usr/bin/gunicorn -w 1 hello:app -b 0.0.0.0:8080

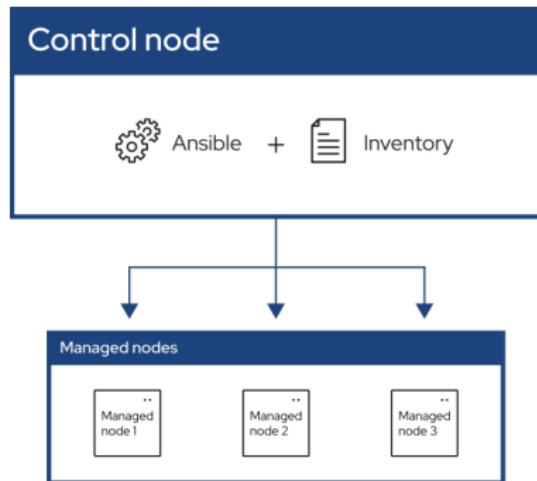
[Install]
WantedBy=multi-user.target
```

Expliquez chaque option de la configuration du service.

Automatisation de la configuration avec Ansible

Définition⁵

Ansible est un outil d’automatisation de la configuration des machines physiques ou virtuelles. Il permet de définir des “playbooks” qui décrivent une configuration à appliquer aux machines qui sont dans “l’inventory” .



⁵Ansible Community Documentation,

https://docs.ansible.com/ansible/latest/getting_started/index.html

Automatisation de la configuration avec Ansible

Example d'inventaire

Pour accéder à la VM créée précédemment dans Qemu, il faut définir un inventaire avec les données dont on aurait besoin pour y accéder par SSH .

```
$ ssh chrichri@localhost -p 5022 -v -i /home/chrichri/Documents/Campus-St-Michel-IT/production-deployment/virt-ubuntu ansible_ssh_user=chrichri
```

Devient dans l'inventaire (on peut créer des variables) :

```
[gunicorn]
localhost:5022 ansible_ssh_private_key_file=/home/chrichri/Documents/Campus-St-Michel-IT/production-deployment/virt-ubuntu ansible_ssh_user=chrichri
```

On utilise le module ping pour vérifier que l'accès SSH est correctement configuré.

```
$ ansible gunicorn -m ping -i inventory.ini
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Automatisation de la configuration avec Ansible

Aucune installation côté client ❤️

Pourquoi Ansible n'a aucun client sur les machines de l'inventaire ?



Automatisation de la configuration avec Ansible

Aucune installation côté client ❤️

Pourquoi Ansible n'a aucun client sur les machines de l'inventaire ?



Avec le seul accès SSH, il exécute les commandes qu'il faut. Il est agnostique à l'OS !

Automatisation de la configuration avec Ansible

Example de “playbook”

Qu'a-t-on installé sur un Ubuntu *base server* comme package(s) pour pouvoir exécuter l'application gunicorn Hello World ?

Automatisation de la configuration avec Ansible

Example de “playbook”

Qu'a-t-on installé sur un Ubuntu *base server* comme package(s) pour pouvoir exécuter l'application gunicorn Hello World ?

```
sudo apt-get install python3-gunicorn
```

Qui devient dans un playbook Ansible `gunicorn.yml` :

```
---
- name: gunicorn
  hosts: gunicorn
  become: yes # Run as root
  tasks:
    - name: Install gunicorn
      apt:
        name: gunicorn # Le package à installer
        state: present
```

À lancer avec la commande :

```
$ ansible-playbook -i inventory.ini gunicorn.yml
```

Automatisation de la configuration avec Ansible

Example de “playbook”

Le résultat de l'exécution du playbook `gunicorn.yml` :

```
$ ansible-playbook -i inventory.ini gunicorn.yml
...
localhost : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

$ ansible-playbook -i inventory.ini gunicorn.yml
...
localhost : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

A la première exécution, le package est installé, à la seconde il est déjà installé, aucun changement.

Automatisation de la configuration avec Ansible

La Ansible Galaxy

Comme souvent, inutile de tout recoder. Ansible est modulaire grâce aux “roles” et aux “playbooks” qui sont aussi des modules qui les composent⁶.

La communauté Ansible a packagé des modules et des playbooks pour les tâches les plus courantes⁷. Ils sont disponibles sur la [Ansible Galaxy](#). Et peuvent être installé avec la commande `ansible-galaxy install <module>`, commande issue du package `ansible-core`.

⁶Roles, https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html

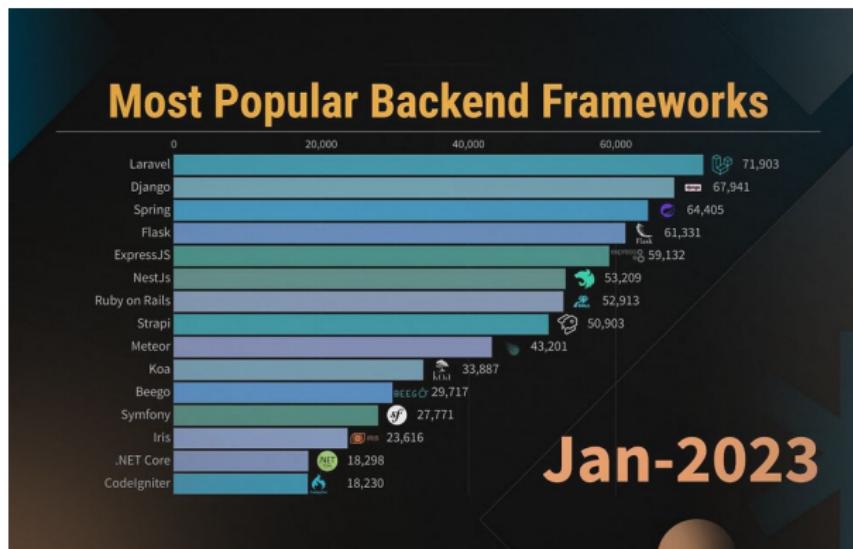
⁷Ansible Galaxy, <https://galaxy.ansible.com/ui/>

Automatisation de la configuration avec Ansible

Exercice

En utilisant la VM créée à l'exercice précédent, configurer un inventaire et un playbook pour installer l'application Spring Boot.

Vous pouvez vous aider du [tutoriel officiel](#).



Automatisation de la configuration avec Ansible

Solution à l'exercice

De nombreuses solutions sont valides. Mais avec Java il faut profiter du “Write once, run anywhere”.

Ce n'est donc pas obligé d'utiliser un système de build comme Maven ou Gradle sur la VM, Java suffit pour exécuter une JAR .

Une JAR (Java ARchive) est un exécutable qui contient toutes les classes zippées qui se lance avec `java -jar <JAR file>`.

Par exemple Java 17 :

```
---
- hosts: gunicorn
  become: yes
  tasks:
    - name: Install Java 17
      apt:
        name: openjdk-17-jdk
        state: present
```

Conclusion sur la virtualisation d'application dans un OS

- On sait créer et configurer une VM avec Qemu.
- On sait créer un service avec systemd pour automatiser des tâches dans la machine hôte ou la VM .
- On sait automatiser la configuration des VMs avec Ansible.

Dans la plus part des clouds privés comme OVH on peut uploader des images de VMs et les lancer.

Ces technologies sont donc compatibles avec le cloud et le on-premise.
On a donc un déploiement sécurisé et agnostique à une infrastructure.

Docker

Définition

“Accelerate how you build, share, and run applications”⁸.

C'est un outil qui permet de créer des conteneurs qui sont des environnements isolés pour exécuter des applications.

Cet environnement portable est défini dans la `Dockerfile`.

Un autre fichier de configuration le `docker-compose.yml` permet de définir comment un ou plusieurs conteneurs communiquent entre eux et avec la machine.



9

Il utilise des outils de la machine, du kernel, `chroot` et des bibliothèques de la machine hôte pour fonctionner mais ne fait pas tourner un autre OS contrairement à la virtualisation.

⁸What is Docker, <https://www.docker.com/>

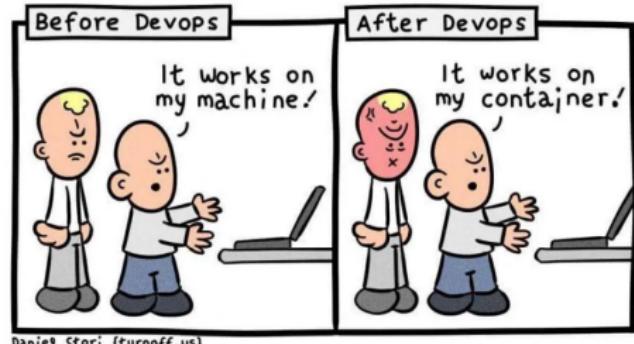
⁹Containers vs. virtual machines, <https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms>

Docker

Définition

Il permet en théorie le déploiement de n'importe quelle application sur n'importe quelle architecture (OS, ISA)¹⁰ :

- ARMv6 32-bit (arm32v6)
- ARMv7 32-bit (arm32v7)
- ARMv8 64-bit (arm64v8)
- Linux x86-64 (amd64)
- Windows x86-64
(windows-amd64)
- ARMv5 32-bit (arm32v5)
- IBM POWER LE (ppc64le)
- IBM z Systems (s390x)
- MIPS64 LE (mips64le)
- RISC-V 64-bit (riscv64)
- x86/i686 (i386)



Daniel Storii (turnoff.us)

¹⁰Docker official Images, <https://github.com/docker-library/official-images#architectures-other-than-amd64>

Docker, créer une image

Définir l'image dans un Dockerfile

L'essentiel de la Dockerfile :

- Le nom de fichier par convention est `Dockerfile`.
- Il vient surcharger une image de base définit dans `FROM`.
- On peut y définir des variables d'environnement avec `ENV`.
- La commande `RUN` permet d'exécuter des commandes dans l'image.
Comme par exemple pour ajouter des dépendances dans l'étape de build de l'image.
- La commande `CMD` définit la commande qui sera exécutée au démarrage du conteneur.
- La commande `EXPOSE` définit les ports qui seront exposés par le conteneur.
- La commande `COPY` permet de copier des fichiers dans l'image.
- La commande `ADD` permet de copier des fichiers dans l'image et de les décompresser.
- *Many more...*¹¹

¹¹Dockerfile reference, <https://docs.docker.com/reference/dockerfile/>

Docker, créer une image

Définir l'image dans un Dockerfile

Que fait cette Dockerfile ?

```
FROM mysql:8-debian

ENV RNCS_PATH=/root/rncs

COPY ./ ${RNCS_PATH}
WORKDIR ${RNCS_PATH}

RUN apt-get update && \
    apt-get install -y jq python3 python3-pip moreutils wget unzip libcairo2-dev && \
    apt-get clean && \
    apt-get autoclean && \
    apt-get autoremove -y && \
    python3 -m pip install --no-cache-dir -r requirements.txt && \
    python3 -m pip install --no-cache-dir -r dev-requirements.txt

EXPOSE 3000/tcp
EXPOSE 5000/tcp
```

Dans la vraie vie tout est très clair grâce à des commentaires explicites 😊.

Le Docker hub

Docker Hub is the world's easiest way to create, manage, and deliver your team's container applications

Encore une fois, inutile de tout recoder. On vient en premier trouver une image la plus complète possible pour son application.

Il probablement une image pour quasi toutes les stacks existantes.

En terme de cybersécurité, veillez à auditer les développeurs des images pour ne pas subir une *supply chain attack*.

Une des bonnes pratiques est d'utiliser des images officielles. Par exemple pour Tensorflow, le développeur est Google, etc.

Concernant les performances les images basées sur Alpine Linux sont souvent plus légères développées autour de `musl libc` et `busybox`.

Les commandes Docker

Builder, lancer et stopper une image

Ci-dessous, quelques commandes Docker de base :

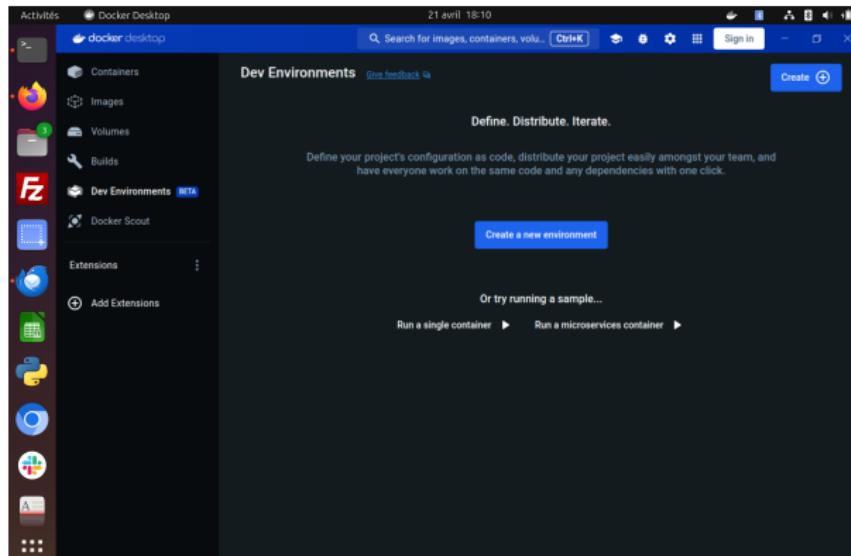
- `docker build -t <image-name>` . pour construire une image Docker.
- `docker run <image-name>` pour lancer une image Docker.
- `docker stop <container-id>` pour arrêter un conteneur.
- `docker ps` pour lister les conteneurs en cours d'exécution.
- `docker ps -a` pour lister tous les conteneurs.
- `docker images` pour lister les images.
- `docker rmi <image-id>` pour supprimer une image.
- `docker rm <container-id>` pour supprimer un conteneur.
- `docker exec -it <container-id> /bin/bash` pour accéder à un conteneur.

Docker, créer une image

Installation d'un environnement de développement Docker

Comme le dit le site Docker, *Build, Share, Run, verify.* Tout cela est possible avec Docker Desktop. Un GUI qui centralise la plupart des fonctionnalités de Docker.

Sur la machine de production le Docker Engine suffit.



Docker, créer une image

Exercice

Développer une image Docker pour une application Spring Boot, la même qu'à l'exercice précédent.

Pensez à optimiser l'image pour qu'elle soit la plus légère possible et le plus sécurisé possible...



Docker, trouver la bonne image

Exercice

Sur le Docker Hub les images pour les applications suivantes:

- Une application NodeJS .
- Une application Flask.
- OpenJDK 17.
- Alpine Linux

Docker compose

Le fichier de configuration Docker Compose¹²

Le fichier `docker-compose.yml` permet de définir comment les conteneurs communiquent entre eux et avec la machine. Limiter les ressources CPU, mémoire prise sur la machine.

Pour les applications web par exemple, il faut définir les ports qui seront exposés par les conteneurs :

```
services:  
  webapp:  
    image: examples/web  
    ports:  
      - "8000:8000"  
    volumes:  
      - "/data"
```

On y définit aussi le(s) Dockerfile(s) :

```
services:  
  client:  
    build:  
      context: ../  
      dockerfile: docker/Dockerfile.production
```

Docker compose

Le fichier de configuration Docker Compose¹³

Les commandes pour builder, lancer et stopper les conteneurs définis dans le fichier `docker-compose.yml` :

- `docker compose build` pour construire les images.
- `docker compose up` pour lancer les conteneurs.
- `docker compose down` pour arrêter les conteneurs.
- `docker compose ps` pour lister les conteneurs en cours d'exécution.
- `docker compose ps -a` pour lister tous les conteneurs.
- `docker compose images` pour lister les images.
- `docker compose exec <container-name> /bin/bash` pour accéder à un conteneur.

Elles sont très similaires aux commandes Docker.

¹³ docker compose, <https://docs.docker.com/reference/cli/docker/compose/>

Docker compose

Exercice, développer une configuration Docker Compose

Un Docker compose pour un site Wordpress, ce dernier a besoin d'une base de données MySQL .

Docker compose

Exercice, développer une configuration Docker Compose

Un Docker compose pour un site Wordpress, ce dernier a besoin d'une base de données MySQL .

```
version: "3.9"
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```