

# Informatique pour l'entreprise

## Machines virtuelles et conteneurs

Marie Pelleau & Olivier Baldellon

`marie.pelleau@univ-cotedazur.fr,`  
`olivier.baldellon@univ-cotedazur.fr`

6 février 2023

- 1 Introduction
- 2 Virtualisation
- 3 Conteneurs
- 4 Les actions github
- 5 Références

# Motivation

## Situation

- On a écrit un logiciel ou programme particulier.
    - Son code nécessite une installation de certains logiciels ou bibliothèques.
    - Éventuellement, ce code fonctionne uniquement
      - avec seulement certaines versions des bibliothèques,
      - voire uniquement sous un système d'exploitation précis.
  - On a envie d'exécuter du code de façon isolée (et sécurisée).
  - On veut automatiser ces traitements.
- 
- Solution 1 : utiliser une **machine virtuelle** pour exécuter le code.
  - Solution 2 : utiliser des **conteneurs**.
  - Solution 3 : on utilise ces **actions** dans github pour automatiser les traitements.

- 1 Introduction
- 2 Virtualisation**
- 3 Conteneurs
- 4 Les actions github
- 5 Références

# Virtualisation : les principes

## Principe général

- Créer une **version virtuelle** d'un système, d'une machine.

## Vocabulaire

- Hôte (*host*) : le système / la machine qui exécute le système de virtualisation.
- Invité (*guest*) : le système / la machine virtualisée.

# Types de virtualisation

## Différents types de virtualisation

- Émulation,
- Hyperviseur,
- Environnement virtuel,
- Virtualisation matérielle,
- Virtualisation complète.

# Émulation

- Un **ordinateur virtuel simulé complet** (bios, processeur, mémoire, disque dur, cartes réseau, vidéo, ...),
- intercepte les instructions du système invité pour les remplacer par leur équivalent sur le système hôte.

## En pratique

- Permet d'exécuter des applications prévues pour d'autres architectures (ordinateurs, consoles, bornes d'arcade...),
- **performances médiocres**,
- le système invité n'a pas conscience de la virtualisation.
- Exemples : DOSBox, MAME, NES Mini.

# Hyperviseur

- Le système invité doit être écrit de manière particulière afin de fonctionner au sein de l'hyperviseur (pilotes et API spécifiques)
- fonctionne directement sans interception des instructions.

## En pratique

- nécessite un système invité compatible avec l'hyperviseur
- excellentes performances (proche de l'exécution native)
- le système invité a conscience de la virtualisation.
- Exemples : Xen, Microsoft Hyper-V, les hyperviseurs des machines virtuelles.



# Virtualisation matérielle

- Le support de la virtualisation peut être intégré au processeur ou assisté par celui-ci
- le matériel peut virtualiser les accès mémoire ou protéger le processeur physique des accès de plus bas niveau
- permet de simplifier la virtualisation logicielle et de réduire la dégradation

## En pratique

- instructions processeur : dépend du processeur.
- Exemples : VM/CMS, AMD-V, Intel VT

# Virtualisation complète : la machine virtuelle

- émule un ordinateur virtuel
- utilise hyperviseur et virtualisation matérielle

## En pratique

- permet d'exécuter des applications prévues pour la même architecture
- assez bonnes performances,
- le système invité n'a pas conscience de la virtualisation.

# Couche de compatibilité

## Définition


- Interface qui permet aux binaires d'un système hérité ou étranger de s'exécuter sur un système hôte

## Exemples

- Wine : pour exécuter les programmes Windows sous Unix.
- Cygwin : collection de logiciels pour simuler un environnement Unix sous Windows.
- WSL (*Windows Subsystem for Linux*) : pour exécuter des programmes Linux sous Windows
  - dans sa version 1 (pas de noyau linux),
  - la version 2 utilise un hyperviseur.

# Logiciels de machines virtuelles

## Quelques exemples

- VMware Workstation : propriétaire édité par VMWare (partie gratuite et parties payantes),
  - VirtualBox : libre et gratuit édité par Oracle (avec des extensions payantes pour un usage professionnel),
  - QEMU : libre et gratuit édité par QEMU team,
  - Et bien d'autres...
- 
- Il faut alors
    - créer une machine virtuelle (quel matériel simuler ?)
    - installer un système d'exploitation (télécharger une image iso)
  -  WSL peut provoquer certaines incompatibilités avec les logiciels de machines virtuelles.

# Exemple : VirtualBox

## VirtualBox

Oracle VM VirtualBox - Gestionnaire de machines

Fichier Machine Aide

Outils

tpsyst1 Éteinte

Slitaz Éteinte

Debian32 Éteinte

Nouvelle Configuration Outils Démarrer

### Général

Nom : tpsyst1  
Système d'exploitation : Linux 2.6 / 3.x / 4.x (64-bit)

### System

Mémoire vive : 3000 Mo  
Ordre d'amorçage : Disquette, Optique, Disque dur  
Accélération : VT-x/AMD-V, Pagnation imbriquée, Paravirtualisation KVM

### Affichage

Mémoire vidéo : 16 Mo  
Contrôleur graphique : VMSVGA  
Serveur de bureau à distance : Désactivé  
Enregistrement : Désactivé

### Stockage

Contrôleur : IDE  
Maître secondaire IDE : [Lecteur optique] Vide  
Contrôleur : SATA  
Port SATA 0 : tpsyst1-disk001.vdi (Normal, 10,00 Gio)

### Audio

Plote hôte : Windows DirectSound  
Contrôleur : ICH AC97

### Réseau

Interface 1: Intel PRO/1000 MT Desktop (NAT)

### USB

Contrôleur USB : OHCI  
Filtres de périphérique : 0 (0 actif)


### Dossiers partagés

Aucun

### Description

Machine virtuelle pour les TP de syst1  
Utilisateur : etudiant  
mdp : syst1\_2020  
utilisateur : administrateur  
mdp : tpsyst1\_2020

### Prévisualisation



# Machines virtuelles : concrètement

## Fichiers créés pour une machine virtuelle

- Un **fichier xml** (un fichier texte structuré par des balises) : définition de la machine virtuelle.
- Un **fichier disque** (fichier binaire .vdi, .vmdk par exemple) : définition du disque virtuel.
- Un **fichier journal** (fichier texte .log) : évènements relatifs à l'exécution de la machine virtuelle.

## Partager ou publier une machine virtuelle

- Il suffit de partager ces fichiers.
- Mais les fichiers disques sont **énormes** (plusieurs Go).

- 1 Introduction
- 2 Virtualisation
- 3 Conteneurs**
  - L'API REST de docker
- 4 Les actions github
- 5 Références

# Conteneurs : principe général

## Sur les systèmes « normaux »

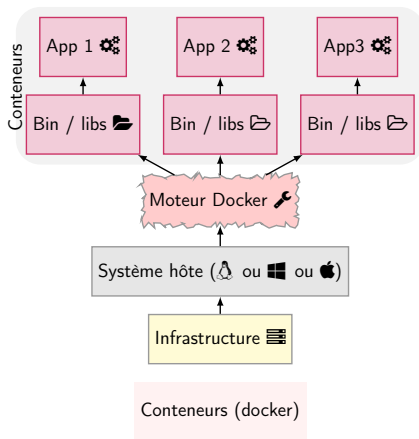
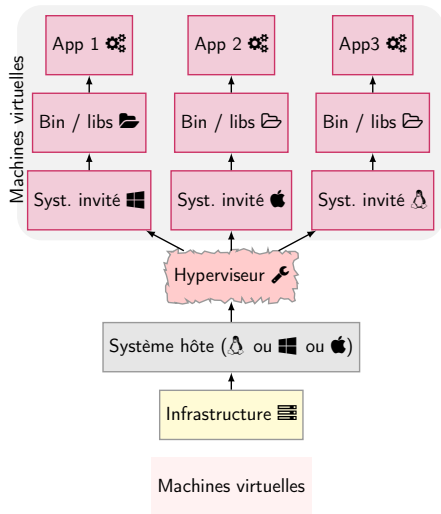
- Un programme peut voir (sans avoir forcément le droit de l'utiliser) les ressources
  - **matérielles** (microprocesseur, connexion réseau..),
  - **de données**, qui peuvent être lues ou écrites (fichiers, répertoires),
  - **périphériques**, avec lequel il peut interagir (webcam, imprimante...)
- Système d'exploitation : autorise ou interdit à l'utilisateur d'utiliser ces ressources.

## Conteneurisation

- Programme exécuté dans des conteneurs, auxquels seule une partie des ressources est allouée.
- Plusieurs conteneurs peuvent être créés, chacun d'entre eux peut contenir plusieurs programmes.
- Exécution en parallèle, en concurrence et même interaction.



# Machine virtuelle vs conteneurs



# Docker : installation



- Télécharger  
<https://www.docker.com/products/docker-desktop>
- Disponible sous
  - Linux,
  - Mac,
  - Windows (utilise votre installation de WSL).
- Logiciel libre et gratuit.

# Docker : fonctionnement

## Aller chercher une image

- `docker pull debian`  
Va chercher une image dans les dépôts Docker.
- `docker images`  
Liste les images disponibles.

## Exécuter un conteneur

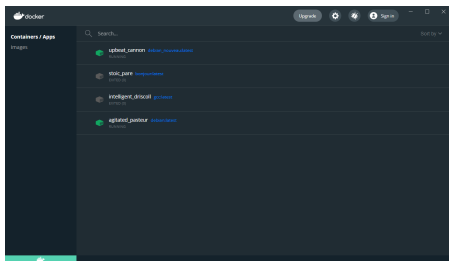
- `docker run debian echo "Bonjour de debian"`  
Exécuter la commande donnée, puis sortir.
- `docker run -it debian`  
Exécution interactive.

## Lister ses conteneurs

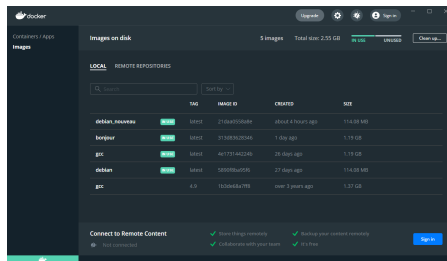
- `docker ps -a`

# Docker Desktop

L'interface graphique de Docker Desktop permet de retrouver ces informations.



La liste des conteneurs.



La liste des images.

# DockerHub

## Trouver les images qui vous intéressent

- <https://hub.docker.com/>
- Exemples :
  - Distributions linux (ubuntu, debian, fedora, ...).
  - Environnements pour exécuter ou compiler vos codes (python, gcc, ...)

## Exemple

Vous voulez avoir différentes versions du compilateur gcc

- `docker pull gcc` : va chercher la dernière version en date.
- `docker pull gcc:4.9` : va chercher la version 4.9.

# Exécution interactive dans un conteneur

## Rien n'est conservé...

- `docker run -it debian` puis dans le conteneur
  - `rm -rf bin`
  - `exit`
- Si on lance un autre conteneur, `bin` est là.

## ... sauf si on le demande

- Faire des changements dans le conteneur (par exemple : `touch coucou`).
- `docker commit [id_conteneur] nouveau_debian`  
Crée une nouvelle image avec nos changements.

# Publier un docker : image ou Dockerfile

## Publier une image

- Rappel : on a créé l'image `debian_nouveau` avec `docker commit`.
- On peut l'exporter :  
`docker save debian_nouveau > debian_nouveau.tar`
- Taille du fichier : 114 Mo.

## Alternative : Dockerfile

- Dockerfile : fichier texte qui explique comment créer l'image avec laquelle travailler.
- On peut construire l'image à partir du Dockerfile.
- Il suffit donc de partager le Dockerfile !

# Syntaxe du Dockerfile

## Situation

Dans le répertoire de travail : `hello.cpp`

## Fichier Dockerfile

```
FROM gcc:latest
COPY . /usr/src/bonjour
WORKDIR /usr/src/bonjour
RUN g++ -Wall -o bonjour hello.cpp
ENTRYPOINT ["/bonjour"]
CMD ["tout le monde"]
```

## Créer l'image

```
docker build -t bonjour .
```

## Syntaxe

- FROM : quelle image de base.
- COPY : copier nos fichiers sources.
- WORKDIR : répertoire de travail (pour la suite).
- RUN : exécuter une commande.
- ENTRYPOINT : commande à exécuter en lançant le conteneur.
- CMD : commande à exécuter (ou paramètres).




## Exemple plus en détail : compilation C++

- On peut exécuter un conteneur à partir de l'image `bonjour`.  
`docker run bonjour`
- On peut donner un paramètre : `docker run bonjour Marie`

### D'autres versions de gcc

- On n'est pas obligé d'utiliser la dernière version de gcc.
- Il suffit de changer FROM.

### Exemples disponibles sur moodle

-  <https://lms.univ-cotedazur.fr/course/view.php?id=4774&section=1>
- Taille du Dockerfile : moins d'1 ko.

## Exemple plus élaboré : un serveur flask avec python

### Dépôt elements

- Le dépôt <https://github.com/plezowski/elements> contient un fichier pour créer la table des premiers éléments dans un fichier markdown.
- On veut créer une image pour créer un serveur qui affiche cette table.

### flask

- Dans le répertoire flask, une application pour lancer le serveur, convertir le markdown en html et l'insérer dans la page.

### Dockerfile

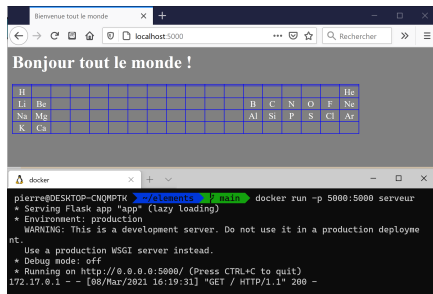
- Le fichier Dockerfile pour créer l'image correspondante.

# Résultat de flask

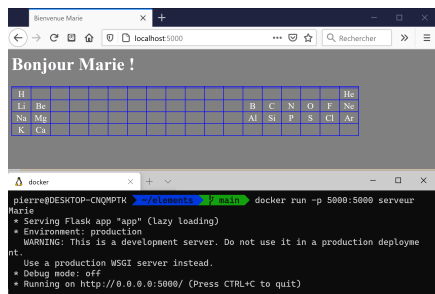
## Création de l'image

`docker build -t serveur .`

## Première exécution



## Avec un paramètre



# Commandes docker de base

## Quelques commandes

- `docker images` : liste des images disponibles.
- `docker ps -a` : liste des conteneurs et leur état.
- `docker run` : exécute une commande dans un nouveau conteneur.
- `docker start` : démarre un conteneur arrêté.
- `docker stop` : arrête un conteneur.
- `docker build` : construit une image à partir d'un Dockerfile.
- `docker commit` : crée une nouvelle image à partir des changements faits dans un conteneur.
- `docker save` : sauvegarder une image vers une archive tar.

## Documentation en ligne

 <https://docs.docker.com/engine/reference/commandline/docker/>

# L'API REST de docker : grands principes

- API : interface de programmation d'application
- Interface pour se connecter au service d'un logiciel ou d'un site (ici Docker daemon) pour programmer ou échanger des fonctionnalités
- On va communiquer avec le Docker daemon par l'intermédiaire de la ligne de commandes, plutôt que par l'application Docker Desktop
- Les retours vont aussi être dans le terminal, mais les effets vont bien être sur nos conteneurs et images (donc visibles aussi dans Docker Desktop)

## Premiers échanges

- Connaître la version pour l'API : `docker version`

```
Client: Docker Engine - Community
Version:      20.10.21
API version:  1.41
Go version:   go1.19.2
Git commit:   baeda1f82a
Built:        Tue Oct 25 17:53:02 2022
OS/Arch:      darwin/arm64
```

- Maintenant, affichons les images :

```
curl --unix-socket /var/run/docker.sock http://localhost/v1.41/images/json
```

```
[
  {
    "Containers":-1,
    "Created":1674491317,
    "Id":"sha256:c07781fc08369f41...
    "RepoTags":["bonjour:latest"],
    "Size":1180052483
  }
]
```

## Création et exécution

- Créer un conteneur :

```
curl --unix-socket /var/run/docker.sock -H "
Content-Type: application/json"-d '{"Image": "bonjour",
  "Cmd": ["Titi"]}' -X POST http://localhost/v1.41/
containers/create
```

```
{"Id": "2de71b0aa5b649f556f7ea9e56f64faf19d5c4fc85be6e35b896156a3fa85896",
"Warnings": []}
```

- Exécuter un conteneur :

```
curl --unix-socket /var/run/docker.sock -X POST http://
localhost/v1.41/containers/2de71b0aa5b6/start
```

- Récupérer la sortie standard d'un conteneur dans un fichier :

```
curl --unix-socket /var/run/docker.sock "http://
localhost/v1.41/containers/2de71b0aa5b6/logs?stdout=1"
--output tmp.txt
```

- 1 Introduction
- 2 Virtualisation
- 3 Conteneurs
- 4 Les actions github**
- 5 Références



# Présentation et motivation

## Quelques objectifs

- (Faire) tester son code de façon automatique.
  - Aussi pour des versions différentes des bibliothèques.
- Compiler / exécuter son code et en récupérer la production.

## Le bouton Actions de github

- Guide pour créer un **workflow** (littéralement, *flux de travaux*).
- Donne les résultats de chaque *workflow*.
  - Et éventuellement les fichiers (**artifacts**) produits.

## En dehors de github

- gitlab et bitbucket ont des outils similaires.

# Les actions github, en pratique

## Les fichiers YAML

- Un fichier `.yaml` (ou `.yml`) par workflow.
  - À placer dans `.github/workflows`
  - Expliquent ce qu'il faut faire, un peu comme un [Dockerfile](#).
- 
- github fournit des [modèles](#) (templates).
  - Nous allons voir un exemple simple.

# Exemple de fichier YAML

```
name: Exemple workflow
on:
  push:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    name: Construire et tester
    steps:
      - uses: actions/checkout@v2
        name: Recupérer les fichiers du dépôt
      - name: Installe Python 3.8
        uses: actions/setup-python@v2
        with:
          python-version: 3.8
      - run: python classifieur.py
        name: Exécuter mon script python
```

- name : nom du workflow
- on : explique à quel(s) moment(s) l'exécuter.  
Ici, à chaque [git push](#).
- runs-on : dit sur quelle machine virtuelle l'exécuter. Ces machines sont mises à disposition par github.
- steps : les étapes à appliquer.  
Ici,
  - récupérer le code,
  - installer python,
  - puis exécuter le script.

## Exemples en ligne

Pour le projet elements : <https://github.com/plezowski/elements>

- Source : dans `.github/workflows`
- Résultats : voir <https://github.com/plezowski/elements/actions>

### Les workflows

- Exécution automatique avec python 3.7, 3.8, 3.9 : `execution.yaml`.
- Génération automatique de release à chaque push (qui contient le fichier produit) : `preversion.yaml`.

- 1 Introduction
- 2 Virtualisation
- 3 Conteneurs
- 4 Les actions github
- 5 Références**

# Références

-  La documentation de VirtualBox  
<https://www.virtualbox.org/wiki/Documentation>
-  Documentation de docker <https://docs.docker.com/>
-  Aide-mémoire docker <https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf>
-  Documentation de github actions  
<https://docs.github.com/en/actions>