Gestion des architectures

M1 RÉSEAUX & TÉLÉCOMS – RT0702

OLIVIER FLAUZAC

Gestion des invités

Eléments de gestion

- gestion des installations
- gestion des configurations
  - interne
  - système
  - réseau

Reproductibilité des installations

- partage des images
- mise en place de dépôts

Configuration

- *provisionning*
- configuration système

Solutions de gestion

Gestion du cycle de vie des invité

- permettre la reproductibilité des installations
- mise en place d'environnements de développement
- mise en place d'environnements de test
- mise en place d'environnements pédagogiques

Assurer un développement indépendant des plates-formes

- portabilité

Solution de gestion

Mise en place d'environnements de développement

- mise en place d'environnements de test
- mise en place d'environnements pédagogiques

Simplifier les configurations

- réseau
- système
- en local / cluster / cloud

Vagrant

Généralités

https://www.vagrantup.com

Outils d'intégration pour
- la configuration
- la portabilité
- la reproductibilité

Surcouche pour des outils
- de virtualisation
- de *provisionning*

Installation sur toute plate-forme

Lien avec une solution de gestion d'invités
- *VirtualBox* par défaut

Principe général

Exploitation en ligne de commande

Commande : `vagrant`

Récupération d'images d'invités dans des dépôts

Description des éléments de configuration dans un fichier

Exécution avec / sans interface graphique

Connexion en console dans le terminal

Création d’un invité

```
vagrant init
```

Création du `Vagrantfile`

- format Ruby
- structuré en zones de configuration

Création possible en spécifiant une distribution

La gestion des Box

Images source, image de base box

Importation des box

Création d'un invité à partir d'une box

```
vagrant box add xxx/yyy
```

xxx : répertoire
yyy : image de distrib

```
vagrant box add nom url
```

```
vagrant box add nom fichier
```

```
recherche222:Vag olivier$ vagrant box add  debian/jessie64
==> box: Loading metadata for box 'debian/jessie64'
    box: URL: https://atlas.hashicorp.com/debian/jessie64
This box can work with multiple providers! The providers that it
can work with are listed below. Please review the list and choose
the provider you will be working with.

1) lxc
2) virtualbox

Enter your choice: 2
==> box: Adding box 'debian/jessie64' (v8.3.0) for provider: virtualbox
    box: Downloading: .../...
    https://vagrantcloud.com/xxx/8.3.0/providers/virtualbox.box
==> box: Successfully added box 'debian/jessie64' (v8.3.0) .../...
    for 'virtualbox'!
```

Les invités

Intégration de la box dans le Vagrantfile

Création automatique de la configuration au premier démarrage

```
vagrant up
```

Possibilité de tester la mise à jour de la box

Facilités

- accès à l'invité `vagrant ssh`
- répertoire partagé `/vagrant`

Affichage des propriétés au démarrage

Fichier Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "debian/jessie64"
  ...
end
```

Démarrage

```
vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'debian/jessie64' is up to date...
==> default: Clearing any previously set network interfaces...
...
...
vagrant ssh
```

Configuration de la machine

`config.vm`

- `box` : box de l'invité
- `box_check_update` : vérification de la box
- `hostname` : nom de l'invité
- `Provider` : gestionnaire de machine virtuelle utilisé
- `network` : configuration réseau
- `provision` : gestionnaire de provisionning
- `synced_folder` : répertoire partagé synchronisé

Connexion SSH

`config.ssh`

Configuration de la connexion `ssh` de `vagrant ssh`

- `config.ssh.username` login pour la connexion ssh
  - par défaut `vagrant`
- `config.ssh.password` mot de passe pour la connexion ssh
  - par défaut pas de mot de passe

Possibilité de gérer la connexion par un échange de clé

Configuration du réseau

`config.vm.network`

Configuration en ajout à la configuration de la box

- généralement box en nat
- ajout de cartes et de la configuration correspondante sur l'invité si nécessaire

3 modes

- forward de port
- réseau privé (NAT)
- réseau public

Configuration automatique ou statique

Forward

```
config.vm.network "forwarded_port", guest: 80, host: 8080
```

```
 ...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 80 (guest) => 8000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
    ...
```

Privé

```
config.vm.network "private_network", type: "dhcp"
```

```
  ...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
    ...
==> default: Configuring and enabling network interfaces...
    ...
```

Public

```
config.vm.network "public_network",
                        bridge:"en4: Ethernet Thunderbolt",
                        type: "dhcp"
```

```
 ...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: bridged
    ...
```

Provisionning

Fichier
- upload d'un fichier / répertoire sur l'invité

Shell
- passage des commandes dans le Vagrantfile
- exécution d'un script après upload

Provisionner
- Ansible, Chef, Puppet, Docker, salt

Provisionning possible
- au premier démarrage
- à l'exécution de `--provision`

Possibilité de refuser le *provisionning* avec `--no-provision`

Exemples de *provisionning*

file

```
config.vm.provision "file",   source: "~/archive.zip",
                              destination: "archive.zip"
```

shell

```
config.vm.provision "shell", path: "script.sh"
```

shell

```
config.vm.provision "shell",
                    path: "http://www.flauzac.eu/scrip.sh"
```

Docker

Docker

https://www.docker.com/

Gestionnaire / administrateur de conteneurs

Gestionnaire *templates*

basé sur LXC

Ensemble de services pour

- créer
- éditer
- exécuter des conteneurs

Permet la gestion des ressources de conteneurs

Architecture Containers as a Service (*CaaS*)

Porté sur Windows et MAC OS X

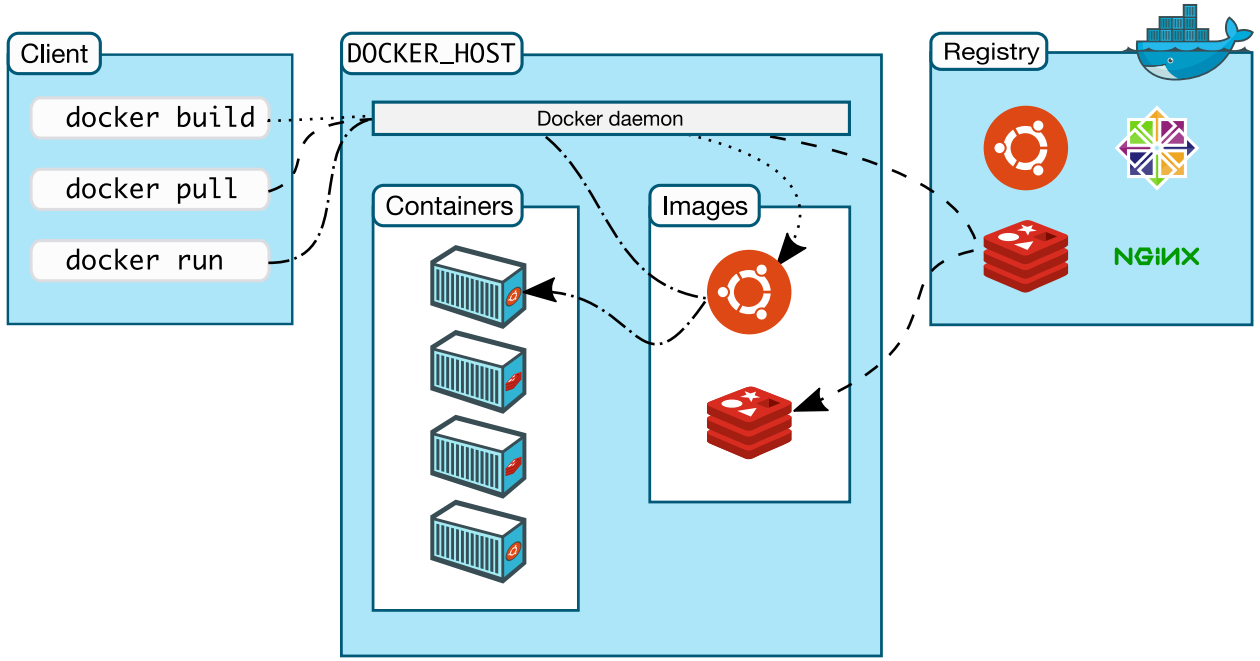
Architecture

Le démon Docker

- s'exécute sur l'hôte
- inter actions avec le client docker
- gère les images
- assure le cycle de vie des conteneurs

Le client Docker

- interface d'accès au démon
- inter actions avec l'utilisateur



Eléments

Les images

- *templates* en lecture seule
- distributions avec ou sans applications
- master des conteneurs docker

Le répertoire Docker

- contient les images
- publics ou privés

Les conteneurs Docker

- environnement d'exécution
- créé à partir d'une image docker
- isolés

Exécution d’un conteneur

```
docker run -i -t ubuntu /bin/bash
```

Vérification de la présence de l'image ubuntu

- téléchargement si nécessaire

Création du conteneur

- allocation du système de fichier
- allocation du réseau
- configuration de l'adresse ip
- exécution de la commande

Gestion des logs et des entrées / sorties

Exécution jusqu'à la terminaison de la commande

```
root@debian:~# service docker start
root@debian:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7
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.
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 Hub account:
  https://hub.docker.com

For more examples and ideas, visit:
  https://docs.docker.com/userguide/
```

Commandes

Exécution

```
docker run options image commande
```

- `-t` allocation d'un pseudo-terminal
- `-i` interactif : liaison STDIN de l'bôte avec le tty
- `-d` more démon exécution en tâche de fond

Liste des conteneurs

```
docker ps
```

Affiche la sortie standard

```
docker logs conteneur
```

Arrêt d'un conteneur

```
docker stop conteneur
```

```
root@debian:~# docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"
907212140d6add5d9fed0a0e1924344ef28316eb88dff020c9c4e06bb848e721
root@debian:~# docker ps
CONTAINER ID  IMAGE   COMMAND                  CREATED          STATUS        PORTS  NAMES
907212140d6a  ubuntu  "/bin/sh -c 'while tr"  13 seconds ago   Up 12 seconds
insane_archimedes
root@debian:~# docker logs insane_archimedes
hello world
hello world
   ....
hello world
hello world
root@debian:~# docker stop insane_archimedes
insane_archimedes
root@debian:~# docker ps
CONTAINER ID   IMAGE COMMAND  CREATED  STATUS   PORTS  NAMES
```

Gestion des images

Le docker hub

- centralisateur des informations d'utilisation des images
- possibilité de créer une image personnelle

Le répertoire docker

- répertoire des images
- commandes

```
pull image
search clé
push image
```

Gestion du réseau
le *forward* de ports

Possibilité de relayer des ports de l'hôte sur l'invité

Deux options :

- -p relaye tous les ports nécessaires à l'invité depuis l'hôte
- -p hôte:invité

Configuration réseau

Mise en place d’un réseau par défaut

Création de 3 instances réseau

- `bridge` : réseau spécifique à docker
- `none` : pas de réseau
- `host` : même configuration que l'hôte

Configuration réseau

```
root@debian:~# docker network ls
NETWORK ID          NAME                DRIVER
680cd5c1b318        none                null
077fa1a68526        host                host
f9536e13b42b        bridge              bridge
```

```
root@debian:~# ip addr show
...
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:ed:90:21:61 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:edff:fe90:2161/64 scope link
       valid_lft forever preferred_lft forever
```

Mode réseau Bridge

réseau des conteneurs sur un seul hôte

Plusieurs bridge possibles

Isolation des conteneur par bridge

Bridge par défaut : docker0

- pas d'interface système
- DHCP intégré : 172.17.x.x
- mode NAT

Possibilité de créer un bridge contenant ethX

Mode NAT

- création du réseau bridge docker
- allocation automatique de la plage d'adresses
- gestion automatique de la passerelle / forward

```
network create --driver bridge mybridge
```

```
root@debian:~# docker network inspect mybridge
[
    {
        "Name": "mybridge",
        "Id":
"7d7146596c4893936678e46957ca11b3c851ec1ffadba88b4fd54e839d80e28b",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Containers": {},
        "Options": {}
    }
]
```

```
root@debian:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
  ...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
  ...
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
  ...
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
      link/ether 02:42:ed:90:21:61 brd ff:ff:ff:ff:ff:ff
      inet 172.17.0.1/16 scope global docker0
         valid_lft forever preferred_lft forever
      inet6 fe80::42:edff:fe90:2161/64 scope link
         valid_lft forever preferred_lft forever
21: br-7d7146596c48: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
      link/ether 02:42:09:66:cc:e5 brd ff:ff:ff:ff:ff:ff
      inet 172.18.0.1/16 scope global br-7d7146596c48
         valid_lft forever preferred_lft forever
```

```
root@debian:~#docker run -t -i --net=mybridge debian /bin/bash
root@89a104f7c590:/# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
22: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:2/64 scope link
       valid_lft forever preferred_lft forever
```

```
root@debian:~# ip addr show
...
21: br-7d7146596c48: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:09:66:cc:e5 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 scope global br-7d7146596c48
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9ff:fe66:cce5/64 scope link
       valid_lft forever preferred_lft forever
23: vetha95fad2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-7d7146596c48 state UP group default
    link/ether 1e:a4:21:25:20:25 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::1ca4:21ff:fe25:2025/64 scope link
       valid_lft forever preferred_lft forever
```

Mode full bridge

Création du bridge

- ajout de l'interface hôte dans le bridge
- ajout du bridge dans le fichier `/etc/default/docker`

```
DOCKER_OPTS="-b=nom_bridge"
```

Redémarrage du service

Docker File

Construction d'images spécifiques

- Description des opérations dans un `dockerfile`
- Commandes regroupées dans un fichier assurant la suite des opérations à exécuter
- Commandes

`ADD / COPY source dep` : copie
`CMD cmd parms` : commande d'exécution
`ENV var val` : définition d'un variable d'environnement
`EXPOSE port` : configure le réseau avec le port
`FROM image` : image de base
`RUN cmd` : exécution d'une commande
`VOLUME dir` : accès au répertoire partagé sur l'hôte

Utilisation

```
build -t file
```

Un exemple de fichier

```
FROM debian:latest

RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get clean

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```