Lab 5 - Provisioning

Dans la terminologie de Vagrant, le processus d'installation et de la configuration automatique du logiciel dans le système d'exploitation invité pendant vagrant up est appelé **provisioning** et les outils pour effectuer cette opération sont appelés provisioners.

Un provisioner peut exécuter des commandes arbitraires; il peut, par exemple,installer

Apache comme ceci:

```
$ sudo apt-get install apache2
```

Le lancer

```
$ sudo service apache2 restart
```

ou même télécharger du code source:

```
$ curl ftp://ftp.ruby-lang.org/pub/ruby/2.2/ruby-2.2.0.tar.gz > ruby.tar.gz
```

Il n'y a aucune restriction; toutes les commandes que vous exécutez habituellement dans une session SSH pour reconfigurer votre système peuvent être exécutées par les provisioners. Il existe de nombreuses approches différentes pour provisioner des machines virtuelles. Vous pouvez provisionner le système d'exploitation invité en écrivant des scripts shell ou en utilisant des outils plus sophistiqués tels que Puppet, Ansible et Chef. Vagrant prend en charge les provisioners suivants: • File • Shell scripts • Ansible • CFEngine • Chef • Docker • Puppet • Salt

Configuring Provisioning

La configuration de l'approvisionnement est stockée dans Vagrantfile. Voici la syntaxe générale pour configurer l'approvisionnement:

```
Vagrant.configure(2) do |config|
   ...
   config.vm.provision "THE-NAME-OF-THE-PROVISIONER" ...
end
```

Le premier paramètre de `config.vm.provision` est le nom de l'approvisionneur que vous souhaitez utiliser. Remplacez "THE-NAME-OF-THE-PROVISIONER" par l'un des provisioners intégrés, comme suit: • shell • puppet • ansible • chef-solo

Les autres paramètres de config.vm.provision dépendent du fournisseur que vous avez choisi. Chacun prend des paramètres différents et certains approvisionneurs (Puppet, par exemple) n'ont aucun paramètre obligatoire car toutes les options ont des valeurs par défaut raisonnables. Pour utiliser Puppet, vous pouvez simplement écrire ce qui suit:

```
Vagrant.configure(2) do |config|
  ...
  config.vm.provision "puppet"
end
```

D'autres approvisionneurs, tels que le shell, ont besoin d'au moins un paramètre supplémentaire. Pour l'approvisionneur shell, vous devez définir le script shell à exécuter:

```
Vagrant.configure(2) do |config|
   ...
   config.vm.provision "shell", path: "script.sh"
end
```

Étant donné que les fournisseurs ne partagent aucune option commune à l'exception du premier paramètre, ils seront traités séparément.

Quand le provisionnement aura-t-il lieu?

Par défaut, l'approvisionnement est exécuté uniquement lors de la première exécution de `vagrant up`.

Si vous arrêtez le système en utilisant `$ vagrant halt` ou `$ vagrant suspend`, la prochaine exécution de `$ vagrant up`sautera les approvisionneurs. Vagrant suppose que les approvisionneurs ont déjà été exécutés et que tous les packages nécessaires sont déjà installés. Vous pouvez modifier le comportement par défaut de plusieurs manières. Tout d'abord, vous pouvez activer et désactiver l'approvisionnement pendant `$ vagrant up` en utilisant les indicateurs **--provision** et **--no-provision**:

```
$ vagrant up --provision
$ vagrant up --no-provision
```

puis exécutez les approvisionneurs:

```
$ vagrant provision
```

Jekyll Box avec Shell Provisioner

Commençons par le premier projet: un box Vagrant pour les sites Web Jekyll configurés avec un approvisionneur shell. (Toutes les commandes nécessaires à la construction de ce box ont déjà été

abordées dans le lab 3) La tâche à accomplir est d'automatiser le processus d'exécution de ces commandes avec un approvisionneur shell. Le but est de définir la configuration de la VM afin qu'après une seule commande `$ vagrant up`, vous ayez un OS invité avec Ruby et Jekyll déjà installés.

Box Source Code

Créez le répertoire vide pour ce projet:

```
# Host OS
$ cd folder/with/examples
$ mkdir vagrant-jekyll-shell
```

Dans ce répertoire, créez deux fichiers: le Vagrantfile et le script.sh. Vagrantfile:

```
Vagrant.configure(2) do |config|
   config.vm.box = " ubuntu/focal64"
   config.vm.provision "shell", path: "script.sh"
end
```

script.sh

```
#!/usr/bin/env bash
echo "Installing: nodejs, lynx, ruby and jekyll..."
sudo apt-get update -y
sudo apt -y install make build-essential ruby ruby-dev
echo "export GEM_HOME=$HOME/gems" >> /home/vagrant/.bashrc
echo "export PATH=$HOME/gems/bin:$PATH" >> /home/vagrant/.bashrc
source /home/vagrant/.bashrc
sudo apt-get install lynx
gem install jekyll bundler
```

Le Vagrantfile définit le nom de le box de base sur ubuntu/focal64 et active l'approvisionnement automatique avec un script shell nommé script.sh. La ligne suivante indique à Vagrant de provisionner le box exécutant le script shell nommé script.sh:

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

First Run of the Shell Provisioner

Une fois que vous avez créé les deux fichiers indiqués, vous pouvez tester le box. Entrez le répertoire qui contient ces fichiers:

```
# Host OS
$ cd vagrant-jekyll-shell
```

et exécutez la commande:

```
# Host OS
$ vagrant up
```

Le script de provisioning télécharge Jekyll et divers gems et les recompile (entre autres processus). Cela peut prendre plus de 10 minutes, vous devez donc être patient.

Vous verrez l'ancienne sortie de Vagrant avec deux nouveaux éléments à la toute fin. L'un d'eux sera le message ❶ concernant l'exécution du fournisseur de shell; l'autre est le message ❷ concernant l'installation de NodeJS, Lynx, Ruby et Jekyll:

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/focal64'...
...
❶ ==> default: Running provisioner: shell...
==> default: stdin: is not a tty
❷ ==> default: Installing: nodejs, lynx, ruby and jekyll...
```

Votre système d'exploitation invité est maintenant en cours d'exécution, ce que vous pouvez vérifier avec la commande suivante:

```
# Host OS
$ vagrant status
```

Ouvrez la session SSH sur le système d'exploitation invité avec ceci:

```
# Host OS
$ vagrant ssh
```

Vous pouvez vérifier que Jekyll est installé et prêt avec les éléments suivants:

```
# Guest OS
$ jekyll --version
```

Vous n'avez pas à exécuter de commandes pour installer Ruby ou Jekyll; cela a déjà été fait par le script shell script.sh. Vous pouvez détruire le système avec ceci:

```
# Host OS
$ vagrant destroy
```

Et ramenez-le à la vie avec ceci:

```
# Host OS
$ vagrant up
```

Générez un nouveau box package à partir de cette configuration.

```
# Host OS
$ vagrant package --output vagrant-jekyll-shell-v0.1.0.box
```

Using the Shell-Provisioned Box

Le box est prête, vous pouvez donc changer le rôle ; vous allez maintenant effectuer le travail du développeur. Tout d'abord, vous devez installer la box dans le système avec la commande suivante :

```
# Host OS
$ vagrant box add vagrant-jekyll-shell-v0.1.0 vagrant-jekyll-shell-v0.1.0.box
```

Vous devez exécuter cette commande dans le dossier qui contient le fichier vagrant-jekyllshell-v0.1.0.box généré dans la section précédente. Vous pouvez également ajouter le chemin d'accès au fichier:

```
# Host OS
$ vagrant box add vagrant-jekyll-shell-v0.1.0 /some/path/vagrant-
jekyllshell-v0.1.0.box
```

La sortie de cette commande doit maintenant contenir l'entrée vagrant-jekyll-shell:

```
# Host OS
$ vagrant box list
```

Jusqu'ici tout va bien. Créons un nouveau projet nommé flowers-vagrant-jekyll-shell, qui sera un site Web dirigé par Jekyll et intitulé Flowers:

```
# Host OS
```

```
$ cd folder/with/examples
$ mkdir flowers-vagrant-jekyll-shell
$ cd flowers-vagrant-jekyll-shell/
$ vagrant init -m vagrant-jekyll-shell-v0.1.0
```

Notez que le nom de le box qui a été installée avec la commande **$ vagrant box add** est passé à la commande **$ vagrant init**.

Modifiez maintenant le fichier Vagrant et ajoutez une directive pour configurer la redirection de port.

```
Vagrant.configure(2) do |config|
    config.vm.box = "vagrant-jekyll-shell-v0.1.0"
    config.vm.network :forwarded_port, guest: 4000, host: 8100, host_ip:
"127.0.0.1"
end
```

Lorsque le Vagrantfile est prêt, vous pouvez démarrer la VM avec ceci:

```
# Host OS
$ vagrant up
```

Cette fois, la procédure de démarrage du système sera raisonnablement courte: environ une minute. Vous n'êtes pas obligé d'exécuter les approvisionneurs; cela a été fait par un ingénieur système lors de la création du fichier box. Ruby et Jekyll sont intégrés dans le box et le développeur n'a pas à les télécharger, les recompiler ou les installer. L'environnement de développement est prêt. Vous pouvez créer un nouveau projet et le servir sur le port transféré:

```
# Host OS
$ vagrant ssh
```

```
# Guest
$ cd /vagrant
$ jekyll new -f .
$ jekyll build
$ jekyll serve -H 0.0.0.0 --detach
$ logout
```

Grâce au partage de répertoires, le répertoire flowers-vagrant-jekyll-shell/ du host contient désormais le code source du site flowers. Et en raison de l'entrée forwarded_port, vous pouvez utiliser votre navigateur

pour accéder à http://127.0.0.1:8100/. Vous verrez la page du projet Jekyll. Maintenant, le projet flowers-vagrant-jekyll-shell est configuré pour que vous deviez exécuter `jekyll serve -H 0.0.0.0 --detach` chaque fois que vous démarrez la VM. Vous pouvez l'éviter avec le **provisioner shell**. Modifiez le Vagrantfile en introduisant les changements suivants:

```
Vagrant.configure(2) do |config|
  config.vm.box = "vagrant-jekyll-shell-v0.1.0"
  config.vm.network :forwarded_port, guest: 4000, host: 8100, host_ip:
"127.0.0.1"
  $script =<<-'SCRIPT'
            cd /vagrant
            jekyll serve -H 0.0.0.0 --detach
            SCRIPT
config.vm.provision "shell", inline: $script, run: "always"
end
```

Le Vagrantfile contient une variable nommée $script. La partie entre <<-'SCRIPT' et SCRIPT est un texte affecté à la variable $script, qui est, bien sûr, le code du script shell composé de deux commandes:cd /vagrant et jekyll serve. Grâce à config.vm.provision, ce script sera automatiquement exécuté au démarrage de la VM invitée. Maintenant, détruisez la VM invitée et redémarrez-la:

```
# Host OS
$ vagrant destroy
$ vagrant up
```

Cette fois, l'URL http://127.0.0.1:8100 est disponible juste après $ vagrant up. Vous n'avez pas besoin d'ouvrir la session SSH et de démarrer manuellement le serveur HTTP.

Synced folders

En utilisant des **dossiers synchronisés** , Vagrant synchronisera automatiquement nos fichiers vers et depuis la machine invitée. En d'autres termes, Vagrant partage notre répertoire de projet (où se trouve le fichier Vagrant) avec le répertoire **/vagrant** de notre machine invitée. Exécutez **à** nouveau **vagrant up** et **SSH** dans notre machine pour voir:

```
$ vagrant up
$ vagrant ssh
$ ls /vagrant
Vagrantfile
```

Le **Vagrantfile** que nous voyons à l'intérieur de la machine virtuelle est en fait le même **Vagrantfile** qui se trouve sur notre machine hôte. Allez-y et touchez un fichier pour le vérifier:

```
# Guest
$ touch /vagrant/share_me
$ ls
$ exit
logout
Connection to 127.0.0.1 closed.
# Host
$ ls
share_me Vagrantfile
```

Le fichier "share_me" est maintenant sur notre machine hôte. Comme nous pouvons le voir, Vagrant a gardé les dossiers synchronisés. Avec les dossiers synchronisés, nous pouvons continuer à utiliser notre propre éditeur sur notre machine hôte et synchroniser les fichiers dans la machine invitée.

Exercice: Apache install via shell provisioner

Ecrivez une configuration vagrant que crée une VM Ubuntu 20.04 et y installe Apache2 par un Shell file provisioner -- Dans cette section, nous allons configurer Apache pour notre projet de base à l'aide d'un script shell, **bootstrap.sh** :

```
$ ls
bootstrap.sh Vagrantfile
```

Le fichier "bootstrap.sh" ressemble à ceci:

```
$ cat bootstrap.sh
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /vagrant /var/www
```

Ensuite, nous devons configurer Vagrant pour exécuter ce script shell lors de la configuration de notre machine. Nous faisons cela en éditant le **Vagrantfile** :

Le fichier "bootstrap.sh" ressemble à ceci:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "ubuntu/focal64"
    config.vm.provision :shell, path: "bootstrap.sh"
end
```

La ligne "provision" indique à Vagrant d'utiliser le **provisioner shell** pour configurer la machine, avec le fichier **bootstrap.sh**. Le **chemin du** fichier est relatif à l'emplacement de la **racine** du **projet** (où se trouve le **Vagrantfile**).

Après la configuration, nous exécutons simplement **vagrant up** pour créer notre machine virtuelle via l'approvisionnement automatique par Vagrant. Nous devrions voir la sortie du script shell apparaître dans notre terminal.

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/focal64' is up to date...
==> default: VirtualBox VM is already running.
```

Si la machine invitée fonctionne déjà à partir d'une étape précédente comme dans notre cas, nous exécutons **vagrant reload --provision**, qui redémarrera rapidement notre machine virtuelle, en ignorant l'étape d'importation initiale.

```
$ vagrant reload --provision

==> default: Attempting graceful shutdown of VM...
==> default: Checking if box ' ubuntu/focal64' is up to date...
==> 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: Forwarding ports...
default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
default: SSH address: 127.0.0.1:2222
default: SSH username: vagrant
default: SSH auth method: private key
default: Warning: Connection timeout. Retrying...
...
default: Warning: Remote connection disconnect. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
==> default: Mounting shared folders...
default: /vagrant => /home/k/my_vagrant
==> default: Running provisioner: shell...
...
==> default: The following NEW packages will be installed:
==> default: apache2 apache2-bin apache2-data libapr1 libaprutil1
libaprutil1-dbd-sqlite3
==> default: libaprutil1-ldap ssl-cert
==> default: 0 upgraded, 8 newly installed, 0 to remove and 0 not
upgraded.
==> default: Need to get 1,270 kB of archives.
==> default: After this operation, 5,050 kB of additional disk space
will be used.
==> default: Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main
libapr1 i386 1.5.0-1 [88.8 kB]
==> default: Get:2 http://archive.ubuntu.com/ubuntu/ trusty/main
libaprutil1 i386 1.5.3-1 [76.6 kB]
==> default: Get:3 http://archive.ubuntu.com/ubuntu/ trusty/main
libaprutil1-dbd-sqlite3 i386 1.5.3-1 [10.3 kB]
==> default: Get:4 http://archive.ubuntu.com/ubuntu/ trusty/main
libaprutil1-ldap i386 1.5.3-1 [8,552 B]
==> default: Get:5 http://archive.ubuntu.com/ubuntu/
trusty-updates/main apache2-bin i386 2.4.7-1ubuntu4.1 [821 kB]
==> default: Get:6 http://archive.ubuntu.com/ubuntu/
trusty-updates/main apache2-data all 2.4.7-1ubuntu4.1 [160 kB]
```

```
==> default: Get:7 http://archive.ubuntu.com/ubuntu/
trusty-updates/main apache2 i386 2.4.7-1ubuntu4.1 [87.6 kB]
==> default: Get:8 http://archive.ubuntu.com/ubuntu/ trusty/main
ssl-cert all 1.0.33 [16.6 kB]
==> default: dpkg-preconfigure: unable to re-open stdin: No such file
or directory
...
==> default: Processing triggers for ureadahead (0.100.0-16) ...
==> default: Processing triggers for ufw (0.34~rc-0ubuntu2) ...
```

L'indicateur de provision sur la commande reload demande à Vagrant d'exécuter les **approvisionneurs**, car généralement Vagrant ne le fera que sur le premier `vagrant up`.

Une fois Vagrant terminé, le serveur Web sera opérationnel. Cependant, nous ne pouvons pas encore voir le site Web à partir de notre propre navigateur, mais nous pouvons vérifier que l'approvisionnement fonctionne en chargeant un fichier depuis SSH dans la machine:

```
$ vagrant ssh

Welcome to Ubuntu 20.04 LTS (GNU/Linux 3.13.0-39-generic i686)
...
vagrant@vagrant-ubuntu-focal-64:~$ wget -qO- 127.0.0.1
```

Cela fonctionne car dans le script shell ci-dessus, nous avons installé Apache et configuré le **DocumentRoot** par défaut d'Apache pour qu'il pointe vers notre répertoire **/ vagrant** , qui est la configuration de dossier synchronisé par défaut par Vagrant.

Nous pouvons vérifier si Apache fonctionne réellement:

```
vagrant@vagrant-ubuntu-focal-64:~$ ps -ef|grep apache2

root 2295 1 0 03:44 ? 00:00:00 /usr/sbin/apache2 -k start

www-data 2297 2295 0 03:44 ? 00:00:03 /usr/sbin/apache2 -k start

www-data 2299 2295 0 03:44 ? 00:00:03 /usr/sbin/apache2 -k start
```

Exercice d'application

Modifiez le box basé sur « cicd-box » créé dans le Lab 4 pour installer les outils d'intégration continu à travers un script shell d'approvisionnement en démarrant automatiquement le serveur jenkins.