

# Rapport M340

---

VIRTUALISER UNE INFRASTRUCTURE INFORMATIQUE

Elie Hausmann

EPAI/MEGGITT | 04/05/2021

## Table des matières

1	Introduction .....	5
1.1	Objectifs du module .....	5
2	Qu'est-ce qu'un hyperviseur ? .....	5
2.1	Les hyperviseurs de type 1 .....	5
2.2	Les hyperviseurs de type 2 .....	6
2.3	Quelle est la différence entre les types 1 et 2 ? .....	7
3	Solution de virtualisation .....	7
4	Qu'est-ce que la virtualisation ? .....	8
4.1	Les avantages.....	8
4.2	Les inconvénients .....	8
4.3	Les domaines d'applications.....	9
4.4	Comment faire de la virtualisation ? .....	9
4.4.1	Création manuelle .....	9
4.4.2	Création automatique .....	10
4.5	Les clusters.....	10
4.5.1	Baies de stockage et SAN .....	10
4.5.2	Le stockage distribué.....	10
5	La virtualisation de serveur.....	11
6	Vagrant.....	11
6.1	Installation .....	12
6.2	Vagrantfile .....	12

6.3	Boxes.....	12
6.4	Provider .....	12
6.5	Commandes de base .....	13
6.6	Configuration basique d'un Vagrantfile .....	14
7	Packer.....	15
7.1	Installation .....	15
7.2	La terminologie.....	15
7.2.1	Build.....	15
7.2.2	Builder .....	15
7.2.3	Artefact.....	16
7.2.4	Provisioner.....	16
7.2.5	Post-processor.....	16
7.2.6	Modèle (template) .....	16
7.2.7	Commandes.....	16
7.3	Introduction aux modèles .....	17
7.3.1	Builders.....	17
7.3.2	Description .....	17
7.3.3	Min_packer_version.....	17
7.3.4	Post_processors .....	18
7.3.5	Provisionners.....	18
7.3.6	Variables.....	18
8	L'approvisionnement (provisioning).....	18
8.1	Automatisation .....	18
8.2	Outils de gestion de configuration .....	18

8.3	Fonctionnement .....	19
8.4	Quand les utiliser ? .....	19
8.5	Quelques exemples .....	19
9	Ansible.....	20
9.1	De quoi Ansible est capable ? .....	20
9.2	L'architecture.....	21
9.3	Les avantages d'Ansible .....	22
10	SaltStack.....	23
10.1	Les caractéristiques.....	23
10.2	Le fonctionnement.....	23
10.3	Les avantages .....	24
11	Les containers .....	24
12	Exercice 1 : Vagrant.....	25
12.1	Mettre en place un répartiteur de charge .....	26
12.1.1	Le Vagrantfile .....	27
12.1.2	Installation et configuration de HAProxy .....	28
13	Exercice 2 : Packer.....	29
13.1	Installation de Packer.....	29
13.2	Modification de la configuration .....	30
13.3	Adaptation à Ubuntu .....	30
14	Exercice 3 : SaltStack.....	33
14.1	Installation .....	33
14.2	Tutoriel de base .....	33
14.3	Les commandes de base .....	34
14.4	Les salt state.....	35

15	Exercice 4 : Proxmox .....	35
15.1	Installer Proxmox .....	36
15.2	Création de la basebox.....	37
15.3	Création des machines virtuelles .....	37
15.4	La mise en cluster.....	38
16	Conclusion.....	40
17	Bibliographie .....	41

# 1 Introduction

Les infrastructures informatiques actuelles sont de plus en plus virtualisées et passe bien souvent dans le cloud. Le cloud représente le futur de l'informatique et la virtualisation y joue le premier rôle. En effet, sans virtualisation il n'y a pas de cloud.

Ce module a donc pour but de nous faire comprendre quels sont les avantages de la virtualisation, quels outils utiliser et comment virtualiser une infrastructure. Cela passe par l'évaluation, la planification et l'exécution de la simulation virtuelle.

Ce document décrira les termes et les concepts théoriques abordés en classe ainsi que les exercices pratiques effectués lors de ce module.

## 1.1 Objectifs du module

L'objectif de ce module est de nous faire acquérir les compétences nécessaires à l'évaluation, la planification et à l'exécution d'une simulation virtuelle d'une infrastructure informatique. Voici les cinq objectifs plus précis que comprend ce module 340 :

1. Décrire les objectifs, les caractéristiques ainsi que les domaines d'application de systèmes de virtualisation.
2. Élaborer un concept approprié de virtualisation (ressources matérielles, moyens financiers, consommation d'énergie) pour un mandat prescrit. Mettre en évidence les possibilités d'économies financières.
3. Virtualiser et tester une infrastructure informatique avec des outils appropriés.
4. Définir, installer et configurer des composants appropriés pour un système de stockage et de sauvegarde.
5. Transférer une infrastructure virtuelle dans l'exploitation productive.

## 2 Qu'est-ce qu'un hyperviseur ?

Un hyperviseur est un logiciel permettant la création et l'exécution de machines virtuelles. La machine hôte est celle qui est utilisé en tant qu'hyperviseur, tandis que les machines virtuelles qui utilisent ses ressources sont appelées « invités ». Grâce à un hyperviseur, il est possible d'exécuter plusieurs systèmes d'exploitation sur une même machine physique, en même temps.

A l'heure actuelle, il existe deux types d'hyperviseurs : le type 1 (natif) et le type 2 (dit « hosted »). La description de ces deux différents types d'hyperviseur fait l'objet des prochains chapitres.

### 2.1 Les hyperviseurs de type 1

Un hyperviseur de type 1 est un logiciel qui s'exécute directement sur la partie matérielle du serveur ou d'un PC. Un hyperviseur de ce type n'a pas besoin qu'un système d'exploitation

soit installé sur la machine hôte. En effet, les hyperviseurs sont constitués d'un noyau hôte allégé, optimisé et souvent basé sur Linux. Ils n'ont donc plus besoin d'un autre OS et se suffisent à eux-mêmes pour créer et exécuter des machines virtuelles.

L'avantage de ce système est de pouvoir éliminer une couche séparant la machine virtuelle du matériel. Cependant, le serveur sur lequel un hyperviseur de type 1 est installé est « condamné » à héberger uniquement des machines virtuelles.

Un hyperviseur de type 1 comporte généralement une interface utilisateur rudimentaire qui permet de :

- Créer et supprimer des VM
- Démarrer et arrête des VM
- Monitorer les VM

Un exemple d'hyperviseur de niveau 1 est KVM. Il en existe plein d'autres évidemment mais l'avantage de la solution KVM est qu'elle a été intégrée au noyau Linux. Il est donc possible d'utiliser KVM sur les versions actuelles de Linux.

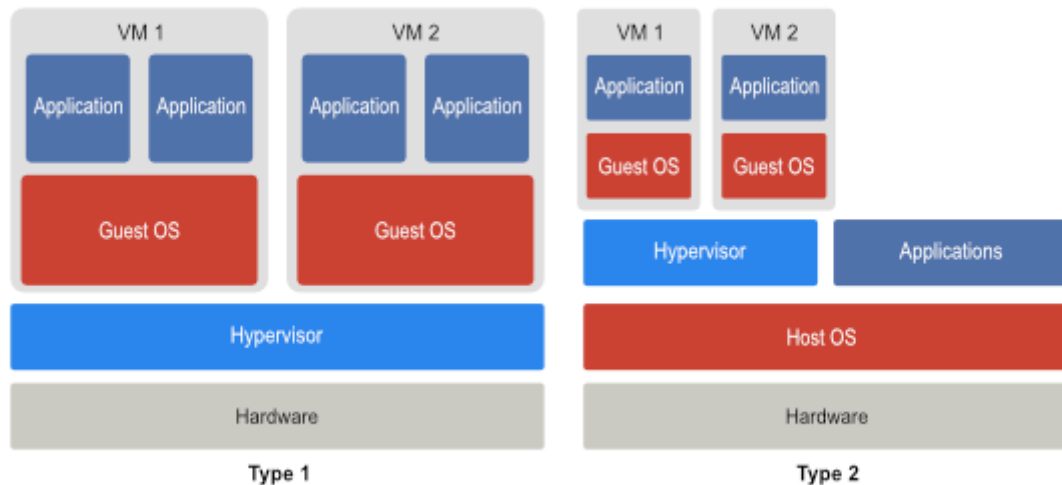
## 2.2 Les hyperviseurs de type 2

Un hyperviseur de type 2 est un logiciel qui s'exécute comme n'importe quelle autre application, sur un système d'exploitation. Il fonctionne en séparant les systèmes d'exploitation invités du système d'exploitation hôte, qui est lui-même exécuté sur le matériel.

Un exemple d'hyperviseur de type 2 que nous avons utilisé à maintes reprises est Virtual Box. Ce logiciel nous permet de créer et d'exécuter des machines virtuelles sur notre ordinateur personnel.

## 2.3 Quelle est la différence entre les types 1 et 2 ?

Foncièrement, un hyperviseur de type 1 ou de type 2 offre exactement les mêmes possibilités, à savoir l'exécution et la création de machines virtuelles. Le réel changement est la manière dont l'hyperviseur lui-même est installé et exécuté.



Comme le montre le schéma ci-dessus, un hyperviseur de type 1 n'a pas besoin d'un système d'exploitation tandis qu'un hyperviseur de type 2 ne peut pas s'en passer. Voilà la différence.

## 3 Solution de virtualisation

Pour virtualiser une infrastructure entière, un hyperviseur seul peut être utilisé mais il ne disposera pas de certaines fonctionnalités très utiles. Le but d'une solution de virtualisation est d'apporter les fonctionnalités nécessaires à la gestion d'une infrastructure virtualisée.

Voici quelques exemples de fonctionnalités :

- Monitoring avec un tableau de bord.
- Gestion des templates pour faciliter la création de machines virtuelles.
- Gestion des backups.
- Gestion d'un cluster d'hyperviseurs et migration de VM entre différents nœuds.
- API disponible pour automatiser l'approvisionnement.

Ces dites fonctionnalités sont fournies par des composants logiciels plus ou moins intégrés à une solution de virtualisation.

Les solutions de virtualisation les plus connues sont :

- VMware vSphere basé sur ESXi.
- Citrix Hypervisor basé sur Xen.
- Proxmox VE basé sur KVM.
- Microsoft Hyper-V basé sur Hyper-V.



## 4 Qu'est-ce que la virtualisation ?

La virtualisation consiste à créer et à exécuter une ou plusieurs représentations virtuelles d'un ordinateur ou de ses différentes ressources sur une même machine physique. En partant de ce postulat, il est possible de virtualiser une machine entière ou seulement les services nécessaires.

Contrairement à l'émulation qui reproduit le matériel de manière logiciel, la virtualisation utilise directement le matériel de la machine physique. L'avantage de la virtualisation par rapport à l'émulation est un réel gain de performances.

### 4.1 Les avantages

La virtualisation propose de nombreux avantages et remédie à plusieurs problèmes que l'utilisation de serveurs soulève. Voici quelques avantages à considérer :

- L'installation de n'importe quel OS dit standard.
- La portabilité des serveurs.
- La diminution de la consommation électrique des salles serveurs.
- L'administration simplifiée de l'ensemble des serveurs.
- La rationalisation des coûts et du matériel.
- La rationalisation des ressources comme le CPU et la RAM.

En effet, la virtualisation permet de maximiser l'utilisation des ressources disponibles d'un serveur physique. De ce fait, le besoin de matériel se fait moindre et donc, il n'est plus nécessaire d'acheter du matériel ni de dépenser autant d'argent dans la climatisation (ce qui permet de diminuer l'empreinte écologique en passant).

L'économie d'argent se fait également sur le coût de la maintenance des serveurs. Grâce à la virtualisation, supprimer puis recréer un service ne prends plus que quelques minutes. Ainsi, il n'est plus nécessaire de chercher l'origine d'une panne pendant plusieurs heures sur un serveur, une simple recréation de la machine virtuelle suffit.

### 4.2 Les inconvénients

Un des inconvénients de la virtualisation est la généralisation des vulnérabilités. En effet, comme tous les serveurs sont virtuels et fonctionnent sur une même machine physique, les vulnérabilités que présente cette unique machine sont généralisées à toutes les machines qu'elle héberge. Un autre inconvénient est l'augmentation de la criticité de la machine hôte. Par exemple, une panne matérielle sur la machine physique interromprait l'ensemble des serveurs et des services consolidés sur la machine.

Outre les risques de sécurité et de vulnérabilité, un autre frein conséquent pour les entreprises est le coût d'investissement. Il est vrai que les prix des licences des hyperviseurs

sont très chers. Le prix des licences varie énormément suivant la taille de l'infrastructure mais les prix peuvent facilement dépasser la centaine de millier de francs pour une grosse infrastructure.

### 4.3 Les domaines d'applications

La virtualisation tend à se démocratiser de plus en plus dans les infrastructures informatiques et elle se développe particulièrement dans les six domaines suivants :

1. **La virtualisation des réseaux** est la capacité de créer des réseaux virtuels logiques. Cela garantit une meilleure intégration du réseau et la prise en charge d'environnements de plus en plus virtuels.
2. **La virtualisation de l'espace de stockage** sépare le logiciel de gestion du stockage de l'infrastructure matérielle. Cela permet de fournir plus de flexibilité et d'évolutivité.
3. **La virtualisation des données** est une approche de la gestion des données qui permet à une application de récupérer et de manipuler des données sans avoir à se soucier de la façon dont elles sont formatées ni de l'endroit où elles sont stockées. Cela permet également de fournir une seule vue client des données.
4. **La virtualisation de l'espace de travail** consiste à virtualiser tous les composants du bureau. Un tel système permet la livraison de bureau très flexible et très sécurisé. En effet, aucune donnée n'est stockée sur le périphérique de l'utilisateur, de ce fait, le risque de fuite de données critiques est limité.
5. **La virtualisation des applications** est une technologie qui permet aux utilisateurs d'accéder et d'utiliser une application à partir d'un ordinateur distinct de celui sur lequel l'application est installée.
6. **La virtualisation des serveurs** est le partitionnement d'un serveur physique en serveurs virtuels plus petits pour optimiser les ressources physiquement disponibles.

### 4.4 Comment faire de la virtualisation ?

Il existe deux manières différentes pour créer et gérer les machines virtuelles :

- De manière manuelle
- De manière automatique

#### 4.4.1 Création manuelle

La manière manuelle consiste à répondre à des boîtes de dialogue en mode interactif. Cette façon de faire est certainement la plus intuitif mais la moins recommandée. Le problème du mode manuel est qu'à chaque nouvelle création d'une machine virtuelle, nous devons recommencer le processus comportant un risque élevé d'erreur. Aujourd'hui, il est conseillé d'utiliser la méthode automatique.

#### 4.4.2 Création automatique

La façon de procéder la plus simple pour automatiser la création et la gestion de nos machines virtuelles est de créer un script. En effet, les scripts offrent les avantages suivants :

- Ils sont testables
- La répétabilité
- La traçabilité des modifications grâce à un outil de gestion de versions (Git par exemple)
- Et la documentation. Même sans commentaire, les instructions du script documentent déjà la procédure.

### 4.5 Les clusters

Un cluster consiste à regrouper plusieurs ordinateurs (appelés nœuds) qui forment une seule et même entité. Les objectifs d'un cluster sont les suivants :

- Améliorer la sûreté de fonctionnement
- Faciliter l'évolutivité (scalable)
- Permettre la répartition de charge

L'un des critères d'un cluster est d'avoir des unités de stockage partagées. Pour ce faire, il existe au moins deux manières de faire différentes :

- Avec des baies de stockage et un SAN
- Avec un système de stockage distribué (hyperconvergence)

#### 4.5.1 Baies de stockage et SAN

Un SAN (Storage Area Network) est un réseau permettant de mutualiser des ressources de stockage. Dans le cadre d'un SAN les baies de stockage n'apparaissent pas comme des volumes partagés sur le réseau. En résumé : chaque serveur voit l'espace disque d'une baie SAN auquel il accède comme son propre disque.

Une baie de stockage (disk array) est un équipement comprenant un RAID et un ou plusieurs contrôleurs. Ces contrôleurs assurent la communication avec les serveurs d'application à travers le SAN ainsi que la gestion du RAID.

#### 4.5.2 Le stockage distribué

Utiliser un système de stockage distribué permet de se passer du SAN et de ses baies de stockages. Au lieu de cela, la mémoire de masse installée dans chaque nœud du cluster sera utilisée.

Les avantages sont les suivants :

- Chaque nœud est identique (pas de baies de stockage).
- Il n'y a qu'une seule infrastructure réseau à gérer (pas de SAN).
- L'accès aux unités de stockage est potentiellement plus rapide.

Et voici les désavantages :

- Un minimum de trois nœuds est requis.
- Seulement 50% du stockage est disponible.
- L'évolutivité du stockage est liée à l'évolutivité des ressources de calculs (CPU).

## 5 La virtualisation de serveur

Voici l'objet sur lequel nous allons spécifiquement nous concentrer durant ce module. Comment évoqué plus tôt, la virtualisation de serveur et le partitionnement d'un serveur physique en serveurs virtuels plus petits pour optimiser les ressources du serveur physique. Dans la virtualisation de serveur, le logiciel est utilisé pour subdiviser le serveur physique en plusieurs environnements virtuels, appelés serveurs virtuels ou privés. Cela contraste avec le fait de dédier un serveur physique à une seule application ou service.

Avant la virtualisation des serveurs, il était courant d'avoir des serveurs surutilisés ou sous-utilisés dans le même datacenter. Grâce à la virtualisation, il est possible d'attribuer les ressources à une tâche ou à un service qui en a réellement besoin. De cette façon, le plein potentiel du matériel est exploité. De plus, la virtualisation permet de déplacer des charges de travail entre les machines virtuelles selon la charge, ce qui nous permet de rester flexible.

## 6 Vagrant

Vagrant est un outil aidant à la création et la configuration de machines virtuelles. Il permet l'automatisation et la répétabilité de la création de machines virtuelles identiques. Le but étant de travailler en développement dans le même environnement qu'en production. Vagrant propose de définir dans un simple fichier de configuration, le système utilisé ainsi que la configuration souhaitée pour l'environnement. Les principaux avantages de cet outil sont la simplicité, la portabilité des VM, la légèreté et également la facilité que l'on a à reproduire ou à dupliquer un même environnement plusieurs fois.

Les cas d'utilisation incluent l'exécution de tests, la mise en place rapide d'un environnement de développement ou encore le partage d'environnements de démonstration.

## 6.1 Installation

Pour installer Vagrant, nous pouvons utiliser un gestionnaire de paquets tel que Chocolatey sous Windows ou Homebrew sous Mac OSX. Il est également possible de télécharger l'exécutable depuis le site internet de Vagrant.

## 6.2 Vagrantfile

Le Vagrantfile est le fichier qui contient le script de configuration et de création de la machine virtuelle. Le langage utilisé pour ce script est issu du DSL de Vagrant, lui-même basé sur le langage Ruby.

Pour créer un Vagrantfile par défaut, il suffit de se placer dans le répertoire du projet et de taper la commande suivante : « vagrant init ».

Le script contient notamment :

- Le nom de la box qui contient l'image de base
- Le provider qui permet de piloter l'hyperviseur
- La configuration réseau
- La méthode de provisionning
- Un ou plusieurs scripts de provisionning

## 6.3 Boxes

Vagrant utilise des images prêtes à l'emploi, appelées « base boxes », pour créer une nouvelle machine virtuelle. Une box de base est une archive (ZIP ou tar.gz) qui contient des métadonnées et des informations sous la forme de fichiers JSON ainsi que les fichiers constituant la machine virtuelle.

Un nombre important de boxes de base sont disponible sur le site Vagrant Cloud (<https://app.vagrantup.com/boxes/search>). Il est donc possible de créer de nouvelles boxes à partir d'une box de base téléchargée depuis Vagrant Cloud.

Il est tout à fait possible de créer ses propres boxes de base, notamment à l'aide d'outils comme Packer qui permet de créer une image à partir d'un fichier ISO d'un OS.

## 6.4 Provider

Il est important de rappeler que Vagrant n'est pas un hyperviseur mais une interface permettant d'automatiser les actions d'un hyperviseur. Pour que Vagrant puisse interagir avec l'hyperviseur, il a besoin d'un adaptateur qu'on appelle « provider » dans le jargon de Vagrant. Un provider est donc un adaptateur qui permet à Vagrant d'interagir avec un hyperviseur. Comme chaque hyperviseur est différent, il y a un « provider » différent pour chacun d'eux. Des providers existent, par exemple, pour les hyperviseurs suivants :

- VirtualBox (par défaut)
- VMWare Workstation et ESXi
- Hyper-V
- AWS

## 6.5 Commandes de base

Dans le tableau suivant figurent les quelques commandes de base de Vagrant.

Commandes	Description
Vagrant up [id]	Démarre ou créer la VM spécifiée.
Vagrant halt [id]	Arrête la VM spécifiée.
Vagrant suspend [id]	Suspend la VM spécifiée.
Vagrant resume [id]	Reprend la VM spécifiée.
Vagrant ssh [id]	Ouvre une session SSH pour la VM spécifiée.
Vagrant destroy [id]	Arrête et supprime la VM spécifiée
Vagrant init	Créer un fichier Vagrantfile de base dans le répertoire courant.
Vagrant global-status	Liste l'identifiant et l'état des machines installées.

Une commande Vagrant sans paramètre s'applique à la machine virtuelle définie par le Vagrantfile se trouvant dans le répertoire courant. Pour appliquer une commande à une VM spécifique, il faut donner l'identifiant de la VM en question comme paramètre de la commande.

## 6.6 Configuration basique d'un Vagrantfile

Ci-dessous se trouve un fichier Vagrantfile permettant de créer une machine virtuelle. Ce script crée non seulement la VM mais il exécute aussi un script (voir la ligne 15) et il configure une redirection de port (voir la ligne 17 du script).

```
1  Vagrant.configure("2") do |config|
2
3      # Import la box de base "debian/buster64"
4      config.vm.box = "debian/buster64"
5      # Synchronise le fichier courant du host avec le répertoire /vagrant de la VM.
6      config.vm.synced_folder ".", "/vagrant"
7
8      # Exécute la configuration pour le provider
9      config.vm.provider :virtualbox do |vb|
10         # Paramètre le nom de la vm virtualbox
11         vb.name = "m340-vagrant-exemple"
12     end
13
14     # Exécute le script bootstrap.sh
15     config.vm.provision :shell, path: "bootstrap.sh"
16     # Configure une redirection de port
17     config.vm.network "forwarded_port", guest: 80, host: 8080, id: "Nginx"
18 end
```

Le script qu'exécute le Vagrantfile est le suivant :

```
1  #!/bin/bash
2  # Update the system
3  sudo apt-get update
4  sudo apt-get upgrade
5  # Install Nginx
6  sudo apt-get install nginx -y
7  # Copy html file
8  cp /vagrant/index.html /var/www/html
9  sudo systemctl restart nginx
```

Ce script fait précisément les choses suivantes :

- Il met à jour le système.
- Il télécharge et installe le serveur web Nginx.
- Puis il copie le fichier html que j'ai créé dans le dossier par défaut de Nginx.
- Pour finir, il redémarre le serveur Nginx.

Le fait de pouvoir exécuter un script directement depuis le Vagrantfile permet de pouvoir configurer bon nombre de chose sur la machine virtuelle dès sa création. Cela permet de recréer sa machine sans devoir tout reconfigurer à chaque fois.

## 7 Packer

Packer est un outil qui permet d'automatiser la création d'images de base pour différentes plateformes. Grâce à un seul et même script, il est possible de produire des images identiques pour plusieurs plateformes.

L'intérêt d'un tel outil est de pouvoir créer ses propres images de base pour ensuite les utiliser afin de créer des machines virtuelles. De cette manière, il est possible de créer une image Ubuntu, par exemple, comportant toutes les configurations et les logicielles de base.

Packer supporte une pléthore de plateformes, en voici quelques-unes :

- Cloud: Amazon EC2, Azure, DigitalOcean, OpenStack, CloudStack, etc.
- Hyperviseur: VMWare, vSphere, Proxmox, Hyper-V, VirtualBox, etc.
- Container: Docker, LXD, LXC
- Automatisation : Vagrant ou encore Terraform

### 7.1 Installation

Comme pour Vagrant, il est possible d'installer Packer depuis un gestionnaire de paquet. Packer est disponible sur Chocolatey (Windows) et sur Homebrew (Mac OSX).

Voici les commandes d'installation :

- Windows: *choco install -y packer*
- Mac OSX: *brew cask install packer*

### 7.2 La terminologie

Une certaine terminologie accompagne Packer. En effet, le jargon de la virtualisation et plus particulièrement celui de Packer comprend un certain nombre de termes à connaître. Nous allons donc passer en revue ces différents termes.

#### 7.2.1 Build

Un « build » est la tâche unique accomplie pour créer une image. Plusieurs build peuvent s'opérer en parallèle.

#### 7.2.2 Builder

Un « builder » est un composant de Packer capable de créer une image de machine pour une plateforme. Le builder se charge de prendre la configuration fournie afin de créer une image correspondante à sa plateforme.

Parmi les builders disponibles, on peut mentionner VirtualBox, VMWare et Amazon EC2. D'autres builders peuvent être ajoutés sous la forme de plugins.



### 7.2.3 Artefact

Un artefact est tout simplement le résultat d'une tâche de build. Généralement, un artefact est constitué d'un ensemble de fichiers qui représentent une image de machine virtuelle. Chaque builder produit un seul artefact. Par exemple, le builder Vagrant produit un artefact de type « .box ».

Pour résumer les trois notions vues jusqu'alors, on peut dire qu'un artefact est le résultat d'un build, lui-même initié par un builder.

### 7.2.4 Provisioner

Un provisioner est aussi un composant de Packer qui a pour but d'installer et de configurer la partie logicielle sur une machine virtuelle. Le provisioner entre en action lorsque la machine est déjà en cours d'exécution mais avant que celle-ci ne soit transformée en image statique.

Son rôle est essentiel : s'assurer que l'image contienne tout le logiciel nécessaire.

Voici quelques exemples de provisioners : un script shell, Ansible, Chef, Puppet, etc.

### 7.2.5 Post-processor

Un post-processor est un composant de Packer qui utilise le résultat d'un builder pour en produire un nouvel artefact. L'utilisation la plus fréquente d'un post-processor est lorsque l'on utilise « compress » pour compresser un artefact ou « upload » pour téléverser les artefacts.

### 7.2.6 Modèle (template)

Un modèle (ou template en anglais) est un fichier JSON qui décrit un ou plusieurs builds en fournissant la configuration des différents composants de Packer (provisioner, builder et post-processor).

Packer a la capacité de lire un modèle et de l'utiliser pour créer plusieurs images de machine en parallèle. Un modèle doit au moins contenir la configuration d'un builder.

### 7.2.7 Commandes

L'interface en ligne de commande (CLI) de Packer nous permet d'utiliser un certain nombre de commandes. Chaque commande correspond à l'un des sous-programmes de Packer. Les commandes principales sont répertoriées dans le tableau ci-dessous.

Commande	Description
build	Construit l'image à partir du modèle JSON.
console	Permet de tester les interpolations des variables.
fix	Prend un modèle et trouve les incompatibilités de version et le met à jour.
inspect	Prend un modèle et génère les différents composants définis par ce modèle.
validate	Vérifie et valide la syntaxe et la configuration d'un modèle.

## 7.3 Introduction aux modèles

Nous allons voir un peu plus en détail comment les modèles sont créés. Comme indiqué plus haut, un modèle est un fichier JSON contenant un certain nombre d'attributs permettant de configurer les divers composants de Packer. Seul l'attribut « builders » est requis, les autres sont optionnelles. Les autres attributs pouvant être utilisés sont les suivants :

- description
- min\_packer\_version
- post\_processors
- provisionners
- variables

### 7.3.1 Builders

L'attribut builders est constitué d'un tableau contenant un ou plusieurs objets qui définissent la configuration des builders. Chaque builder est spécifique à une certaine plateforme avec ses propres options et paramètres. Il existe donc un builder différent pour chaque plateforme.

La liste exhaustive de plateformes supportées par Packer est disponible sur leur site :

<https://packer.io/docs/builders/index.html>

### 7.3.2 Description

L'attribut description est tout simplement une chaîne de caractère qui décrit le modèle

### 7.3.3 Min\_packer\_version

Cet attribut a pour but de spécifier la version minimale requise pour analyser le modèle. De cette manière, une version plus vieille de Packer pourra informer l'utilisateur que le modèle ne pourra pas être correctement interprété.

### 7.3.4 Post\_processors

L'attribut « post\_processors » est un tableau d'un ou plusieurs objets qui définit les différentes étapes de post-traitement à effectuer sur l'image produite. Si cela n'est pas spécifié, aucun post-traitement ne sera effectué.

### 7.3.5 Provisionners

Cet attribut se charge de définir, dans un tableau d'un ou plusieurs objets, les provisionners qui seront utilisés pour installer et configurer la partie logicielle de l'image créée. S'il n'est pas spécifié, aucun provisionner ne sera exécuté.

### 7.3.6 Variables

L'attribut variable est un objet qui regroupe toutes les variables utilisateurs utilisés dans le modèle. Si aucun attribut « variables » n'est défini, aucune variable ne sera définie.

## 8 L'approvisionnement (provisioning)

Dans le contexte de la virtualisation de machines, l'approvisionnement désigne deux choses :

1. L'ajout et la suppression de composants logiciels (application, service, fichier de configuration, etc.) sur une machine.
2. Le déploiement d'une ou plusieurs machines virtuelles dans un hyperviseur (vSphere, Proxmox, ...) ou dans un cloud comme OpenStack ou Amazon EC2.

Tous les outils d'approvisionnement dont nous allons parler dans ce document (Ansible et SaltStack) permettent de faire l'un et l'autre. Néanmoins, nous nous concentrerons sur le premier point, soit sur l'ajout et la suppression de composants logiciels.

### 8.1 Automatisation

Il est possible d'automatiser l'approvisionnement grâce à un langage de commandes (bash, powershell, etc.) mais un lot de difficultés accompagne cette pratique :

- Le langage est souvent dépendant de la plateforme.
- Le script d'une installation complexe est souvent difficile à réaliser avec un script.
- Il est compliqué d'assurer que le script a toujours le même effet s'il est exécuté plusieurs fois (l'idempotence).
- La distribution des scripts sur les machines cibles doit aussi être automatisée.

### 8.2 Outils de gestion de configuration

Les outils de gestion de configuration facilitent l'automatisation du provisioning. Nous retrouvons généralement les caractéristiques suivantes chez un outil de ce type :

- Possession d'un outil de gestion pour le déploiement et l'exécution des scripts sur les machines cibles.
- Utilisation d'un langage spécifique qui permet de décrire l'état désiré et qui facilite la prise en charge de différentes plateformes.
- Assurance de l'idempotence des scripts.

Les outils de gestion de configuration ne s'utilisent pas seulement avec des machines virtuelles mais ils sont également utilisés pour gérer la configuration de machines physiques.

### 8.3 Fonctionnement

Certains outils de gestion de configuration fonctionnent avec un agent installé sur chaque machines cibles.

Un agent est un programme qui fonctionne en tâche de fond (démon ou service) et permet les choses suivantes :

- La récupération d'une configuration sur la machine de gestion (**mode pull**). C'est le client qui va demander la configuration à la machine de gestion.
- La réception d'une configuration de la machine de gestion (**mode push**). C'est la machine de configuration qui va envoyer la configuration aux clients.

Les outils n'utilisant pas d'agent sont appelés « agentless » (littéralement : « sans agent »). La machine de gestion utilise donc ssh ou WinRM pour communiquer avec les machines cibles. Cependant, il n'y a que le mode push de possible.

### 8.4 Quand les utiliser ?

Dans notre contexte particulier, les outils de gestion de configuration sont principalement utilisés pour :

- Créer une image de base pour nos machines virtuelles.
- Approvisionner des machines virtuelles existantes

Nous pouvons les utiliser également dans ces cas plus concrets :

- Un script Packer pour créer une image de base.
- Un script Vagrant pour personnaliser une image de base en y ajoutant de l'approvisionnement.

### 8.5 Quelques exemples

Voici une liste non-exhaustive des outils de gestion de configuration :

- Ansible
- Salt (SaltStack)

- Puppet
- Chef
- Mgmt
- Capistrano
- Rudder
- Fabric

Dans ce document, nous allons nous concentrer plus particulièrement sur Ansible et SaltStack.

## 9 Ansible

Ansible est un puissant outil open source d'automatisation, de configuration, de déploiement d'application, etc. Ansible a été créée en 2012 et a ensuite été rachetée par Red Hat. Le but de ce chapitre est d'en savoir un peu plus sur l'étendue de l'offre qu'Ansible propose et sur son fonctionnement.

### 9.1 De quoi Ansible est capable ?

Ansible peut faire énormément de chose dont les points suivants :

- **Gestion de configuration :**  
Ansible a été conçu pour être simple, fiable et cohérent dans la gestion de configuration. Les configurations Ansible sont des descriptions de données simples de l'infrastructure et sont aisément lisibles par les humains et analysables par les machines (fichier YAML). Prenons un exemple : si vous souhaitez installer une mise à jour d'un programme sur toutes les machines de votre entreprise, il vous suffit d'écrire toutes les adresses IP des nœuds (également appelé hôtes distants) et d'écrire un « playbook Ansible » pour l'installer sur tous les nœuds, puis d'exécuter le playbook depuis votre machine de contrôle.
- **Orchestration :**  
Comme son nom l'indique, l'orchestration consiste à rassembler les différentes étapes d'une opération en une seule tâche, parfaitement rodée. Par exemple, dans le cas d'un déploiement d'une application, il y a non seulement le backend et le frontend à gérer mais il y a aussi les bases de données, le réseau, le stockage etc. Toutes ces tâches doivent être exécutées dans le bon ordre. Pour faciliter l'orchestration d'un tel processus, Ansible utilise des flux de travail automatisés, le provisionning et plus encore. Une fois l'infrastructure définie à l'aide des playbooks Ansible, vous pouvez utiliser cette même orchestration partout où vous en avez besoin, grâce à la portabilité des playbooks d'Ansible.
- **Déploiement d'application :**  
Lorsque vous utilisez Ansible pour déployer votre application, vous êtes en mesure

de gérer efficacement le cycle de vie des applications : du développement à la production. Vous n'aurez pas à configurer manuellement les applications sur chaque machine. Il n'y a qu'à dresser la liste des tâches à faire dans un playbook et Ansible se charge d'amener votre système dans l'état voulu. Lorsque vous exécutez un playbook depuis votre machine de contrôle, Ansible utilise SSH pour communiquer avec les hôtes distants et exécuter toutes les commandes (ou tâches) nécessaires.

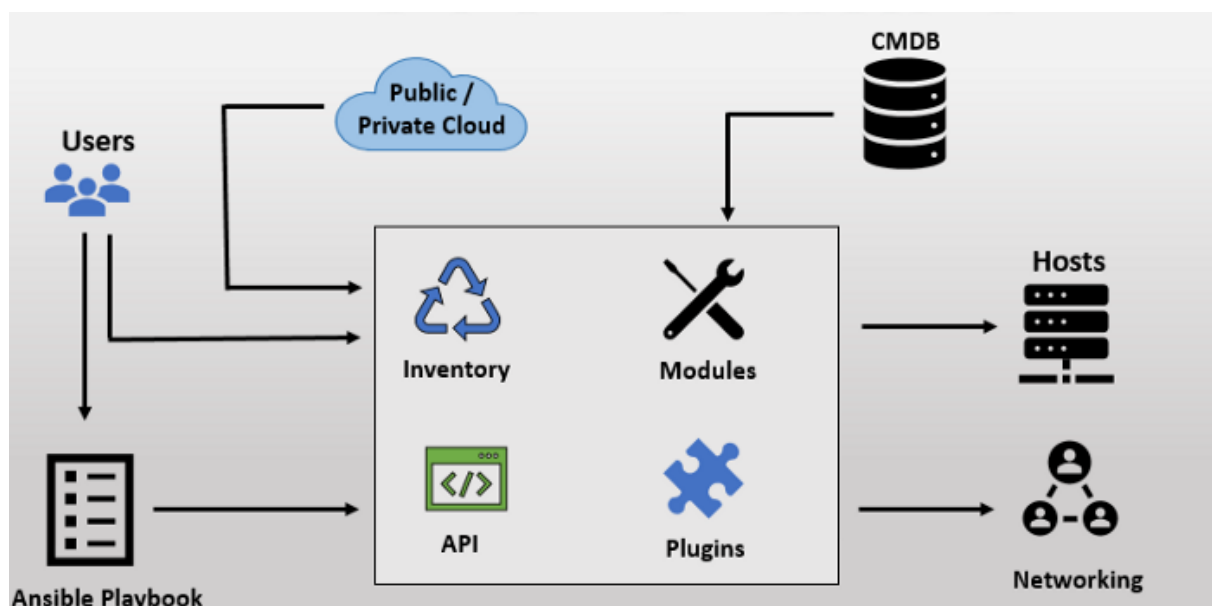
- **Approvisionnement :**

La première étape de l'automatisation du cycle de vie d'une application consiste à automatiser l'approvisionnement de l'infrastructure. Avec Ansible, il est possible de gérer l'approvisionnement de plateformes de « cloud computing », d'hôtes virtualisés, d'équipement réseau et de serveurs « bare-metal ».

- **Sécurité et conformité :**

Lorsqu'une politique de sécurité est définie dans Ansible, elle sera directement intégrée dans tout ce qui est déployé. Cela signifie qu'il n'y a qu'une seule fois à configurer les détails de sécurité, dans une machine de contrôle. De plus, toutes les informations d'identification qui sont stockés dans Ansible ne peuvent être récupérées en clair par aucun utilisateur.

## 9.2 L'architecture



Comme vous pouvez l'observer sur le schéma ci-dessus, l'architecture du moteur d'automatisation d'Ansible s'articule autour de différents composants. Les voici plus en détail :

- **Hôtes :**

Les hôtes dans l'architecture d'Ansible sont simplement des systèmes de nœuds qui sont automatisés par Ansible. Il peut s'agir de n'importe quel type de machine : Windows, Linux, RedHat, etc.

- **Playbooks :**  
Les playbooks sont de simples fichiers écrits au format YAML qui décrivent les tâches à exécuter par Ansible. Les playbooks peuvent déclarer des configurations, mais ils sont également capables d'orchestrer les étapes de tout processus ordonné manuellement. Ils peuvent aussi lancer des tâches de manière synchrone ou asynchrone.
- **Inventaires :**  
Les inventaires d'Ansible sont des listes d'hôtes (nœuds) avec leurs adresses IP, serveurs, base de données, ... qui doivent être gérés.
- **Une APIs :**  
Les APIs dans Ansible sont utilisés comme moyen de transport pour les services du Cloud, publics ou privés.
- **Modules :**  
Les modules sont exécutés directement sur les hôtes distants via des playbooks. Les modules peuvent contrôler des ressources système (services, paquets ou des fichiers) ou exécuter des commandes systèmes. Ils le font en agissant sur les fichiers système, en installant des paquets ou en faisant des appels aux APIs du réseau de services.
- **Plugins :**  
Les plugins sont des morceaux de code qui ajoute une fonctionnalité à Ansible. Ansible est livré par défaut avec un certain nombre de plugins pratiques mais vous pouvez facilement écrire les vôtres.
- **CMDB :**  
Il s'agit simplement d'une base de données qui contient des données relatives à une collection de biens informatiques ainsi que des données permettant de décrire les relations entre ces biens.
- **Cloud :**  
Il s'agit d'un réseau de serveurs distants hébergés sur Internet pour stocker, gérer et traiter des données. Il est possible de lancer des ressources sur le cloud et de se connecter à ses serveurs distants. Cela permet de ne pas avoir à le faire sur un serveur local.

### 9.3 Les avantages d'Ansible

Voici les principaux avantages d'Ansible :

- C'est gratuit et open-source.
- Il est assez simple à configurer et à utiliser.
- C'est un outil puissant qui permet de faire beaucoup de chose.
- Ansible est flexible.
- Il est autosuffisant dans le sens qu'Ansible ne requiert aucune installation de logiciels supplémentaires chez les clients. Cela le rend aussi plus efficace car il y a plus de place pour les ressources applicatives sur les serveurs.

## 10 SaltStack

SaltStack est un logiciel open source rendu public pour la première fois en 2011. Tout comme Ansible, c'est un outil de gestion de configuration utilisé pour la gestion et la surveillance automatisés des systèmes de serveurs. Grâce à SaltStack, il est possible d'installer et de configurer un logiciel à partir d'un ordinateur central et d'exécuter des commandes de configuration en nombre illimité.

Dans ce chapitre nous en apprendrons plus sur ces caractéristiques, son fonctionnement ainsi que sur ses avantages.

### 10.1 Les caractéristiques

SaltStack peut être utilisé sur plusieurs plateformes. Il permet la maintenance à distance, la création de statuts cibles prédéfinis et le lancement de contrôles dans votre propre centre de données ou dans un Cloud externe.

La principale caractéristique de ce logiciel est la polyvalence de ses applications ainsi que sa rapidité.

Pour communiquer entre les clients et les serveurs, SaltStack utilise la bibliothèque d'échange de messages asynchrone « ZeroMQ ». Cette bibliothèque permet de distribuer rapidement des petites et grandes quantités de données. De plus, la sécurité des communications client-serveur est assurée par le protocole de chiffrement symétrique AES. Pour finir, la fonction RAET (Reliable Asynchronous Event Transport) offre une sécurité supplémentaire lors du transport de données.

Les configurations de SaltStack se font via un fichier texte au format YAML. Comme SaltStack est écrit en Python, il est possible d'utiliser ce langage pour combiner de nombreuses commandes de configuration prêtes à être exécutées.

### 10.2 Le fonctionnement

Les clients sont approvisionnés en commandes par le serveur centrale (appelé « Salt-Master ») via des bus tels que ZeroMQ, SSH ou Proxy-Minion. Les clients font leur rapport avec une clé individuelle. Ensuite, le Salt-Master doit l'accepter lors du premier contact afin que la commande puisse alors être exécutée. La communication chiffrée avec les paires de clés a donc lieu.

Prenons un cas concret : imaginons que chaque client du parc informatique a besoin d'avoir Office d'installé. Pour ce faire, il nous faut donner l'ordre au Salt-Master d'installer Office sur tous les clients. Il se chargera lui-même de transmettre cet ordre aux clients. Le logiciel sera ensuite automatiquement configuré sur tous les PC du parc.



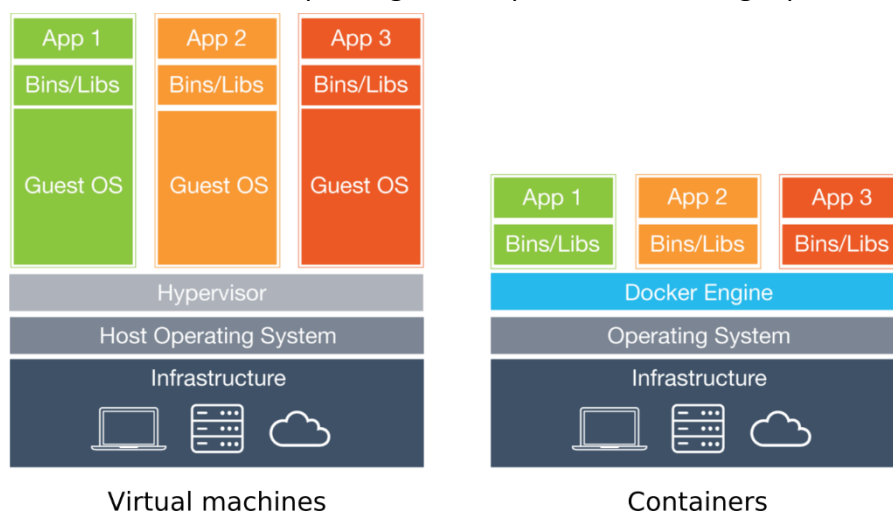
## 10.3 Les avantages

Les avantages de SaltStack sont les suivants :

- **Fonctionnement simple :**  
En effet, quel que soit le système cible que les administrateurs veulent modifier, les commandes SaltStack restent les mêmes. Cela simplifie le fonctionnement et facilite à maîtriser.
- **Répartition automatisée des configurations :**  
En effet, l'outil assure non seulement la répartition automatisée des configurations mais il peut aussi réagir aux événements, car il enregistre le type de communication échangé dans l'infrastructure.
- **Disponible à tous :**  
SaltStack est totalement inclus dans Salt Open et disponible pour tous sous la licence Apache 2.0. Une version entreprise est également disponible.
- **Robustesse :**  
SaltStack est un outil puissant et robuste compatible avec de nombreux systèmes.
- **Sécurisé :**  
SaltStack gère simplement les paires de clés SSH pour authentifier les communications. Il protège également les données sensibles à l'aide d'un protocole chiffré.

## 11 Les containers

Les containers sont une autre solution pour virtualiser une infrastructure. Le concept est le suivant : Un container permet d'isoler une application du monde extérieur sans avoir besoin d'un système d'exploitation dédié et d'un logiciel lourd pour simuler les composants physiques du serveur. Le nombre de couche est donc réduit par rapport à une machine virtuelle. Ainsi, les containers sont plus légers et rapides au démarrage qu'une VM.

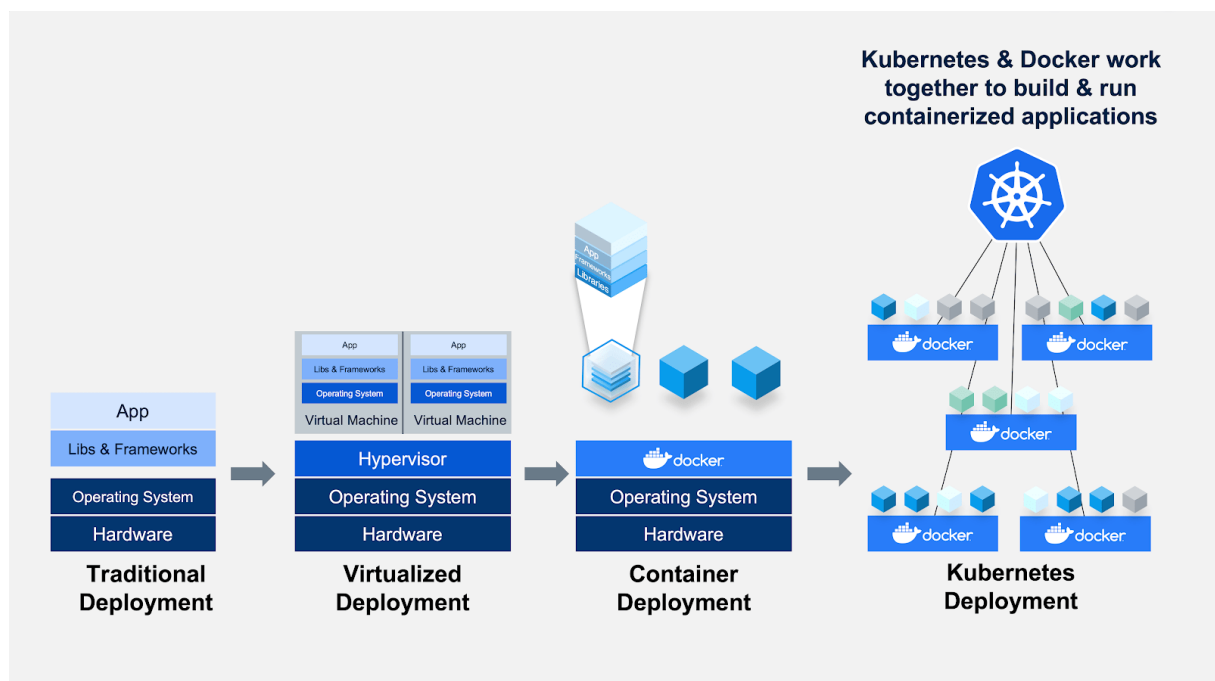


La légèreté et la rapidité des containers nous permettent de diviser une application en plusieurs services sous-jacents communiquant entre eux. Chaque service est donc isolé des autres, fonctionnant dans son propre container. Cela simplifie la gestion de l'application car un service spécifique peut être réparé, redémarré ou recréé sans impacter les autres services de l'application.

Un exemple d'outil très utilisé pour la conteneurisation est Docker avec son outil d'orchestration propre : Docker Compose.

L'orchestration consiste à gérer les containers d'une infrastructure. L'orchestrateur intervient entre le système d'exploitation et l'outil de conteneurisation (Docker par exemple). Kubernetes est un orchestrateur très répandu créé à l'origine par Google dans le but de gérer les millions de container que comportait leur infrastructure. Un orchestrateur permet de guider le déploiement de conteneurs, d'automatiser les mises à jour, la surveillance d'état et bien plus.

Voici un schéma résumant les différentes manières de faire de la virtualisation :



## 12 Exercice 1 : Vagrant

Lors d'une activité, nous devons créer plusieurs machines virtuelles. Selon la consigne, les noms des machines devaient être « m340-nginx-1 » et « m340-nginx-2 ». Pour réaliser cela, j'ai choisi de réaliser une structure de boucle qui créera mes deux machines virtuelles.

Voici le Vagrantfile correspondant :

```

1  Vagrant.configure("2") do |config|
2    ... config.vm.box = "debian/buster64"
3    ... config.vm.synced_folder ".", "/vagrant"
4    ... #dns-resolving-name-issue-solution
5    ... config.vm.provider :virtualbox do |vb|
6    ... |... vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
7    ... end
8    ... (1..2).each do |i|
9    ... |... config.vm.define "m340-nginx-#{i}" do
10   ... |... |... config.vm.provision :shell, path: "bootstrap.sh"
11   ... |... end
12   ... end
13 end

```

Le script « bootstrap.sh » est exactement le même que celui du chapitre précédent à savoir :

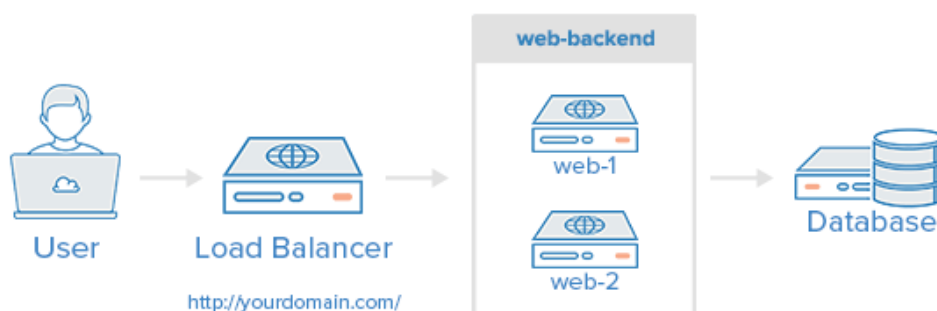
```

1  #!/bin/bash
2  # Update the system
3  sudo apt-get update
4  sudo apt-get upgrade
5  # Install Nginx
6  sudo apt-get install nginx -y
7  # Copy html file
8  cp /vagrant/index.html /var/www/html
9  sudo systemctl restart nginx

```

## 12.1 Mettre en place un répartiteur de charge

Pour cette deuxième partie de l'activité, nous devons modifier notre fichier Vagrantfile afin que nos deux machines virtuelles Nginx soient en backend et ajouter une nouvelle VM en frontend. Cette nouvelle machine remplira le rôle d'un répartiteur de charge (load balancing) avec HAProxy installé et configuré.



Le répartiteur de charge fera du « round robin » ce qui veut dire qu'il redirigera une requête sur deux vers le serveur web 1 et le restant des requêtes iront vers le serveur web 2.

### 12.1.1 Le Vagrantfile

J'ai commencé par modifier le fichier Vagrantfile de tel sorte qu'il puisse créer les trois machines virtuelles demandées. J'ai gardé la boucle pour créer les deux serveurs backend et j'ai rajouté la définition d'une troisième VM pour le frontend.

Comme les trois machines virtuelles devront communiquer ensemble, elles doivent donc avoir une adresse IP privé dans le même réseau. La machine frontend possédera une seconde interface réseau en « bridge » doté d'une adresse IP publique, attribué via DHCP.

Je précise que les deux serveurs backend n'ont pas besoin de redirection du port 80 car on ne leur enverra plus directement les requêtes. Une redirection de port n'est requise que pour la machine frontend car c'est par elle que les requêtes du client transiteront vers les servers backend.

Voici le fichier Vagrantfile en entier :

```
1  Vagrant.configure("2") do |config|
2
3      config.vm.box = "debian/buster64"
4      config.vm.synced_folder ".", "/vagrant"
5
6      #dns resolving name issue solution
7      config.vm.provider :virtualbox do |vb|
8          vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
9      end
10
11      (1..2).each do |i|
12          config.vm.define "m340-nginx-#{i}" do |node|
13              node.vm.hostname = "backend#{i}"
14              node.vm.provision :shell, path: "bootstrap.sh"
15              node.vm.network "private_network", ip: "192.168.2.#{i}"
16              node.vm.network "forwarded_port", guest: 80, host: 8080, disabled: true
17              node.vm.provider :virtualbox do |vb|
18                  vb.name = "m340-nginx-#{i}"
19              end
20          end
21      end
22
23      config.vm.define "m340-nginx-frontend" do |frontend|
24          frontend.vm.hostname = "frontend"
25          frontend.vm.provision :shell, path: "frontend.sh"
26          frontend.vm.network "forwarded_port", guest: 80, host: 8080, id: "proxy"
27          frontend.vm.network "private_network", ip: "192.168.2.3"
28          frontend.vm.network "public_network"
29          frontend.vm.provider :virtualbox do |vb|
30              vb.name = "m340-nginx-frontend"
31          end
32      end
33  end
```

### 12.1.2 Installation et configuration de HAProxy

Pour installer et configuré HAProxy sur le serveur frontend j'ai procédé de la même manière qu'avec les machines backend, c'est-à-dire avec un script. Le script est exécuté lors de l'installation de la machine par Vagrant. Voici son contenu :

```
1  #!/bin/bash
2
3  sudo apt-get update
4  sudo apt-get upgrade
5
6  sudo apt install haproxy -y
7  cp -f /vagrant/haproxy.cfg /etc/haproxy/haproxy.cfg
8  sudo systemctl restart haproxy.service
```

Voici ce qu'il fait précisément :

- Premièrement, il met à jour le système.
- Ensuite il installe le paquet « haproxy ».
- Puis il copie le fichier de configuration du haproxy au bon endroit sur la VM.
- Pour finir, il redémarre le service haproxy.

Après avoir installé HAProxy il ne reste plus qu'à le configurer. Pour ce faire, j'ai décidé de d'abord éditer le fichier de configuration « haproxy.cfg » en local, sur ma machine Windows, pour ensuite le copier au bon endroit sur la machine frontend.

```
1  global
2      log /dev/log      local0
3      log /dev/log      local1 notice
4      chroot /var/lib/haproxy
5      stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
6      stats timeout 30s
7      user haproxy
8      group haproxy
9      daemon
10
11     ca-base /etc/ssl/certs
12     crt-base /etc/ssl/private
13
14     ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS
15     ssl-default-bind-options no-sslv3
16
17  defaults
18      log global
19      mode http
20      option httplog
21      option dontlognull
22      timeout connect 5000
23      timeout client 50000
24      timeout server 50000
25      errorfile 400 /etc/haproxy/errors/400.http
26      errorfile 403 /etc/haproxy/errors/403.http
27      errorfile 408 /etc/haproxy/errors/408.http
28      errorfile 500 /etc/haproxy/errors/500.http
29      errorfile 502 /etc/haproxy/errors/502.http
30      errorfile 503 /etc/haproxy/errors/503.http
31      errorfile 504 /etc/haproxy/errors/504.http
32
33  frontend http
34      bind *:80
35      default_backend web-backend
36
37  backend web-backend
38      balance roundrobin
39      server srv1 192.168.2.1:80
40      server srv2 192.168.2.2:80
```

La seule partie du fichier de configuration que j'ai modifié et la partie encadrée en rouge. La première partie consiste à définir le frontend.

A l'aide de la commande « `bind *:80` » j'indique au proxy d'écouter le port 80 de toutes les adresses IP qu'il possède. La ligne du dessous sert à spécifier le backend par défaut.

Les lignes 37 à 40 concernent la définition des serveurs backend. Premièrement, on définit la méthode de répartition de charge que l'on veut utiliser, dans notre cas c'est « roundrobin ». Ensuite, il faut spécifier tous les serveurs web avec leur adresse IP ainsi que leur port d'écoute.

Cela étant fait, il ne reste plus qu'à essayer. Pour vérifier que tout fonctionne, il faut entrer l'adresse IP publique de l'interface réseau du frontend dans un navigateur. Si tout se passe comme prévu, la requête est redirigée vers un des serveurs web. Ce serveur doit nous renvoyer ce que nous demandons : un fichier HTML étant à la racine du site.

## 13 Exercice 2 : Packer

Cette activité concernant Packer est composée de trois parties différentes. Nous allons donc parcourir les trois parties de cette activité dans ce chapitre.

### 13.1 Installation de Packer

Cette première partie consistait à installer Packer et à créer notre première image. Pour installer Packer, j'ai utilisé Chocolatey. Pour créer une première image, l'école nous a mis à disposition un modèle JSON déjà fait et téléchargeable via le Gitlab de l'école.

Une fois le fichier télécharger, j'ai lancé la commande suivante pour démarrer un build : « `packer build -var 'version=1.0.0' debian10.json` ». Cependant une première erreur apparut. Cette erreur stipulait que la version de Packer n'était pas la même. J'ai donc utilisé « `packer fix` » en créant un nouveau fichier. En comparant les deux fichiers, j'ai remarqué que le champs « `iso_checksum_type` » n'existait pas dans le nouveau fichier. Effectivement dans la nouvelle version de packer, l'iso checksum type est directement compris dans le champs « `iso_checksum` ».

À la deuxième tentative du lancement du build, une deuxième erreur apparut : la version de l'OS que le fichier JSON devait charger n'était pas à jour. Je suis donc allé sur le site « <http://cdimage.debian.org> » pour trouver la nouvelle version. Pour que cela fonctionne, j'ai également dû mettre le « `ISO_SUM` » à jour avec l'ISO\_SUM correspondant à la nouvelle version.

Une fois tous ces problèmes réglés et l'image créée, j'ai pu modifier le fichier Vagrant avec le nom de la box puis faire un « `vagrant up` » pour démarrer la machine virtuelle. Cela a fonctionné du premier coup.

## 13.2 Modification de la configuration

Cette deuxième partie de l'exercice consistait à modifier le fichier JSON pour que la machine virtuelle ait 2 CPUS, 2048 MB de RAM et 10 GB d'espace disque. Voici ce que j'ai dû modifier :

- La variable « VM\_DISK\_SIZE » avec la valeur « 10240 »
- Le champ « vboxmanage » modifié comme suit :

```
"vboxmanage": [
  ["modifyvm", "{{.Name}}", "--memory", "2048"],
  ["modifyvm", "{{.Name}}", "--cpus", "2"]
],
```

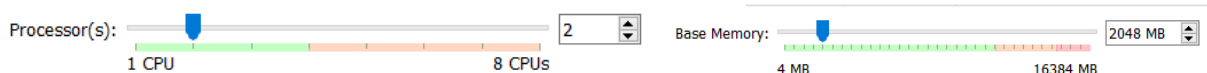
Nous devons également ajouter un script d'installation et de configuration du serveur web Nginx. J'ai donc créé un fichier « nginx.sh » contenant les lignes suivantes :

```
1 sudo apt-get install nginx -y
2 echo "==> Installing Nginx"
```

Ensuite, j'ai rajouté le script dans le champ « scripts » :

```
"scripts": [
  "scripts/base.sh",
  "scripts/virtualbox.sh",
  "scripts/vagrant.sh",
  "scripts/dep.sh",
  "scripts/cleanup.sh",
  "scripts/zerodisk.sh",
  "scripts/nginx.sh"
],
```

Pour être sûr que la configuration soit correcte, j'ai fait un build puis j'ai créé une machine virtuelle avec Vagrant que j'ai démarré. Toutes les modifications apportées étaient bien là.



## 13.3 Adaptation à Ubuntu

Pour terminer cette activité, nous devons modifier le modèle en JSON que l'on nous avait donné. A l'origine, il était prévu pour créer des machines Debian et le but de cette dernière partie était justement d'adapter ce fichier à la création d'une machine Ubuntu.

Pour ce faire, j'ai modifié les choses suivantes :

- Le nom de la VM.
- Le type de la VM.
- L'iso\_url : j'ai mis le lien de la nouvelle image que Packer va devoir utiliser.
- L'iso\_sum pour qu'il corresponde à la nouvelle image.

- Voici le résultat des modifications en image :

```
"variables": {
  "ISO_SUM": "caf3fd69c77c439f162e2ba6040e9c320c4ff0d69aad1340a514319a9264df9f",
  "ISO_URL": "http://releases.ubuntu.com/20.04/ubuntu-20.04-live-server-amd64.iso",
  "VBOX_VERSION": "6.0.14",
  "VM_DISK_SIZE": "10240",
  "VM_NAME": "ubuntu20",
  "VM_TYPE": "Ubuntu_64"
}
```

J'ai déjà essayé de lancer un build rien qu'avec ces modifications pour voir si cela suffisait mais non, cela n'a pas fonctionné.

Comme cela n'a pas fonctionné, j'ai donc cherché des solutions sur Internet et à force de voir des fichiers JSON assez différents, je me suis dit qu'il fallait peut-être complètement changer de fichier JSON. J'ai alors copié un fichier JSON prévu pour Ubuntu et je l'ai modifié pour qu'il corresponde à ce que je voulais. (2)

[illegible]



J'ai relancé un build mais après quelques minutes, une nouvelle erreur est apparue :

```
Build 'virtualbox-iso' errored after 2 minutes 51 seconds: Packer experienced an authentication error when trying to connect via SSH. This can happen if your username/password are wrong. You may want to double-check your credentials as part of your debugging process. original error: ssh: handshake failed: ssh: unable to authenticate, attempted methods [none password], no supported methods remain

==> Wait completed after 2 minutes 51 seconds

==> Some builds didn't complete successfully and had errors:
--> virtualbox-iso: Packer experienced an authentication error when trying to connect via SSH. This can happen if your username/password are wrong. You may want to double-check your credentials as part of your debugging process. original error: ssh: handshake failed: ssh: unable to authenticate, attempted methods [none password], no supported methods remain

==> Builds finished but no artifacts were created.
```

En bref, Packer n'arrive pas à ouvrir une session SSH avec la VM.

J'ai alors cherché où était configuré l'utilisateur et le mot de passe SSH pour la VM. C'est en parcourant le fichier JSON que j'ai vu un fichier que je ne connaissais pas « buster-preseed.cfg » : c'est dans ce fichier que sont configurés les identifiants SSH. Après avoir vérifié que le nom d'utilisateur et le mot de passe était le même dans les fichiers « ubuntu20-04.json » et « buster-preseed.cfg » j'ai relancé un build mais rien n'a changé.

Comme pour le fichier JSON, je me suis dit qu'il fallait aussi un fichier .cfg qui soit spécifiquement conçu pour Ubuntu. Alors j'ai trouvé un fichier « preseed.cfg » sur Internet pour Ubuntu. (2)

```
http > ⚙ ubuntu-preseed.cfg
1 d-i debian-installer/language string fr_CH
2 d-i debian-installer/locale string fr_CH.UTF-8
3 d-i localechooser/preferred-locale string fr_CH.UTF-8
4 d-i localechooser/supported-locales fr_CH.UTF-8
5 d-i netcfg/get_hostname string ubuntu
6 d-i netcfg/get_domain string local
7 d-i console-setup/ask_detect boolean false
8 d-i keyboard-configuration/xkb-keymap select fr_CH
9 d-i time/zone string UTC
10 d-i clock-setup/utc-auto boolean true
11 d-i clock-setup/utc boolean true
12 tasksel tasksel/first multiselect standard, ubuntu-server
13 d-i mirror/http/proxy string
14 d-i pkgsel/install-language-support boolean false
15 d-i pkgsel/update-policy select none
16 d-i pkgsel/upgrade select full-upgrade
17 d-i pkgsel/include string openssh-server cryptsetup build-essential libssl-dev libreadline-dev zlib1g-dev
18 d-i partman-auto/method string lvm
19 d-i partman-auto-lvm/guided_size string max
20 d-i partman-auto/choose_recipe select atomic
21 d-i partman-auto/disk string /dev/sda
22 d-i partman-lvm/confirm boolean true
23 d-i partman-lvm/confirm_nooverwrite boolean true
24 d-i partman-lvm/device_remove_lvm boolean true
25 d-i partman/confirm_nooverwrite boolean true
26 d-i partman/confirm boolean true
27 d-i partman/confirm_write_new_label boolean true
28 d-i partman/choose_partition select finish
29 d-i passwd/user-fullname string vagrant
30 d-i passwd/username string vagrant
31 d-i passwd/user-password password vagrant
32 d-i passwd/user-password-again password vagrant
33 d-i user-setup/allow-password-weak boolean true
34 d-i user-setup/encrypt-home boolean false
35 d-i passwd/user-default-groups vagrant sudo
```

Je l'ai de nouveau adapté pour qu'il corresponde au fichier JSON et à ce que je voulais. Une fois toutes les modifications effectuées, je relance un build et... de nouveau la même erreur.

Après de multiples tentatives, modifications, relances de build et plusieurs heures passées à rechercher une solution pour ne toujours rien trouver, je me suis résolu à arrêter là pour ne pas prendre de retard sur la suite du cours.

## 14 Exercice 3 : SaltStack

Le but de cette activité est de découvrir SaltStack en suivant le tutoriel de SaltStack. Ce tutoriel nous apprendra les fondamentaux de SaltStack. (3)

### 14.1 Installation

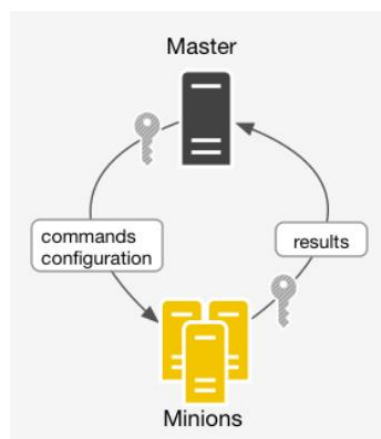
Avant d'installer SaltStack, jetons d'abord un œil aux prérequis. Pour créer un environnement de test de SaltStack nous devons avoir Virtualbox et Vagrant d'installés. Suite à cela, nous pouvons faire un « git clone » du repository GitHub contenant l'environnement de test. Ceci étant fait, il ne nous reste plus qu'à nous déplacer dans le dossier courant où le repository se trouve et entrer la commande suivante pour créer nos machines virtuelles « vagrant up ».

Après une dizaine de minutes environ, notre environnement de test est opérationnel et nous pouvons enfin passer à la suite.

### 14.2 Tutoriel de base

Dans ce tutoriel, nous découvrons l'architecture de SaltStack. Cette architecture se compose du « Salt master » qui est en réalité le serveur depuis lequel les commandes seront envoyées et des « Minions » qui ne sont autres que les clients.

Chaque connexion entre le « Salt master » et un « minion » est sécurisée par des clés cryptographiques. Après l'installation, chaque minion envoie sa clé publique au Salt Master et attend que la connexion soit acceptée. Les clés de chaque minion doivent être acceptées avant que le minion reçoive des commandes du master.



## 14.3 Les commandes de base

Avant de nous plonger dans les commandes de base, observons, dans un premier temps, la syntaxe des commandes :



La target (ou cible) : détermine sur quels systèmes va s'appliquer la commande.

La commande (module.function) : Les commandes se composent d'un module et d'une fonction. SaltStack est livré avec des modules intégrés qui permettent de faire la plupart des tâches à automatiser.

Les arguments : Définis les données supplémentaires nécessaires à l'exécution de la commande.

Grâce à ce tutoriel de base, nous apprenons les commandes de base de SaltStack. Voici les plus importantes :

Commande	Description
<code>salt-key --accept=&lt;key&gt;</code>	Accepte la clé d'un minion spécifique.
<code>salt-key --accept-all</code>	Accepte toutes les clés qui attendent d'être acceptées.
<code>salt '*' test.ping</code>	Envoi la commande « test.ping » sur tous les minions.
<code>salt '*' cmd.run 'ls -l /etc'</code>	Exécute une commande shell sur tous les minions.
<code>salt '*' disk.usage</code>	Renvoie l'utilisation du disque de chaque minion.
<code>salt '*' pkg.install cowsay</code>	Installe le paquet « cowsay » sur tous les minions.
<code>salt '*' network.interfaces</code>	Liste toutes les interfaces réseaux de tous les minions.
<code>salt 'minion1' disk.usage</code>	Exécute la commande sur le minion nommé « minion1 ».
<code>salt 'minion*' disk.usage</code>	Exécute la commande sur tous les minions dont le nom commence par « minion ».
<code>salt -G 'os:Ubuntu' test.ping</code>	Exécute la commande sur tous les minions dont le système d'exploitation est Ubuntu.
<code>salt -E 'minion[0-9]' test.ping</code>	Exécute la commande sur tous les minions dont le nom se compose de « minion » suivi d'un chiffre entre 0 et 9.

<code>salt -L 'minion1,minion2' test.ping</code>	Exécute la commande sur les minions définis dans la liste.
<code>salt -C 'G@os:Ubuntu and minion*' or S@192.168.50.*' test.ping</code>	Exécute la commande sur les minions dont les critères correspondent à ce qui est définis.

## 14.4 Les salt state

L'approvisionnement permet de gagner du temps mais elle présente aussi quelques inconvénients. En effet, la plupart des tâches que l'on effectue sont une combinaison de plusieurs commandes, tests et opérations. On tente souvent de combiner toutes ces étapes dans un script shell mais celui-ci devient rapidement difficile à manier.

C'est pourquoi SaltStack permet la création de modèles de configuration réutilisables appelés state. Le but de ces modèles est de décrire tout ce qui est nécessaire pour mettre un composant système ou une application dans une configuration connue. Les states sont décrits en YAML et simples à créer et à lire.

Voici, ci-dessous, un exemple d'un state contenu dans un fichier « nettools.sls » :

```
install_network_packages:
  pkg.installed:
    - pkgs:
      - rsync
      - lftp
      - curl
```

Pour appliquer ce state au minion 2 par exemple, il suffit d'entrer la commande suivante depuis le master :

```
salt 'minion2' state.apply nettools
```

## 15 Exercice 4 : Proxmox

Tout d'abord, définissons ce qu'est Proxmox. Proxmox Virtual Environment est une solution de virtualisation basé sur l'hyperviseur KVM. Il offre également une solution de containers avec LXC. Voici ces principales caractéristiques :

- Propose deux types de virtualisation :
  - La virtualisation matérielle avec KVM. Cela permet de virtualiser tout système d'exploitation.
  - La virtualisation par container avec LXC. Ce type de virtualisation permet de créer des instances de système d'exploitation isolées et n'est disponible que sous Linux.
- Gratuit avec la possibilité de payer pour bénéficier du support.

Concernant cette activité, nous devons créer un cluster avec trois machines virtuelles et Proxmox dans le but de découvrir les fonctionnalités offertes par une telle solution. Les prochains chapitres apportent quelques détails quant à la réalisation de cette activité.

## 15.1 Installer Proxmox

La première partie de cette consistait tout simplement à installer une machine virtuelle avec Proxmox. Le but étant de comprendre comment installer Proxmox avant d'automatiser cela.

Tout d'abord, il faut télécharger l'image ISO de Proxmox sur le site internet que voici :

<https://www.proxmox.com/en/downloads/category/iso-images-pve>

Il nous faut aussi une machine virtuelle sur VirtualBox. Voici le minimum requis pour installer Proxmox sur une machine :

- CPU 64bits
- 1GB de RAM
- 1 disque dur
- 1 carte réseau

J'ai donc créé une machine virtuelle, à la main, correspondant à ces prérequis. Une fois la machine créée, il faut y insérer l'ISO précédemment téléchargé puis démarrer la VM. L'installation de Proxmox n'est pas compliqué. Il suffit de cliquer sur « suivant », d'entrer le mot de passe lorsque demandé ainsi qu'une adresse mail et c'est fait. Proxmox étant installé, il ne nous manque plus qu'à paramétrer l'accès à la console web.

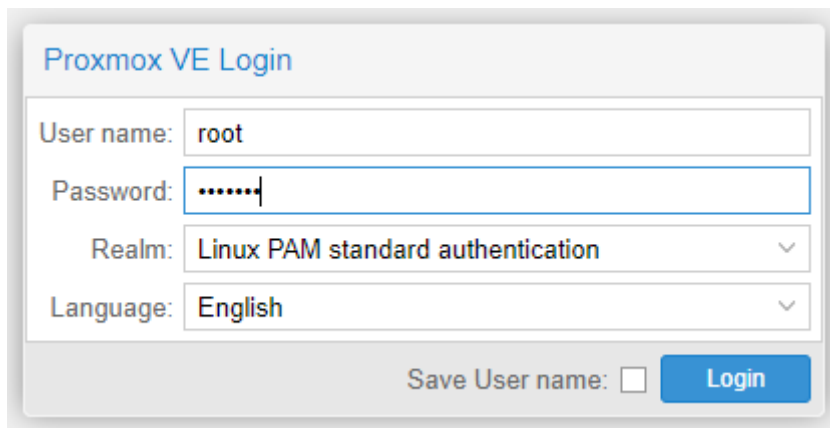
Comme la carte réseau de notre machine virtuelle est en NAT, une des solutions est de créer une redirection de port. Le concept consiste à rediriger les paquets allant sur la carte réseau dédié à VirtualBox de notre PC vers notre machine virtuelle. Pour créer une redirection de port, il faut aller dans les paramètres de la VM, dans « Network », dans « Advanced », puis dans « Port forwarding ». Il suffit maintenant d'ajouter la redirection de port suivante en cliquant sur le bouton vert :

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Proxmox	TCP	192.168.56.1	8006	10.0.2.15	8006

De cette manière, nous pouvons accéder à la console web via l'URL suivante :

<https://192.168.56.1:8006>

Le nom d'utilisateur par défaut est « root » et le mot de passe est celui que l'on a définis durant l'installation.

A screenshot of the Proxmox VE Login interface. It features a title 'Proxmox VE Login' in blue. Below the title are four input fields: 'User name:' with 'root' entered, 'Password:' with masked characters '.....', 'Realm:' with 'Linux PAM standard authentication' selected from a dropdown, and 'Language:' with 'English' selected from a dropdown. At the bottom right, there is a checkbox labeled 'Save User name:' which is unchecked, and a blue 'Login' button.

## 15.2 Création de la basebox

Avant de procéder à la création de la basebox avec la dernière version de Proxmox, j'ai déjà regardé sur le Vagrant Cloud s'il y avait une basebox existante. Malheureusement, aucune box n'existait avec la dernière version de Proxmox.

Je me suis donc appuyé sur un repository GitHub pour créer ma propre box. Voici le repository en question :

<https://github.com/sticky-note/proxmox-ve>

J'ai repris le fichier « .json » et je l'ai adapté avec un nouvel iso et le checksum correspondant. Après avoir finis les modifications et après avoir créé ma basebox, j'ai décidé de mettre mon code sur mon GitHub et ma basebox sur Vagrant Cloud car cela n'existait pas encore. Donc voici les liens du repository GitHub que j'ai créé ainsi que celui du Vagrant Cloud :

<https://github.com/hsmnn/proxmox-ve>

<https://app.vagrantup.com/hsmnn/boxes/proxmox-ve>

## 15.3 Création des machines virtuelles

Pour créer mes machines virtuelles, j'ai bien entendu utilisé Vagrant. Comme le but est de créer trois machines virtuelles pour créer un cluster, j'ai donc fait une boucle dans mon Vagrantfile qui crée les trois machines. Le Vagrantfile s'occupe aussi de faire directement la redirection de port. De cette manière, une fois les machines créées, leur console web est directement accessible sans autres manipulations à effectuer.

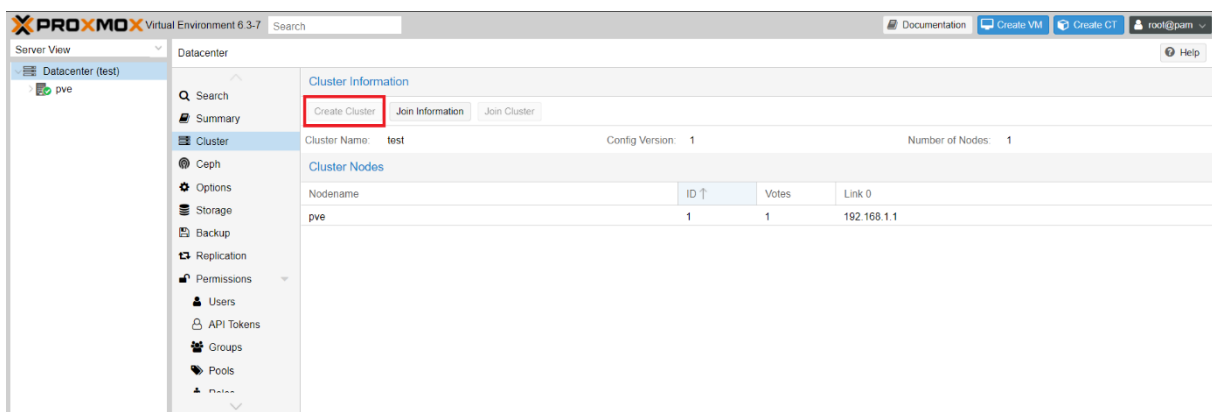
Voici une capture d'écran du Vagrantfile :

```
1 Vagrant.configure("2") do |config|
2   config.vm.box = "packer/output/proxmox-ve-amd64-virtualbox.box"
3
4   (1..3).each do |i|
5     config.vm.define "proxmox-node-#{i}" do |node|
6       node.vm.hostname = "node#{i}"
7       node.vm.network "forwarded_port", guest: 8006, host: 8006, auto_correct: true
8       node.vm.provider :virtualbox do |vb|
9         vb.name = "proxmox-node-#{i}"
10      end
11    end
12  end
13 end
```

## 15.4 La mise en cluster

Maintenant que nous avons les trois machines virtuelles de créées, le prochain objectif est d'en faire un cluster.

Pour créer un cluster, il faut aller dans l'onglet « Datacenter », cliquer sur « Cluster » puis sur « Create cluster ».



Une fenêtre s'ouvre alors et demande d'entrer le nom du cluster puis le réseau du cluster. Ensuite, il faut cliquer sur « Create ».

The screenshot shows the 'Create Cluster' dialog box. It has a title bar with a close button. The form contains two main input fields: 'Cluster Name:' with a text input field, and 'Cluster Network:' which includes a 'Link:' dropdown set to '0' and a text input field containing '192.168.1.2'. Below these fields is an 'Add' button and a note: 'Multiple links are used as failover, lower numbers have higher priority.' At the bottom of the dialog, there is a 'Help' button with a question mark icon and a 'Create' button.

Au début, il m'était impossible de passer cette étape car une erreur arrivait : « Connection refused ». J'ai cherché un moment une solution et finalement, c'est en jetant un œil au Vagrantfile d'un collègue que j'ai compris mon erreur. En effet, dans mon Vagrantfile je paramétrais un « hostname » pour chaque machine et c'est précisément cela qui empêchait la création du cluster. Après avoir modifié le fichier et recréé mes VMs, j'ai enfin pu créer mon cluster. Voici le fichier Vagrant corrigé :

```
1 Vagrant.configure("2") do |config|
2   config.vm.box = "hsmnn/proxmox-ve"
3
4   (1..3).each do |i|
5     config.vm.define "proxmox-node-#{i}" do |node|
6       node.vm.network "private_network", ip: "192.168.1.#{i}"
7       node.vm.network "forwarded_port", guest: 8006, host: "800#{i}", auto_correct: true
8       node.vm.provider :virtualbox do |vb|
9         vb.name = "proxmox-node-#{i}"
10      end
11    end
12  end
13 end
```

Maintenant que le cluster est créé, il ne manque plus qu'à y ajouter les deux autres machines Proxmox. Pour ce faire, il faut d'abord copier les informations permettant aux autres machines de rejoindre le cluster.

Cluster Join Information

Copy the Join Information here and use it on the node you want to add.

IP Address:

10.0.2.15

Fingerprint:

B3:B8:69:F0:9B:19:BE:DB:F4:C7:32:4C:64:0D:81:53:18:F0:34:E5:1C:52:05:77:C1:80:8A:74:BA:09:53:E1

Join Information:

eypJcEFkZHJlc3MiOiIxMC4wLjluMTUilCJmaW5nZXJwcmludCl6IklzOkI0YjY5OkYwOjIjCOJE50kJFokRCOKY0OkM3OjMyOjRDQjY0OjBEQjgxOjUzOjE4OkYwOjM0OkU1OjFDQjUyOjA1Ojc3OkMxOjgwOjhBOjc0OkJB0jAS0jUzOkUxlwicGVickpbmtZljp7ljAiOixOTlUmTY4LjEuMSJ9LCJyaW5nX2FkZHI0IsImTkyLjE2OC4xLjEiXSwidG90ZW0iOnsidmVyc2lybil6IiIlLCJib25maWdfdmdVmc2lybil6IiILCJibHVzdGvYX25hbWUiOiJ0ZXN0IiwiaXBf

Copy Information

Ensuite, il faut les coller au bon endroit, sur la machine qui va rejoindre le cluster.

The screenshot shows the Proxmox VE web interface. At the top, the header includes the Proxmox logo, version 'Virtual Environment 0.3.7', and navigation links for Documentation, Create VM, Create CT, and Host/Join. The left sidebar contains a 'Server View' menu with 'Datacenter (test)' selected, showing a tree with 'pve'. The main content area is titled 'Datacenter' and has a 'Cluster Information' tab selected. Within this tab, the 'Join Cluster' button is highlighted with a red rectangle. Below the buttons, the cluster details are shown: 'Cluster Name: test', 'Config Version: 1', and 'Number of Nodes: 1'. A 'Cluster Nodes' table lists the node 'pve' with ID 1, 1 vote, and IP 192.168.1.1. The bottom of the image shows a 'Cluster Join' dialog box with the 'Assisted join' checkbox checked and a text area for pasting encoded cluster information.

Et pour finir, il faut encore entrer le mot de passe de la machine sur laquelle se trouve le cluster ainsi que l'adresse IP de la machine rejoignant le cluster.



Cluster Join

☒ Assisted join: Paste encoded cluster join information and enter password.

Information:

50jUzOkUxiwicGVickxbmtzjp7IjAoiIxoTlUmTY4LjEuMSJ9LCJyaW5nX2FkZHIiOiIsImTkyLjE2OC4xLjEiXSwiZG90ZW0iOiIsbnB2dCklsmiwX3ZlcnNpb24iOiJpcHY0LTlYLjZzZW5hdXRoljoib24iLjpbncmZnY2UiOiIsMCI6eyJsaW5nRnVtYmVyjoicj9kX9k19

Peer Address:

10.0.2.15

Password: \*\*\*\*\*

Fingerprint:

B3:B8:69:F0:9B:19:BE:DB:F4:C7:32:4C:64:0D:81:53:18:F0:34:E5:1C:52:05:77:C1:80:8A:74:BA:09:53:E1

Cluster Network:

Link: 0

peer's link address: 192.168.1.1

Help

Join 'test'

Je n'ai malheureusement jamais réussi à mettre un nœud dans mon cluster car je rencontrais l'erreur suivante :

Task viewer: Join Cluster

Output

Status

Stop

Establishing API connection with host '10.0.2.15'  
Login succeeded.  
check cluster join API version  
Request addition of this node  
An error occurred on the cluster node: can't lock file '/var/lock/pvecm.lock' - got timeout  
TASK ERROR: Cluster join aborted!

J'ai consacré les deux dernières demi-journées de ce module à chercher une solution à ce problème mais je n'ai rien trouvé. J'ai essayé plein de choses différentes (redémarrer le service, supprimer le fichier, recréer le fichier, recréer mes VMs, etc.), j'ai cherché de l'aide auprès de mes collègues qui avaient tous la même erreur ainsi qu'auprès du professeur mais personne n'a pu m'aider. Comme le module touchait à sa fin, j'ai dû arrêter mes recherches infructueuses jusqu'alors.

## 16 Conclusion

Ce module nous a permis d'apprendre les concepts de la virtualisation et comment mettre en place une infrastructure virtuelle grâce à Vagrant. Nous avons également appris à créer nos propres basebox avec Packer, à approvisionner nos machines virtuelles avec SaltStack et à utiliser une solution de virtualisation avec Proxmox.

L'informatique se dirige vers des infrastructures de plus en plus virtualisées et dans le cloud. Toutes les connaissances et compétences que l'on a accumulés lors d'un tel module ne peuvent que nous aider à mieux comprendre l'informatique d'aujourd'hui et de demain.

## 17 Bibliographie

1. **ict-berufsbildung**. 340 Virtualiser une infrastructure informatique. *ict-berufsbildung.ch*. [En ligne] 12 09 2017. [Citation : 04 03 2021.] <https://cf.ict-berufsbildung.ch/modules.php?name=Mbk&a=20101&cmodnr=340&noheader=1>.
2. **Mutai, Josphat**. How To Install and use Packer on Ubuntu 20.04/18.04/16.04. *computingforgeeks.com*. [En ligne] 02 mai 2020. [Citation : 02 avril 2021.] <https://computingforgeeks.com/how-to-install-and-use-packer/>.
3. **SaltStack**. SaltStack Fundamentals. *docs.saltproject.io*. [En ligne] [Citation : 25 avril 2021.] <https://docs.saltproject.io/en/getstarted/fundamentals/index.html>.
4. **Hat, Red**. Un hyperviseur, qu'est-ce que c'est ? *redhat.com*. [En ligne] [Citation : 04 mars 2021.] <https://www.redhat.com/fr/topics/virtualization/what-is-a-hypervisor>.
5. **Wikipedia**. Hyperviseur. *wikipedia.org*. [En ligne] 27 décembre 2020. [Citation : 04 mars 2021.] <https://fr.wikipedia.org/wiki/Hyperviseur#:~:text=En%20informatique%2C%20un%20hyperviseur%20est,machine%20physique%20en%20m%C3%A4me%20temps..>
6. **L., Bastien**. Virtualisation : qu'est-ce que c'est et à quoi ça sert ? *lebigdata.fr*. [En ligne] 13 mai 2019. [Citation : 14 mars 2021.]
7. **Motta, L  lio**. Emuler, simuler, virtualiser : de quoi parle t-on ? *openclassrooms.com*. [En ligne] 15 d  cembre 2020. [Citation : 05 mars 2021.] <https://openclassrooms.com/fr/courses/2581701-simulez-des-architectures-reseaux-avec-gns3/4823141-emuler-simuler-virtualiser-de-quoi-parle-t-on#:~:text=Dans%20le%20cadre%20des%20syst%C3%A8mes,%C3%A0%20quelques%20diff%C3%A9rences%20notables%20pr%C3%A8s.&te>.
8. **Charv  riat, Laurent**. Les risques majeurs de la virtualisation. *journaldunet.com*. [En ligne] 01 septembre 2010. [Citation : 07 mars 2021.] <https://www.journaldunet.com/solutions/dsi/1031138-les-risques-majeurs-de-la-virtualisation/>.
9. **HP**. D  FINITION DE LA VIRTUALISATION DES SERVEURS. *hpe.com*. [En ligne] [Citation : 07 mars 2021.] <https://www.hpe.com/fr/fr/what-is/server-virtualization.html#resources>.
10. **Salaun, Alexandre**. Vagrant et la virtualisation pour faciliter le d  veloppement. *synbioz.com*. [En ligne] 27 f  vrier 2013. [Citation : 11 mars 2021.] <https://www.synbioz.com/blog/tech/vagrant-et-la-virtualisation-pour-faciliter-le-developpement>.

11. **Anicas, Mitchell.** An Introduction to HAProxy and Load Balancing Concepts. *digitalocean.com*. [En ligne] 13 mai 2014. [Citation : 18 mars 2021.] <https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts>.
12. **Nandadasa, Savithri.** An introduction to Ansible. *medium.com*. [En ligne] 27 juin 2020. [Citation : 02 avril 2021.] <https://medium.com/@snsavithrik1/an-introduction-to-ansible-aa62559d97de>.
13. **Ansible.** Ansible Documentation. *docs.ansible.com*. [En ligne] [Citation : 02 avril 2021.] <https://docs.ansible.com/ansible/latest/index.html>.
14. **Ionos.** Qu'est-ce que SaltStack ? *ionos.fr*. [En ligne] 16 07 2020. [Citation : 22 04 2021.] <https://www.ionos.fr/digitalguide/serveur/configuration/quest-ce-que-saltstack/>.
15. **tutorialspoint.** *tutorialspoint.com*. [En ligne] SaltStack - Overview. [Citation : 22 04 2021.] [https://www.tutorialspoint.com/saltstack/saltstack\\_overview.htm](https://www.tutorialspoint.com/saltstack/saltstack_overview.htm).
16. **Wikipedia.** Proxmox VE. *wikipedia.org*. [En ligne] 20 janvier 2021. [Citation : 04 avril 2021.] [https://fr.wikipedia.org/wiki/Proxmox\\_VE](https://fr.wikipedia.org/wiki/Proxmox_VE).
17. **Academy, Billysoft.** How To Install The Proxmox VE 6.2 Hypervisor On Oracle VM VirtualBox 6.1 In 10 Minutes Or Less. *billysoftacademy.com*. [En ligne] [Citation : 04 avril 2021.] <https://billysoftacademy.com/how-to-install-the-proxmox-ve-6-2-hypervisor-on-oracle-vm-virtualbox-6-1-in-10-minutes-or-less/>.
18. **Proxmox.** Proxmox VE inside VirtualBox. *pve.proxmox.com*. [En ligne] 15 septembre 2020. [Citation : 29 avril 2021.] [https://pve.proxmox.com/wiki/Proxmox\\_VE\\_inside\\_VirtualBox](https://pve.proxmox.com/wiki/Proxmox_VE_inside_VirtualBox).
19. **Ahbib, Ismaayl.** De la machine virtuelle au conteneur. *spiria.com*. [En ligne] 23 mai 2019. [Citation : 03 mai 2021.] [https://www.spiria.com/fr/blogue/applications-desktop/de-la-machine-virtuelle-au-conteneur/#:~:text=Machine%20virtuelle%20\(VM\)%20et%20conteneur,son%20propre%20syst%C3%A8me%20d'exploitation.&text=Et%20ce%20n'est%20pas%20le%20seul%20avantage%20des%20contene](https://www.spiria.com/fr/blogue/applications-desktop/de-la-machine-virtuelle-au-conteneur/#:~:text=Machine%20virtuelle%20(VM)%20et%20conteneur,son%20propre%20syst%C3%A8me%20d'exploitation.&text=Et%20ce%20n'est%20pas%20le%20seul%20avantage%20des%20contene).
20. **Cantor.** L'orchestration de containers : quelles alternatives à Kubernetes ? *cantor.fr*. [En ligne] [Citation : 03 mai 2021.] <https://cantor.fr/orchestration-de-containers/>.