M2I Cybersécurité

Vagrant répond-t-il à un besoin important dans le monde de l'entreprise ?

CCI VAUCLUSE

Fabian CATANI et Romain VALDEBON
CCI Vaucluse

ESIEE [it]

Remerciement

Nous tenons tout d'abord à remercier toute l'équipe pédagogique qui on su nous accompagner tout au long de cette année et avoir pu collaborer avec nous.

Nos remerciements vont également à nos maitres d'apprentissage, pour leur précieux conseils, leur expertise et leur soutien déterminé ont été d'une aide inestimable et ont grandement contribué à l'aboutissement de notre projet.

Également, un remerciement spécial à nos camarades de classe et amis, pour leurs encouragements et leurs échanges intellectuels.

Enfin, nous tenons à remercier chacune des personnes qui ont contribué à la réalisation de ce mémoire. Que ce soit à travers des discussions enrichissantes, des conseils ou simplement par leur présence, leur contribution a été précieuse.

Ce mémoire fut une expérience enrichissante tant sur le plan technique et connaissance. Vagrant nous a aidés à voir plus loin dans la virtualisation. La virtualisation qui est très présente de nos jours dans les entreprises et qui permet de faciliter la tâche de tout un personnel informatique.

Remerciement **2**
1. Abstract **5**
1.1 A little history 5
1.2 What is a VagrantFile ? 5
1.3 How does Vagrant work? 6
1.4 Concrete examples of Vagrant 6
1.5 What are the advantages of Vagrant ? 7
1.6 Difficulties encountered 8
2. Veille Technologique **10**
2.1 Qu'est ce qu'une veille technologique ? 10
2.2 Déroulement de la veille 11
2.2.1 Objectifs de la veille 11
2.2.2 Sélections et collecte des sources d'information 11
2.2.3 Plan et Méthodologie utilisée 12
3. Introduction **13**
3.1 Qu'est ce que Vagrant ? 13
3.2 Objectifs de Vagrant 13
3.3 Historique de Vagrant 14
4. Concepts Fondamentaux **15**
4.1 Automatisation de la virtualisation 15
4.2 La fondation de Vagrant 15
5. Architecture de Vagrant **17**
5.1 Schéma général 17
5.2 Fonctionnement de Vagrant 19
5.3 Schéma complet sur le fonctionnement de Vagrant 20
6. Installation et Configuration **22**
6.1 Création et gestion d'environnement virtuel 22
6.1.1 Installation de Vagrant 22
6.2 Commandes Vagrant essentielles 22
6.2.1 Les commandes de gestion des machines virtuelles 23
6.2.2 Les commandes de gestion des box Vagrant 24
7. Provisionnement avec Vagrant **25**
7.1 Explication 25
7.2 Provisioning avec d'autres Outils 25
7.2.1 Chef 25
7.2.2 Ansible 25
7.3 Scripts de Provisionnement 26
7.3.1 Provisionnement commun dans un VagrantFile 26
7.3.2 Exemple d'un extrait de script : 26
8. Sécurité dans Vagrant **28**

8.1 Sécurité incluse avec Vagrant..... 28
8.1.1 Virtualisation et réseau isolé..... 28
8.1.2 Gestions des fichiers VagrantFile..... 28
8.2 Risques et vulnérabilités connus..... 29
8.3 Sécurisation avec Ansible..... 30
9. Comparaison avec d'autres outils..... 31
9.1 Docker..... 31
9.1.1 Qu'est ce que Docker ?..... 31
9.1.2 Différence avec Vagrant..... 32
9.2 Kubernete..... 32
9.3 Avantages et Inconvénients de Vagrant..... 33
10. Preuve de concept..... 34
10.1 Phase d'implémentation..... 34
10.2 Organisation de travail..... 35
10.2.1 Utilisation de Vagrant Cloud..... 36
10.2.2 Utilisation d'un espace GitHub..... 38
10.3 Déploiement de nos systèmes..... 38
11. Conclusion..... 41
12. Sources..... 42
13. Glossaire..... 44
14. Annexes..... 45
14.1 Configuration d'une box Vagrant Linux :..... 45
14.2 Fichier Vagrantfile..... 46
14.3 Vagrantfile Windows Server..... 49
14.4 Vagrantfile Windows Client..... 50
14.5 Vagrantfile Ubuntu Server..... 51
14.6 Vagrantfile Kali Linux..... 53
14.7 Playbook Ansible Windows Server..... 54
14.8 Playbook Ansible Windows Client..... 56

1. Abstract

1.1 A little history

Vagrant was conceived in 2010 by Mitchell HASHIMOTO, a software development enthusiast. His idea was to facilitate production environments and make it a tool for rapidly creating new infrastructures. The idea is to be able to build any infrastructure using ruby scripts. Vagrant integrates with several hypervisors, including VirtualBox, VmWare and Hyper-V, making it a major asset in terms of compatibility.

Vagrant is a portable virtualization software package designed to simplify and standardize the configuration workflow of IT development environments. The main aim is to meet the needs of developers by facilitating the task of creating and managing reproducible virtual machines, and all this with the greatest of ease.

In today's world, software development projects may require dependencies, specific programming language versions, or even particular configurations and the like. This makes configuration tasks time-consuming and complicated, and could even lead to errors.

This is why, using Vagrant, we could tackle configuration instead of launching virtualization directly. This configuration would be based on a file called “Vagrantfile” and would then allow developers to define their desired requirements inside.

1.2 What is a VagrantFile ?

This file is like a recipe sheet, in that step by step, it defines the features and parameters required for the project. It's so simple that once this file has been configured, all that's left to do is run a Vagrant command which will create, start and then configure the environment automatically and according to the configurations specified in the “Vagrantfile”.

The main advantage of Vagrant is its ability to create coherent, copyable development environments that are independent of the host system. This approach increases the guarantee of parity between development/production and the stability of environments across deployments and software updates, reducing conflicts due to external updates and sources of variation, ensuring that the excuse “it works on my machine” is no longer possible.

In conclusion, Vagrant aims to simplify the management of stable versions of environments by enabling the creation of frozen versions of configurations.

This diagram shows how Vagrant works:

- Vagrant Cloud: stores our Vagrant boxes (personal virtual machines)

- VagrantFile: The vagrantfile is used to deploy our virtual machine using a script. This script is based on the ruby programming language.

- Boxes : Boxes are where our virtual machines are stored. They enable us to deploy them at any time.

- Virtual machines: This is where our virtual machines are deployed and operational.

The aim of this study on Vagrant technology is to discover another way of quickly and easily designing a system and network infrastructure. Vagrant has many advantages in this area, but is not without its shortcomings.

1.3 How does Vagrant work?

Vagrant is based on a concept called “box”. A box allows us to store our environments and then deploy them. These boxes contain our virtual machines and their configurations. Once the boxes have been created, Vagrant needs to initialize them using the command “Vagrant init <box name>”. This command will then initialize our box with a VagrantFile and a “.vagrant” folder containing all our previous configuration. Once these steps have been completed, we can launch our environment using the “vagrant up” command. Our virtual machine will then be imported into our hypervisor and configured with its provisioning.

1.4 Concrete examples of Vagrant

In this example (Proof of concept), we're going to deploy a training lab for a pentest in a private network. The lab will include a Windows server hosting a vulnerable **domain controller**, a Linux server hosting a vulnerable **FTP server** and **GLPI instance**, and a Kali Linux machine to be used by the user for penetration testing.

A **domain controller** is a server that responds to authentication requests and controls computer network users.

The **FTP server** lets you transfer files over the Internet or a computer network.

GLPI is an open-source software package for IT service management and help desk management.

First, we define the different virtual machines and their configurations in the Vagrantfile. In it, we specify a Linux box (Linux server), a Windows 2022 Server box and a Kali Linux box to create virtual machines. Each machine will have parameters such as allocated memory, number of sockets and an IP address to integrate it into our private network.

In a second step, we will define the provisioning of each virtual machine. For the Windows server, we will use Ansible. We will therefore create an Ansible playbook file that will contain the installation of the "domain controller" role and its configuration. This will be launched by the Vagrantfile. Then for GLPI and FTP we will use Shell as a provisioner, where we will write directly in the Vagrantfile the commands to install and configure the software.

In a third step, to run our Vagrantfile and deploy our Lab, we will execute the "Vagrant up" command, which will follow step by step the specifications for the deployment of virtual machines. Then, Vagrant will use Ansible to run the Playbook and do the provisioning of the Windows server and Shell to run step by step the commands for the Linux server. Provisioning completed, our Lab is operational.

In a cybersecurity-oriented educational context, Vagrant offers the opportunity to quickly deploy and manage a training Lab for pentesting and other activities. The user has a wide range of choices in terms of technology and configuration to make the experience all the more real.

Using distributions such as Kali Linux, students can apply the cybersecurity knowledge they have acquired to identify, analyze and exploit the security vulnerabilities present in these virtual environments.

In the field of teaching information and communication technologies, Vagrant is an interesting tool as it enables students to understand network, system, infrastructure and security concepts more easily, in a practical and interactive way, through environments that approach reality.

1.5 What are the advantages of Vagrant ?

One of Vagrant's key features is its compatibility. As an open source tool, Vagrant offers extensive flexibility with various versions of operating systems:

- Windows
- Linux

the most famous virtualization provider :

- Virtualbox
- Hyper-V
- Vmware
- AWS

And the provisioning software :

- Shell
- Ansible
- Chef
- and others

So, in fact, Vagrant is a flexible and automatic solution, because on the one hand users can choose the base (virtualization providers) that best meets their needs for their environment, and

on the other hand they can choose between different provisioning tools to customize and configure it, while simplifying the process by automating each task.

Another advantage of Vagrant is its portability, enabling environments to be easily shared between team members and then reproduced on different machines.

On the other hand, it's important to consider certain aspects that can pose challenges when using Vagrant. Here are a few to keep in mind:

- System resources: The use of virtual machines can require significant system resources, in terms of memory and computing performance (CPU), which may restrict their deployment on certain less powerful machines.
- Complexity: Although Vagrant simplifies the process of configuring development environments, learning how it functions and how to integrate it with other tools can take time and practice.
- Maintenance: Managing development environments (templates) created with Vagrant requires a certain amount of maintenance to ensure stability and security, particularly with regard to updates and security patches that could have an impact.
- Versioning: Although Vagrant is compatible with several virtualization providers and provisioning tools, compatibility or versioning problems can sometimes arise, depending on the use of more recent or less widespread technologies.

It is therefore important to bear in mind that, while Vagrant offers many advantages for the creation and management of lab-oriented environments and small-scale infrastructure, the challenges associated with its use still need to be taken into account to ensure that it runs smoothly, particularly in terms of versioning complementary tools, for example.

1.6 Difficulties encountered

Vagrant is a comprehensive and easy-to-use tool. However, during our testing and implementation phase, we encountered several problems.

Firstly, Vagrant has several versions which are not compatible with each other. During our first tests, we noticed version problems between Vagrant and ansible for machine provisioning. During deployment, Vagrant will make an ssh connection with our host machine, and a configuration is required to establish the link. Box creation can be tricky and repetitive. When a box is created, it's important to make sure that all the necessary components are present, to avoid additions or additional bugs in the final phase. Finally, Vagrant is easy to use once you've got the hang of it, but when you're just starting out, understanding Vagrant can be tricky and frustrating. Vagrantfile files are sensitive to indentation and formulations.

In conclusion, Vagrant represents a real advancement in simplifying and automating the configuration of computing environments, whether for development, modeling or lab training. It enables users to quickly and easily deploy reproducible and consistent virtual environments via Vagrantfile files. It also offers extensive flexibility and compatibility with virtualization providers and provisioning tools.

Vagrant's portability also makes it easy to share environments between team members, fostering collaboration and coherence between a pre-production and a production environment, preventing the famous phrase "But it runs on my machine". All the same, it remains essential to be aware of the challenges associated with setting up an environment using Vagrant.

In the end, by taking into account both the advantages and the challenges, users can optimize their various work processes in areas such as learning, development, network and system administrators and others.

2. Veille Technologique

2.1 Qu'est ce qu'une veille technologique ?

Le principal avantage de la veille technologique est d'identifier ou anticiper les avancées sur un secteur d'activité. Elle permet de développer nos connaissances techniques et d'en apprendre davantage sur un domaine spécifique. Ici notre sujet est Vagrant.

Mettre en place une veille technologique sur des domaines identifiés permet de constituer une solide base d'information pour comprendre les technologies.

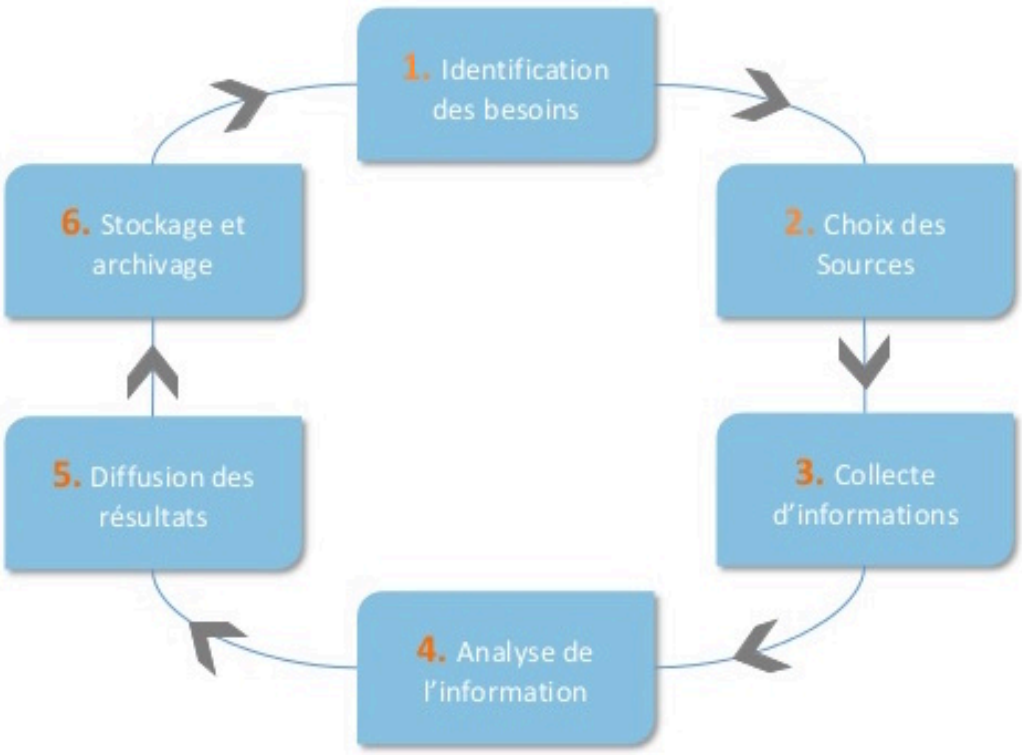


Schéma d'une veille technologique
(source : https://graphizm.fr/greta/veille/mener-une-veille-informationnelle/)

2.2 Déroulement de la veille

2.2.1 Objectifs de la veille

Pour commencer la veille sur la technologie Vagrant, nous avions dû définir des objectifs pour collecter un maximum d'informations.

Objectifs :

- **Comprendre les bases de Vagrant** : apprendre son fonctionnement, ses avantages et ses inconvénients.
- **Pratiquer l'utilisation de la technologie** : identifier et comprendre l'utilisation des commandes et l'architecture de Vagrant.
- **S'informer sur l'évolution de la solution** : se tenir au courant des mises à jour de la technologie et donc d'apprendre les nouveautés futures.
- **Outils annexes** : Comprendre comment intégrer des outils externes à Vagrant comme Docker, Ansible ou encore Chef.
- **Étudier les ressources d'apprentissages** : Utilisation de la documentation officielle de Vagrant, également apprendre la solution via des vidéos et toute autre documentation.
- **Identification des problèmes** : S'informer des problématiques fréquentes sur l'utilisation de Vagrant et d'en anticiper de futurs problèmes.

2.2.2 Sélections et collecte des sources d'information

Pour avancer sur le projet, nous devons faire une sélection des sources d'informations. Il existe de nombreux sites pour cette collecte.

Le site de **Vagrant** inclut sa documentation pour comprendre l'utilisation de **Vagrant**. Les ressources sont très utiles pour obtenir des informations de l'outil.

Nous avons les répertoires de **GitHub** liées à **Vagrant** qui peuvent donner un aperçu de la configuration du logiciel.

2.2.3 Plan et Méthodologie utilisée

Pour conclure, le fait d'avoir suivi un ordre logique pour l'apprentissage et la connaissance de Vagrant nous ont permis de comprendre très rapidement le fonctionnement et l'application de celui-ci. Nous nous sommes basés sur le référentiel : (https://www.scribbr.fr/category/memoire/). Nous l'avons simplifié et obtenu sur notre plan d'action :

1. **Avoir un plan structuré** : avant de commencer notre mémoire, nous avons dû réfléchir à un plan d'action avec les différentes catégories qui seront évoquées ainsi que la clarté de notre discours.
2. **Avoir un bon sommaire** : créer et rédiger un sommaire simple, mais efficace. Cela va permettre aux jurys de trouver aisément les chapitres.
3. **Rédiger des remerciements** : il est important d'avoir en première page des remerciements qui s'adressent directement à tous nos collaborateurs.
4. **Avoir une Introduction** : une introduction se doit d'être claire et précise et qui doit faire accrocher le lecteur.
5. **Partie théorique** : Une explication théorique doit être très complète pour toucher le maximum de personnes. Entre autres, expliquer chaque outil utilisé, chaque configuration utilisée.
6. **Partie technique** : La partie technique est le plus gros point d'un mémoire. Nous devons avoir une bonne connaissance de notre sujet et comprendre tout le mécanisme de celui-ci.
7. **Conclusion** : Intégrer une conclusion claire et pertinente qui doit faire un comparatif avec d'autres outils, la facilité d'intégration et également les problèmes rencontrés.

3. Introduction

3.1 Qu'est ce que Vagrant ?

Dans les domaines du développement logiciel, gestion d'infrastructure et concepteur web, créer puis maintenir des environnements cohérents pour leurs applications devient un problème majeur.

Mitchell HASHIMOTO fondateur de HashiCorp passionné d'informatique et de développement logiciel à créé un outil nommé Vagrant en 2010, qui est un logiciel open source qui possède un flux de travail facile à utiliser et une concentration sur l'automatisation.

Entre autres, il offre une solution flexible pour créer des environnements de développement isolés, portables et reproductibles et cela, quel que soit le système d'exploitation requis.

Vagrant est également considéré comme un **wrapper** autour des logiciels de virtualisations comme **Virtualbox, Hyper-V, Vmware** ou **AWS.**

3.2 Objectifs de Vagrant

L'objectif de Vagrant est de simplifier l'implémentation de machines virtuelles et le partage d'environnements de machines virtuelles en utilisant des fichiers de configuration simples (VagrantFile) qui par la suite les développeurs pourront configurer selon leurs dépendances et paramètres requis pour leurs besoins.

Vagrant permet :

- Environnements de développement homogènes
- Simplification des tests
- Intégrations et livraison continues
- Reproductibilité des environnements de développement
- Collaboration et partage d'environnements
- Flexibilité des choix du fournisseur de virtualisation

Pour résumer, Vagrant a pour objectif de simplifier la gestion des versions stables des environnements en permettant la création de versions figées des configurations. Cette méthode augmente la garantie de la parité entre développements/production et la stabilité des environnements à travers les déploiements et les mises à jour logicielles, réduisant ainsi la dépendance aux mises à jour externes et aux sources de variation, ce qui permet de faire en sorte que l'excuse « ça marche sur ma machine » ne soit plus possible.

3.3 Historique de Vagrant

Vagrant est créé en 2010 par Mitchell Hashimoto, il est un passionné de développement logiciel. Il veut trouver une solution dans ce domaine qui permettrait de standardiser la création d'environnements virtuels. Cette innovation permet aux développeurs de travailler avec les machines virtuelles, offrant une solution pour éliminer les problèmes de compatibilité entre environnements.

Dans les premiers temps, Vagrant permettait de rendre les environnements de développement aussi proches que possible de la production, afin de résoudre les problèmes récurrents mentionnés précédemment. Grâce à sa simplicité d'utilisation et à sa gestion homogène des environnements, l'outil s'impose rapidement comme un incontournable du développement logiciel.

Vagrant évolue en intégrant de nouvelles fonctionnalités et en élargissant sa compatibilité avec différents fournisseurs de services de virtualisation reconnue sur le marché mais également l'intégration d'outils de provisionnement automatisé renforce encore plus son efficacité.

Pour finir, la création de HashiCorp en 2012 marque une étape majeure dans son développement, offrant à Vagrant les ressources nécessaires pour continuer à innover et à répondre aux besoins de sa vaste communauté.

Source : https://blog.stephane-robert.info/docs/infra-as-code/provisionnement/vagrant/introduction/#:~:text=Petit%20Historique%20de%20Vagrant%E2%80%8B,cr%C3%A9ation%20d'environnements%20de%20d%C3%A9veloppement.

4. Concepts Fondamentaux

4.1 Automatisation de la virtualisation

Vagrant permet de mettre au point une ou plusieurs machines virtuelles sous forme de code appelé **VagrantFile**. C'est-à-dire que sur le plan technique, cette automatisation fonctionne à partir de fichiers de configuration, qui sont généralement écrits en langage de script appelé **Vagrantfile**. Le fichier **VagrantFile** est basé sur le langage de programmation **Ruby**. Ces fichiers permettent de configurer les caractéristiques et paramètres de l'environnement, tels que le système d'exploitation, le nombre de cœurs par socket virtuel, la mémoire, les logiciels à installer, les paramètres réseau, etc. Cela permet de définir toutes les caractéristiques d'une VM sans passer par l'interface de **Virtualbox**, **VmWare** ou autre.

4.2 La fondation de Vagrant

La fondation de Vagrant permet de comprendre son bon fonctionnement et son utilisation dans un contexte technique.

Dans un premier temps, les machines virtuelles (VM) :

Vagrant s'appuie sur le principe de machines virtuelles, qui sont des environnements virtualisés isolés. Chaque machine virtuelle créée est donc une instance isolée du système d'exploitation, permettant d'effectuer des tests dans des environnements définis et facilement reproductibles en cas de panne technique.

Dans un second temps, les fichiers de configuration (Vagrantfile) :

Vagrant utilise des fichiers appelés Vagrantfile qui font office de fichiers de configuration pour les machines virtuelles. Ils permettent de choisir le système d'exploitation et les ressources qu'il faudra allouer à la création de la VM. Ils sont également rapides à prendre en main, car ils possèdent une syntaxe claire et facile à comprendre.

Dans un troisième temps, le provisionnement automatique :

Vagrant permet d'automatiser le provisionnement de la machine virtuelle en intégrant des outils de configuration automatique connus du grand public. Ces outils permettent entre autres de rajouter un gain de temps car il ne sera plus nécessaire à la suite de configurer chaque VM manuellement. Cela améliorera également la cohérence entre les environnements au sein des équipes.

Dans un dernier temps, les fournisseurs de virtualisation :

Vagrant est compatible avec de nombreux logiciels de virtualisation comme VMWare ou Virtualbox pour exemple. Les utilisateurs peuvent donc choisir le fournisseur de virtualisation qui

répond le mieux à leurs besoins ou à leurs infrastructures, en fonction de leurs performances, compatibilités et fonctionnalités spécifiques.

Ainsi en combinant ces concepts, Vagrant offre donc aux utilisateurs un moyen efficace et flexible de créer, gérer et partager des environnements de développement virtuels, facilitant ainsi le développement logiciel et favorisant la collaboration au sein des équipes.

5. Architecture de Vagrant

5.1 Schéma général

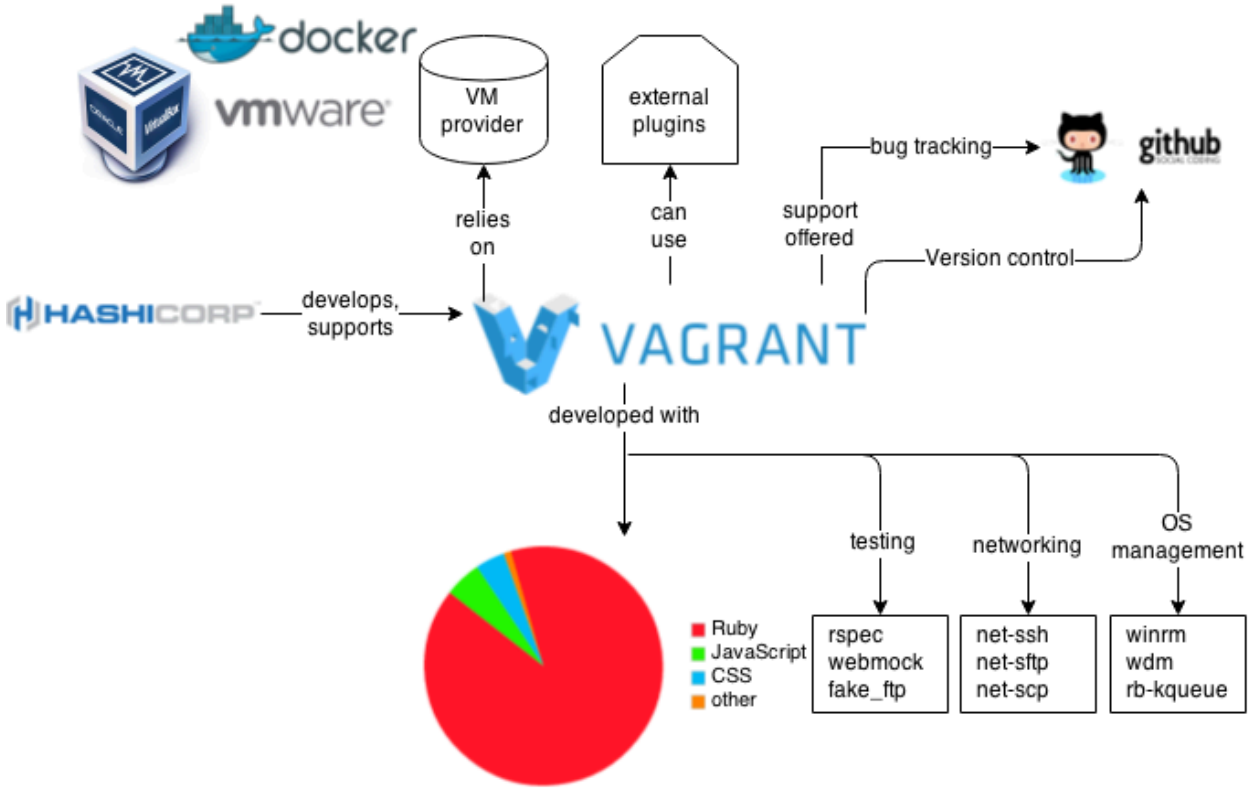


Schéma simplifié de Vagrant
(source : https://delftswa.github.io/chapters/vagrant/)

Ce schéma montre un aperçu de toutes les entités externes interagissant avec l'écosystème de Vagrant. Au cœur de cet écosystème se trouve HashiCorp, qui développe et maintient Vagrant.

Ensuite on retrouve GitHub qui, en tant que plateforme de développement collaboratif, offre un support important pour les utilisateurs en hébergeant des dépôts de code source contenant par exemple des configurations Vagrantfile et des scripts de provisionnement. GitHub offre une approche simplifiée et pratique pour le travail collaboratif, dès qu'une personne met à jour le fichier vagrantfile alors qu'elle sera effective pour tout le monde.

Nous avons également des providers qui permettent la virtualisation de nos machines virtuelles.

Ensuite, nous avons les plugins, ils constituent une partie importante Vagrant car ces extensions permettent d'étendre les fonctionnalités de celui-ci, telles que des commandes ou

des intégrations avec d'autres outils de développement comme Ansible, Packer ou autre. Les plugins sont en partie développés par des membres de la communauté et permettent aux utilisateurs de personnaliser leur expérience avec Vagrant.

Pour finir, Vagrant est développé principalement en ruby, les autres langages sont pour la plupart des langages de développement web. La catégorie de test est représentée par des outils comme :

RSpec (vagrant-spec) est un outil de test pour Ruby, il permet aux développeurs d'écrire des tests automatisés pour valider le comportement de leur code.

WebMock intervient pour simuler les requêtes HTTP dans les tests, ce qui permet ainsi un contrôle précis sur les interactions avec des services externes.

FakeFTP simule de fausses connexions FTP, permettant aux développeurs de tester le comportement de leur code et environnement lors d'interactions avec des serveurs FTP, sans pour autant nécessiter de devoir établir de véritables connexions.

La catégorie networking contient des modules comme Net-SSH, Net-SFTP et Net-SCP. Ces outils permettent à Vagrant d'établir des connexions (sécurisées) avec des machines virtuelles à travers SSH, permettant ainsi le déploiement de configurations et de fichiers sur les machines distantes. En autres, Net-SFTP et Net-SCP permettent des fonctionnalités de transfert de fichier entre la machine hôte et les box, tandis que Net-SSH permet d'exécuter des commandes à distance sur les machines virtuelles comme par exemple des commandes préalablement notées dans le Vagrantfiles.

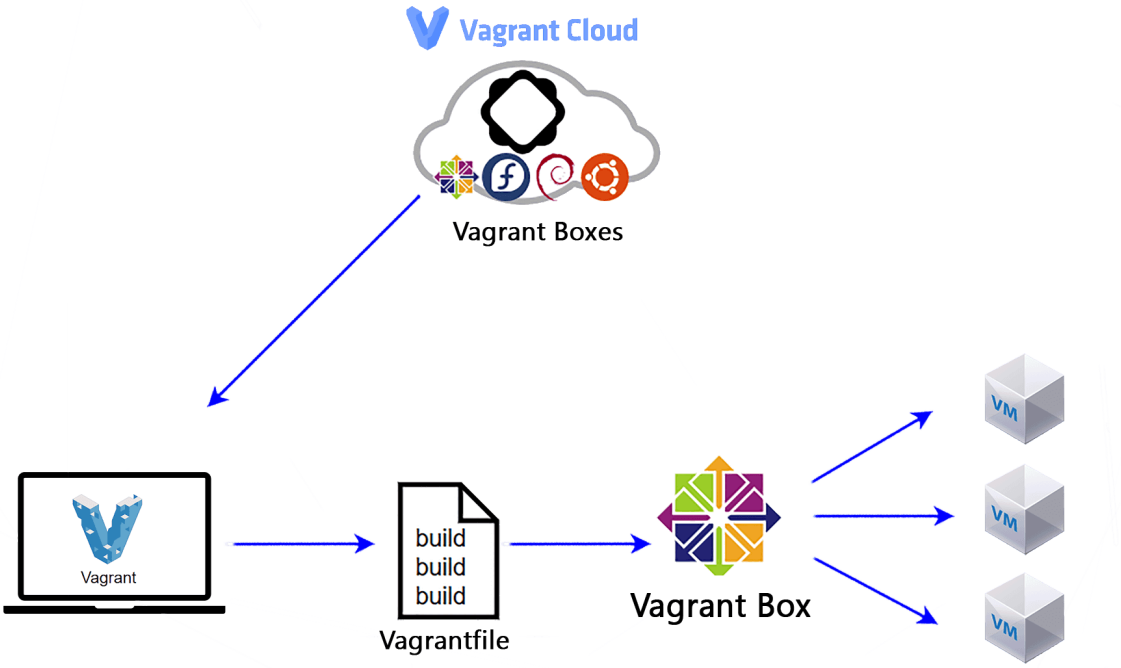
Pour finir, la catégorie de gestion des systèmes d'exploitation (OS) contient des outils comme :

WinRM qui est utilisé pour la gestion à distance des VM Windows, permettant ainsi les opérations de configuration (réseau, utilisateur,etc.) et de maintenance (mise à jour).

WDM qui permet d’offrir des fonctionnalités spécifiquement faites pour la gestion des VM Windows, facilitant donc l'installation et la configuration des pilotes et des logiciels nécessaires.

Rb-kqueue, qui quant à lui, permet la supervision des nouveaux changements de fichiers sur les systèmes Unix, favorisant ainsi l'automatisation des tâches de gestion dans les environnements Vagrant.

5.2 Fonctionnement de Vagrant



Fonctionnement de Vagrant schématisé

(source: https://www.linkedin.com/posts/malek-bouzayani_springboot-docker-d%C3%A9veloppementlogiciel-activity-7177298269604114432-JeEv/?originalSubdomain=fr)

Ce schéma simplifié démontre en quelques étapes la création et la configuration d'une machine virtuelle via Vagrant :

1. Construction de la base de la VM (Base Box) pour cela :
   - Soit-on la récupère déjà faite sur **VagrantCloud**
   - Soit créer une box en local
2. Construction de la VM finale :
   - Configuration de la VM avec le Vagrantfile
   - Provisionnement de la VM avec un provider (avec Ansible sur cet exemple)
   - Intégration au logiciel de virtualisation (avec VirtualBox sur cet exemple)

On peut donc dire que les deux outils sont complémentaires.

5.3 Schéma complet sur le fonctionnement de Vagrant

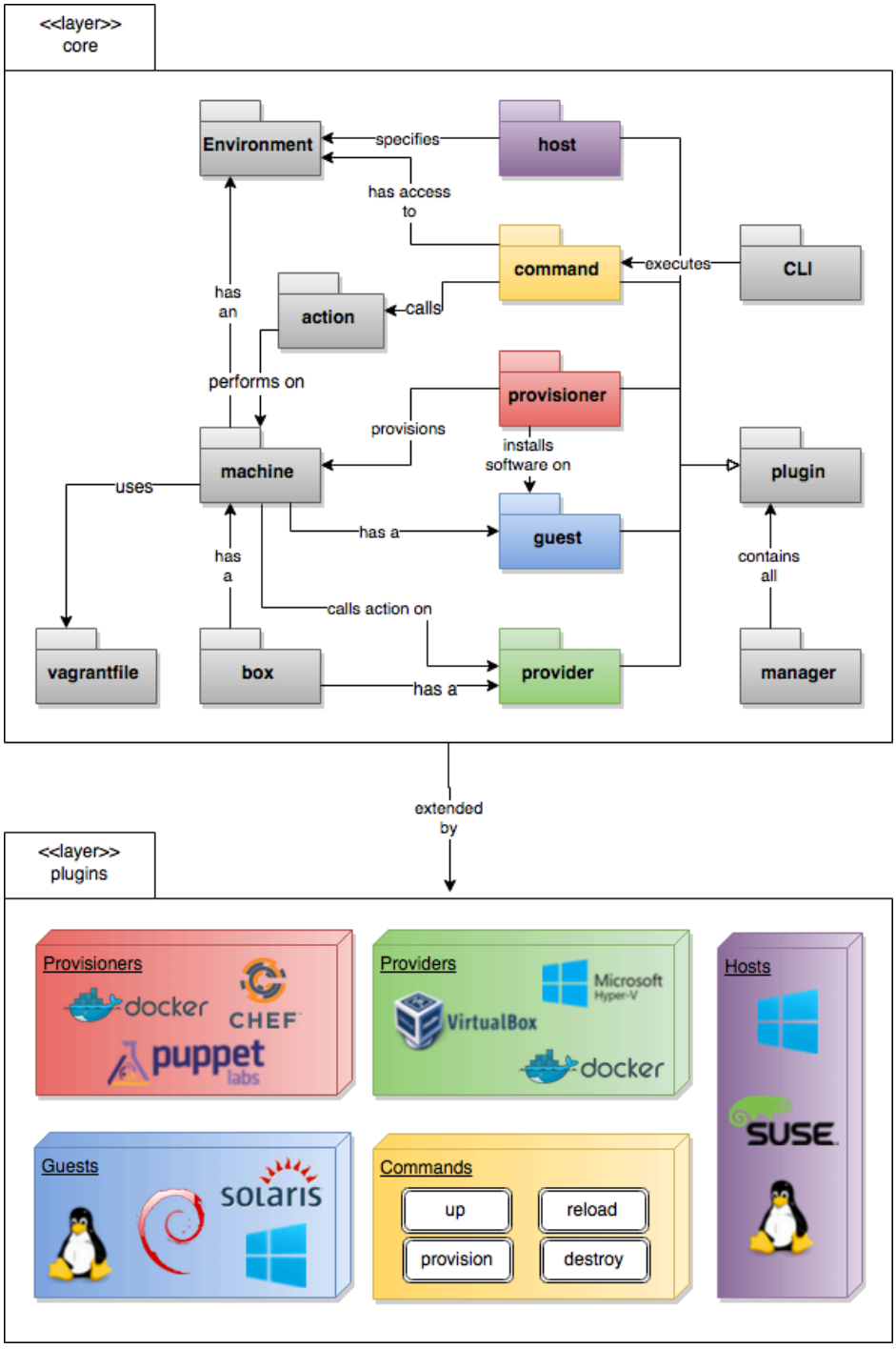


Schéma complet de Vagrant
(source : https://delftswa.github.io/chapters/vagrant/)

Lorsque nous exécutons un script, Vagrant va faire un enchainement de flux pour déployer notre machine virtuelle :

Initialisation : Vagrant va initialiser notre projet à l'aide de la commande `vagrant init`. Cela crée un nouveau `Vagrantfile` dans notre dossier, qui peut être modifié pour définir la configuration de la machine virtuelle. Il a également créé un dossier ".vagrant" contenant les informations relatives à notre provisionnement présent dans notre vagrantfile.

Configuration : Après la première étape, Vagrant va chercher un fichier vagrantfile créé lors de la phase d'initialisation et donc permettre son lancement.

Lancement et accès : En exécutant la commande `vagrant up` , Vagrant crée et lance la machine virtuelle selon la configuration définie dans notre vagrantfile.

Provisionnement : Le provisionnement est exécuté automatiquement lors du lancement de la VM pour installer et configurer les logiciels nécessaires.

Arrêt et Destruction : La VM peut être arrêtée avec la commande `vagrant halt` et redémarrer avec `vagrant reload` puis elle peut être détruite avec la commande : `vagrant destroy <nom de la box>`

Ce flux permet de partager et de reproduire des environnements de développement de manière cohérente et efficace.

6. Installation et Configuration

Comme tout logiciel Vagrant doit au préalable être installé sur un ordinateur. Son installation et configuration est simple à comprendre. Vagrant va utiliser les ressources de notre ordinateur pour y exécuter des scripts avec le langage de programmation Ruby.

6.1 Création et gestion d'environnement virtuel

6.1.1 Installation de Vagrant

Installation sur un système **Linux** (Ubuntu/Debian) :

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
sudo apt-add-repository "deb [arch=amd64]
https://apt.releases.hashicorp.com $(lsb_release -cs) main"
sudo apt update && sudo apt install vagrant
```

Installation sur un système **Windows** :

```
IEX (New-Object
System.Net.Webclient).DownloadString('https://releases.hashicorp.com/vagran
t/2.4.1/vagrant_2.4.1_windows_amd64.msi')
```

Installation sur un système **macOs** :

```
brew tap hashicorp/tap
brew install hashicorp/tap/hashicorp-vagrant
```

6.2 Commandes Vagrant essentielles

Dans Vagrant, on peut classer les commandes en deux grandes catégories, **les commandes de gestion des machines virtuelles** qui sont des commandes qui interagissent directement avec les machines virtuelles qui sont elles-mêmes gérées par vagrant et **les commandes de gestion des box Vagrant** qui sont utilisées pour gérer les box.

6.2.1 Les commandes de gestion des machines virtuelles

vagrant init :La commande va créer un fichier Vagrantfile dans notre répertoire. Ce fichier contient la configuration de la Vagrant Box. On peut utiliser cette commande pour créer un nouveau projet Vagrant.

```
➜ git:(main) ✗ vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

La commande va alors initialiser un fichier vagrantfile. Ce vagrantfile contient toutes les configurations nécessaires pour notre système.

vagrant up : démarre notre Box. Si la box n'est pas encore installée, Vagrant la télécharge et la configure avant de la démarrer.

```
➜ git:(main) ✗ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'StefanScherer/windows_10'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'StefanScherer/windows_10' version
'2021.12.09' is up to date...
==> default: Setting the name of the VM: Windows_Client
```

vagrant ssh : permet de se connecter à notre box Vagrant en ssh.

```
➜ vagrant ssh
Linux LinuxServer 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1
(2024-04-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr 22 14:57:49 2024
vagrant@LinuxServer:~$
```

Comme vus précédemment, nous sommes bien connectés à notre machine virtuelle avec ssh. Cependant pour que la connexion se fasse il nous faut ajouter la clé publique ssh de Vagrant.

vagrant halt : arrête notre box Vagrant.

```
➜ vagrant halt
==> default: Attempting graceful shutdown of VM...
```

vagrant destroy : supprime la Box Vagrant. La commande supprime et arrête notre box.

```
➜ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Destroying VM and associated drives...
```

6.2.2 Les commandes de gestion des box Vagrant

vagrant box add : permet d'ajouter une image de notre machine virtuelle dans une box. Par exemple, pour ajouter une box nommée « Vagrant_Project_M2I/Kali»

```
➜ git:(master) ✗ vagrant box add --name ServerLinux ServerLinuxBox.box
==> box: Box file was not detected as metadata. Adding it directly...
==> box: Adding box 'ServerLinux' (v0) for provider:
    box: Unpacking necessary files from:
file:///home/rvaldebon/Documents/Vagrant/ServerLinuxBox.box
==> box: Successfully added box 'ServerLinux' (v0) for ''!
```

vagrant box list : liste nos box disponibles sur notre pc.

```
➜ git:(master) ✗ vagrant box list
ServerLinux (virtualbox, 0)
```

vagrant box remove : permet de supprimer une Vagrant Box de la liste des box disponibles.

```
➜ vagrant box remove ServerLinux
Removing box 'ServerLinux' (v0) with provider 'virtualbox'...
```

vagrant box update : permet de mettre à jour une Vagrant Box. Si une nouvelle version de la box est disponible, cette commande la téléchargera et la remplacera par la version existante.

Source : https://linux-man.fr/index.php/2021/08/07/vagrant/

7. Provisionnement avec Vagrant

7.1 Explication

Le provisionnement avec **Vagrant** permet d'installer automatiquement des logiciels, de modifier des configurations sur les machines dans le processus de démarrage.

L'approvisionnement de scripts doit se faire à certains moments pendant la durée de vie de l'environnement Vagrant :

- Sur le premier lancement qui crée l'environnement, l'approvisionnement est exécuté. Si l'environnement a déjà été créé et que le lancement ne fait que reprendre une machine ou la démarrer, ils ne fonctionneront pas à moins que l'indicateur `--provision` ne soit explicitement fourni.
- Lorsque la commande de provisionnement (Vagrant provision) est utilisée sur un environnement en cours d'exécution.
- Lorsque l'environnement est relancé (vagrant reload) avec l'indicateur `-no-provision`.

7.2 Provisioning avec d'autres Outils

7.2.1 Chef

Chef Provisioning est une approche différente du SHELL de Vagrant car il regroupe un ensemble de bibliothèques créé par la communauté. Il permet d'intégrer d'autres outils plus facilement.

7.2.2 Ansible

Ansible est conçu pour être simple et efficace, il se distingue par sa capacité à simplifier des processus complexes, rendant l'Automatisation accessible.

Il permet d'apporter un plus pour le provisionnement avec Vagrant. Il se démarque car il est plus complet et permet de faire de la personnalisation plus poussée.

La création de Ansible remonte à 2012, l'outil permet de faciliter la vie des administrateurs système, mais aussi être facilement compris par ceux qui débutent dans le domaine de l'automatisation. Ce qui rend Ansible particulièrement attrayant, c'est sa polyvalence et sa facilité d'utilisation. Que ce soit pour le déploiement d'applications, la gestion de configuration ou l'orchestration de tâches complexes, Ansible offre une solution intuitive.

7.3 Scripts de Provisionnement

7.3.1 Provisionnement commun dans un VagrantFile

Dans un fichier les configurations **config.box** et ses variantes permettent de spécifier et de gérer la "box" Vagrant utilisée. Ces configurations sont essentielles pour personnaliser l'environnement de développement en sélectionnant la machine virtuelle appropriée et en assurant une configuration répondant au besoin de l'utilisateur.

config.vm.box : Spécifie la box que l'on souhaite utilisé soit en local avec nos box soit en cloud avec Vagrant Cloud

config.vm.provider : Ici nous définissons le provider à utiliser. Cela peut aller de Virtualbox, VmWare, etc…

config.vm.network : Permet de définir un réseau propre à notre machine virtuelle. Nous pouvons définir d'utiliser un réseau privé, un réseau NAT ou un réseau dit DHCP en spécifiant le bon réseau.

config.vm.hostname : définie un nom d'hôte

config.vm.provision : Cet ajout permet de définir où Vagrant doit chercher son script de provisionnement. Il peut utiliser le provisionnement SHELL intégré au VagrantFile ou choisir une solution tierce comme Ansible, Puppet ou Chef.

7.3.2 Exemple d'un extrait de script :

```
Vagrant.configure("2") do |config|
  config.vm.box = "generic/debian12"
  config.vm.provider "virtualbox" do |v|
    v.name = "Serveur_DHCP"
  end

  config.vm.provision "shell", inline: <<-SHELL
    apt-get install x11-xkb-utils -y
    echo 'XKBLAYOUT="fr"' >> /etc/default/keyboard
    dpkg-reconfigure keyboard-configuration
    service keyboard-setup restart
    useradd -p $(openssl passwd -1 'M2I') Memoire
  SHELL
end
```

Dans cet exemple de script nous pouvons voir les configurations suivantes :

```
Vagrant.configure("2") do |config|
  config.vm.box = "generic/debian12"
  config.vm.provider "virtualbox" do |v|
    v.name = "Serveur_DHCP"
  end
```

Vagrant.configure("2") do | config : Va initialiser les paramètres de notre machine virtuelle. Le **config.vm.box** récupère notre box donc la machine virtuelle puis le **config.vm.provider** permet de désigner sur quel hyperviseur nous avons lancé notre vm.

```
  config.vm.provision "shell", inline: <<-SHELL
    apt-get install x11-xkb-utils -y
    echo 'XKBLAYOUT="fr"' >> /etc/default/keyboard
    dpkg-reconfigure keyboard-configuration
    service keyboard-setup restart
    useradd -p $(openssl passwd -1 'M2I') Memoire
  SHELL
end
```

Notre config.vm.provision “shell” va exécuter notre provisionning donc exécuter les commandes pour configurer notre machine virtuelle.

8. Sécurité dans Vagrant

La sécurité des systèmes d'information joue un rôle très important pour le bien-être des infrastructures. Vagrant est un outil de création de gestion d'environnements virtuels, donc il joue un rôle important pour la santé des systèmes d'information.

Cette partie va se consacrer sur les possibilités en termes de cybersécurité possible avec l'outil Vagrant et des extensions annexes.

8.1 Sécurité incluse avec Vagrant

8.1.1 Virtualisation et réseau isolé

Vagrant est un outil permettant la virtualisation d'environnement de travail. De ce fait, lorsqu'il va déployer une machine virtuelles, celle-ci est isolée du reste, cette configuration limite l'accès à d'éventuelles attaques externes. Au même titre qu'une machine virtuelle créée manuellement, celle-ci est séparée de notre ordinateur. Également celle-ci peut être configurée avec son propre réseau et donc totalement séparée de notre environnement de travail. Pour notre cas l'utilité de Vagrant sera importante pour démontrer le fonctionnement d'un environnement destiné à l'exploitation de vulnérabilité.

8.1.2 Gestions des fichiers VagrantFile

Les fichiers VagrantFile sont la pièce la plus importante dans Vagrant. Dans ce fichier, toute la configuration de notre environnement virtuel y repose. Le VagrantFile va reposer sur plusieurs critères de sécurité :

1. **Configuration d'un réseau sécurisé** : Il est important de définir un réseau privé pour limiter certains accès, de ce fait cette configuration joue un rôle très important. Le blocage des ports non utilisés peut être défini. Exemple :

```
 boxes = [
        { :name => "Ubuntu_Server", :box => "Vagrant_Project_M2I/Linux",
:ip => "192.168.56.201", :forwarded_port => 2121 }
]
```

   Ici notre serveur Ubuntu a pour IP "**192.168.56.201"** défini dans un réseau privé puis il a une redirection de port vers **2121** pour permettre l'accès SSH.

2. **Gestion des clés SSH** : les clés SSH dans Vagrant sont essentielles pour exécuter notre script. Lors de la phase de déploiement d'une machine virtuelle, il va effectuer une

connexion SSH avec une clé publique. Cette clé est présente sur l'utilisateur local : **/home/vagrant/.ssh/id_rsa**.

3. **Mise à jour automatique** : Lors du lancement du VagrantFile, il est préférable d'exécuter les mises à jour de notre machine virtuelle. Voici un exemple avec la méthode **SHELL** :

```
config.vm.provision "shell", inline <<-SHELL
            apt-get update
            apt-get upgrade -y
      SHELL
end
```

Cette méthode peut parfois être assez longue en fonction des mises à jour.

4. Ajout d'un pare-feu : Il est possible d'ajouter des règles de firewall dans notre provisioning Vagrant avec l'intégration. Exemple :

```
config.vm.provision "shell", inline <<-SHELL
            ufw allow http
            ufw allow 2222
            ufw enable
      SHELL
end
```

8.2 Risques et vulnérabilités connus

Vagrant peut présenter quelques risques de vulnérabilités s'il n'est pas utilisé correctement. Cependant au cours de son existence il n'a jamais eu de failles critiques et les failles les plus courantes sont dues à une mauvaise configuration avec l'utilisation du provisioning et donc l'ajout de fonctionnalités supplémentaires. Voici une liste des CVE connus depuis la création de Vagrant :

CVE-2017-11741	Cette vulnérabilité sortie en 2017 offre la possibilité de faire une escalation de privilège sur VMWare.
CVE-2023-5834	Une vulnérabilité sur l'application Vagrant de Windows permet une élévation de privilèges.

(sources : 1. https://www.exploit-db.com/exploits/43224 2. https://vuldb.com/fr/?id.243780)

Aujourd'hui Vagrant ne possède pas de "réel" vulnérabilités connues. Cependant nous retrouvons beaucoup de mauvaises configurations qui fragilisent les infrastructures.

8.3 Sécurisation avec Ansible

Avec l'intégration d'autres outils, il est possible de sécuriser efficacement notre environnement. Il en existe plusieurs comme mentionné précédemment. Nous pouvons commencer par le plus connu "**Ansible**". Ansible permet de faciliter la configuration de son environnement via des playbooks. Ces playbooks permettent de rajouter une couche de configuration au niveau de la cybersécurité à la configuration. Cela permet d'avoir des templates opérationnels pour corriger d'éventuelles failles. Pour exemple, nous pourrions utiliser un playbook de sécurisation du ssh. Avec Ansible nous pouvons utiliser une configuration comme celle-ci :

```
name: Désactivation du compte root
      lineinfile:
            path: /etc/ssh/sshd_config
            line: 'PermitRootLogin no'
            state: present
      become: true
```

Cette configuration permet d'interdire la connexion ssh avec le compte root. De ce fait, cette option joue un rôle important en limitant les accès. De plus, le paramétrage du ssh pourrait être utilisé comme template.

Pour aller plus loin, il est possible de fixer une limite à la tentative de connexion. Nous pouvons fixer cette limite à 4, au-delà alors l'accès ssh est bloqué. Ce qui donne :

```
name: Limite tentative connexion
      lineinfile:
            path: /etc/ssh/sshd_config
            line: 'MaxAuthTries 4'
            state: present
      become: true
```

---

source : https://sansorg.egnyte.com/dl/9B46RBzgaB

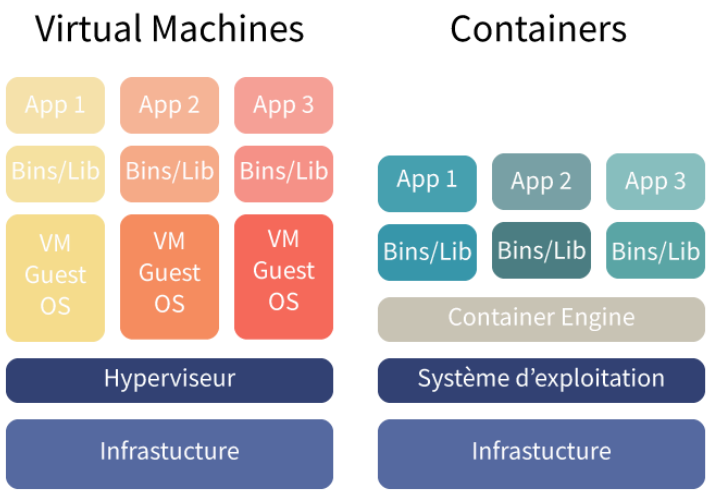
9. Comparaison avec d'autres outils

9.1 Docker

9.1.1 Qu'est ce que Docker ?

Docker permet de concevoir et déployer des applications rapidement sous forme de conteneurs. Les conteneurs sont une unité logicielle qui regroupe toutes les dépendances nécessaires au bon fonctionnement de Docker. Ils permettent l'isolation de notre environnement.Il permet de mettre en œuvre des applications isolées à l'aide d'un fichier dockerfile. Ce dockerfile reprend des configurations similaires à Vagrant.

Différence entre Docker et les machines virtuelles :



(source : https://blog.webnet.fr/introduction-a-docker/)

Ici notre schéma montre la différence entre les deux possibilités. Les machines virtuelles ont besoin d'un hyperviseur pour fonctionner contrairement à Docker qui a besoin d'être seulement installé sur le système d'exploitation. Docker fonctionne sur un principe de conteneur qui est exécuté via des commandes.

9.1.2 Différence avec Vagrant

Vagrant permet de créer des infrastructures de machines virtuelles via des scripts. Le script est exécuté alors Vagrant va récupérer sa box (image d'une machine virtuelle) puis l'importer dans un hyperviseur. Suite à ça il va apporter un provisionnement sur celle-ci pour ajouter les configurations souhaitées.

Docker quant à lui exécute des conteneurs via un dockerfile qui sont constitués de la configuration de notre environnement avec son image et sa configuration. Les conteneurs sont souvent des environnements légers et consomment moins de ressources que des machines virtuelles.

Pour conclure, Docker utilise un format d'image et un environnement d'exécution plus flexible avec son haut taux de personnalisation. Vagrant quant à lui va miser sur la partie infrastructure à taille réelle avec l'importation d'un environnement à travers des hyperviseurs.

9.2 Kubernete

Kubernetes est une plateforme open source pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Il regroupe des conteneurs qui composent une application pour une gestion facile.

Il permet d'avoir moins de paramétrages que Vagrant et est facile à maintenir. Cependant avec l'arrivée de Docker il est devenu obsolète et est plus complexe à configurer qu'un Docker.

9.3 Avantages et Inconvénients de Vagrant

Avantages	Inconvénients
Vagrant dispose d'un seul outil à apprendre	Vagrant peut rencontrer des limites pour l'ajout de configurations complexes. Il est donc utile voir obligatoire d'utiliser des outils comme Ansible, Puppet ou Chef.
Il est possible de créer nos propres machines virtuelles en local ou cloud.	Les machines virtuelles utilisées par Vagrant peuvent être couteuses en ressources, ce qui peut ralentir le système de l'hôte, surtout sur des machines avec des ressources limitées.
La configuration des machines virtuelles est rapide à personnaliser et exécuter via les fichiers vagrantfile avec une meilleure perspective de celle-ci.	Le processus de création et de démarrage des machines virtuelles peut être assez long.
L'harmonisation des environnements de travail.	Les boxes Vagrant sont parfois volumineuses, ce qui peut non seulement poser des problèmes de stockage, mais rendre également les longs téléchargements.
Isolation des machines virtuelles, ce qui offre un bon niveau de sécurité.	Des problèmes de compatibilité peuvent arriver entre Vagrant, le fournisseur de virtualisation, les logiciels de provisionnement et les différentes versions des boxes. Les dépendances entre ces composants peuvent nécessiter des sessions de débogage fréquentes.
La plupart des systèmes d'exploitation sont pris en charge comme Windows ou Linux.	

10. Preuve de concept

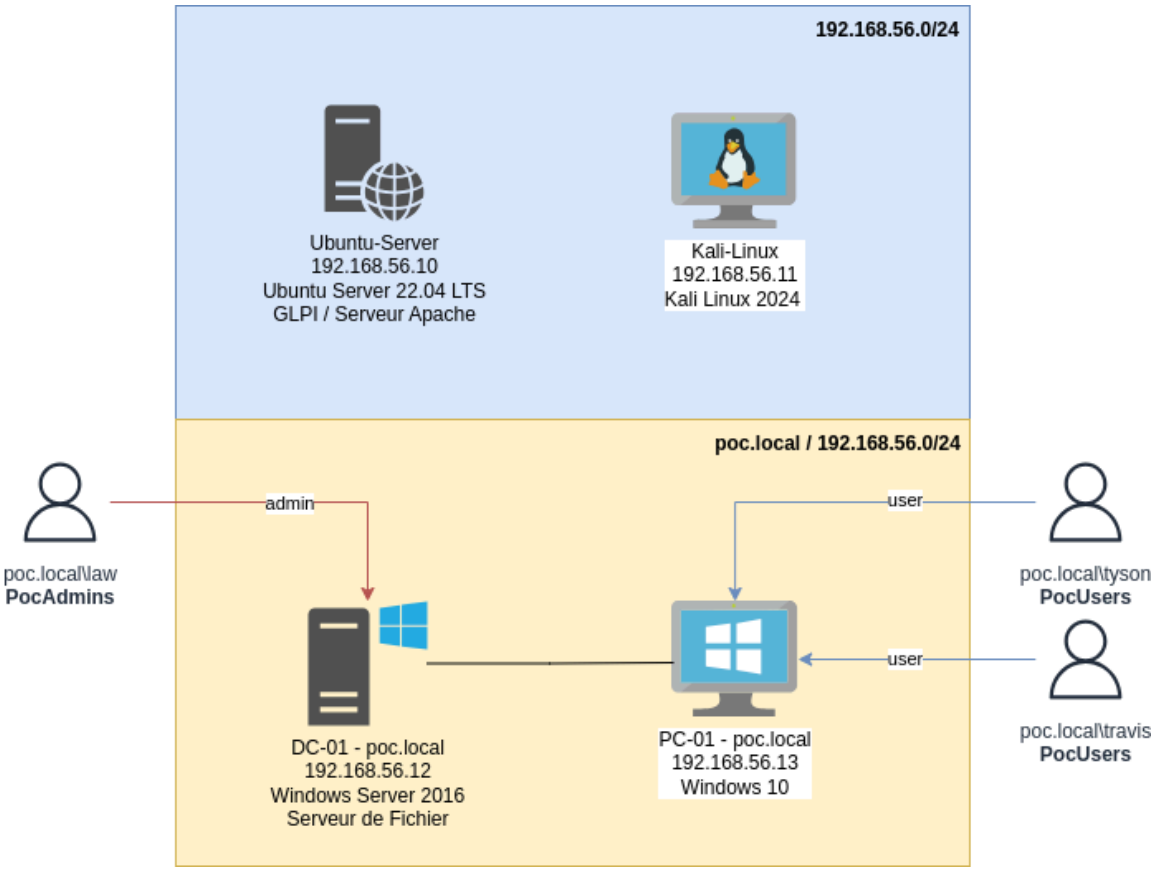
10.1 Phase d'implémentation

Pour faire fonctionner notre environnement Vagrant nous devons faire quelques manipulations.

Vagrant va devoir aller chercher les images qu'on lui impose sur le cloud de Vagrant. Ce principe se fait de la même manière que Docker, lors de la création de notre environnement Vagrant il va alors l'initialiser en prenant notre image sur le cloud. A la différence de Docker est que les images proposées sont plus variées, nous pouvons y trouver des systèmes Linux comme des systèmes Windows avec interface graphique.

Une fois l'image récupérée et la box initialisée, Vagrant crée un fichier nommé vagrantfile. Ce fichier contient notre système d'exploitation ainsi que toute sa configuration pour l'implémentation dans une infrastructure systèmes et réseaux.

Schéma de présentation de preuve de concept :



Notre démonstration démontre l'intérêt de Vagrant au sein d'une infrastructure jeune et nouvelle. Elle propose un environnement consacré à l'automatisation d'une infrastructure destinée aux entreprises. Avec l'ajout d'un Kali Linux, elle permet également de pratiquer un audit de cybersécurité dessus. Nous nous baserons sur une infrastructure composée :

- **Windows Server (192.168.56.13)**
    - Active Directory **poc.local** (3 Utilisateurs, 2 groupes)
    - Serveur de fichier (Shared_Folder)
- **Windows Client (192.168.56.12)**
    - Connecté avec un utilisateur Active Directory
    - Applications installées nativement (Firefox)
- **Serveur Ubuntu (192.168.56.10)**
    - GLPI (fraîchement installé)
    - Proftpd 1.3.5
    - Serveur Apache (permettre l'ajout de site web)
- **Linux Kali (192.168.56.11)**
    - Outils pentester

L'intérêt de cette démonstration est de montrer la force de Vagrant de monter n'importe quelle infrastructure système et réseaux via de simples scripts. Les machines virtuelles seront toutes connectées entre elles.

Ces machines virtuelles seront intégrées sur VirtualBox à l'aide de notre script VagrantFile.

10.2 Organisation de travail

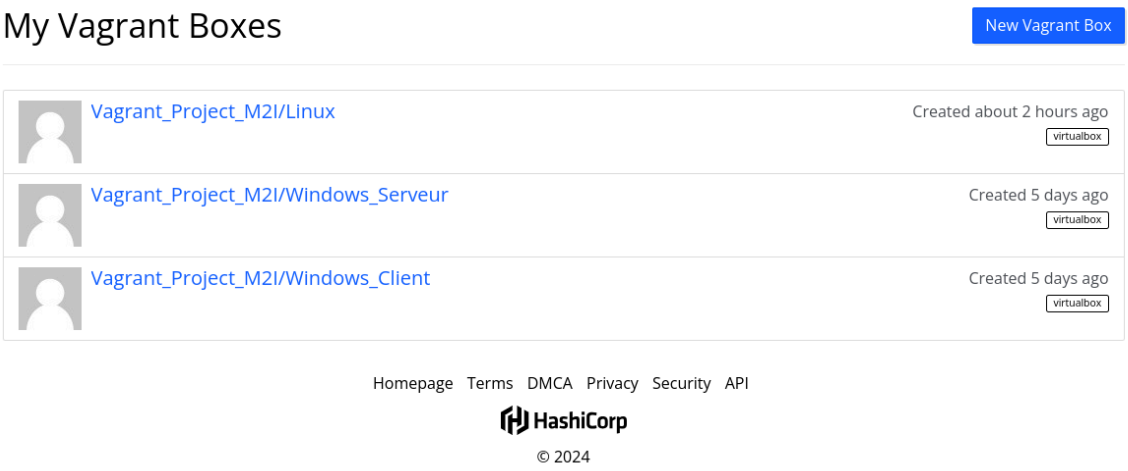
Vagrant permet de déployer des machines virtuelles sous forme de template, c'est donc pour ça qu'on a mis nos fichiers vagrantfile dans le cloud de Vagrant : Vagrant Cloud.

Vagrant Cloud nous permet de modifier directement nos box en ligne et donc de ne pas les perdre. Cet avantage permet de garder une uniformité de notre infrastructure.

10.2.1 Utilisation de Vagrant Cloud

1.Se rendre sur https://app.vagrantup.com/ et se connecter avec son compte. Nous avons chacun notre compte et avons le projet partagé.

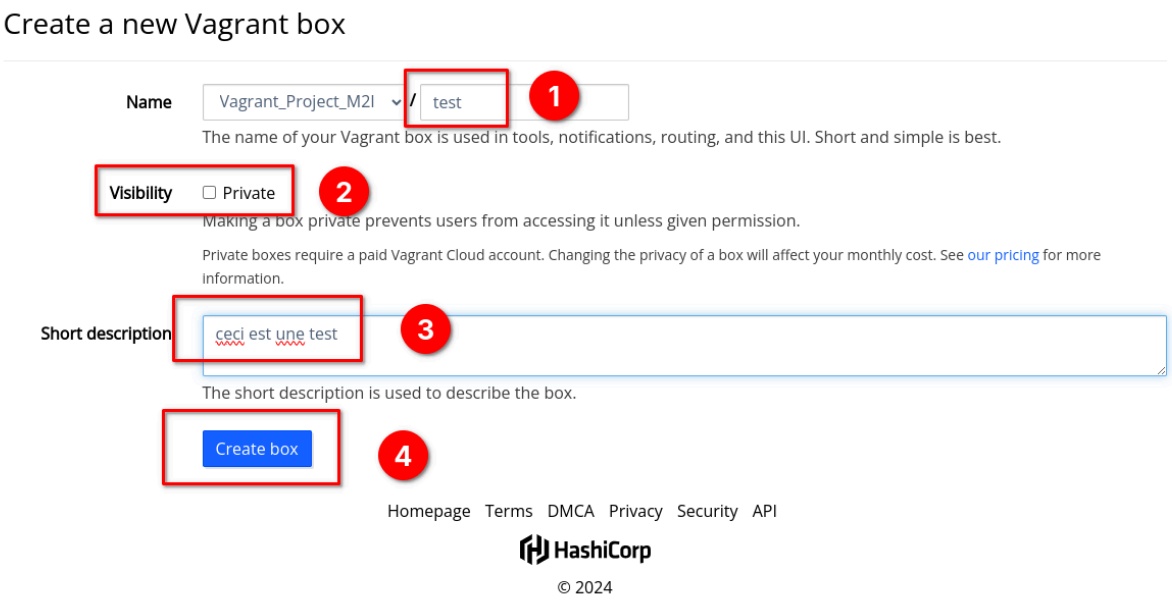
2.Une fois arrivés sur le site, nous pouvons voir notre tableau de bord avec tous nos boxs :



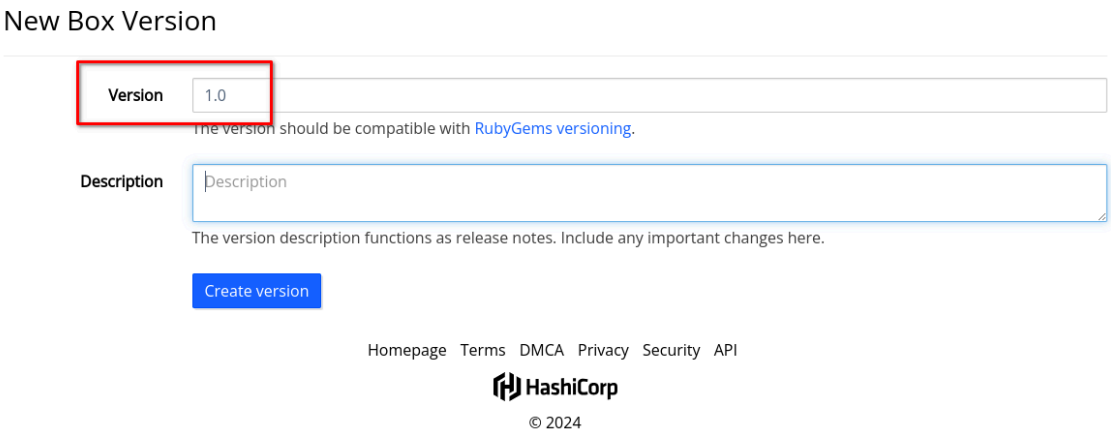
Ici nous possédons 3 boxs. Comme mentionné précédemment ce sont nos boxs pour la montée de notre infrastructure.

3.Pour ajouter une image, il nous suffit de cliquer : “New Vagrant Box”

4.Ensuite, donner un nom à notre box(1), sélectionnez sa visibilité publique(2), une courte description de la box(3) puis cliquez sur “Create Box”(4) :



5. Mettre un numéro de version par défaut et mettre 1.0 :



10.2.2 Utilisation d'un espace GitHub

Une fois les box mises sur notre Cloud de **Vagrant**, il nous faut un environnement collaboratif pour modifier nos fichiers vagrantfile. De ce fait, une fois que nos boxs sont ajoutées sur Vagrant il suffit de cloner le répertoire GitHub et donc de modifier les fichiers. Les fichiers vont contenir le provisionnement et la configuration de nos machines virtuelles.

Notre environnement GitHub se compose de trois dossiers pour nos environnements et également un script bash pour les lancers un à un. Pour l'instant les fichiers vagrantfile sont séparés mais à terme ils seront en un seul fichier.

Lien vers le répertoire GitHub : https://github.com/Rvaldebon/Vagrant_Project/

10.3 Déploiement de nos systèmes

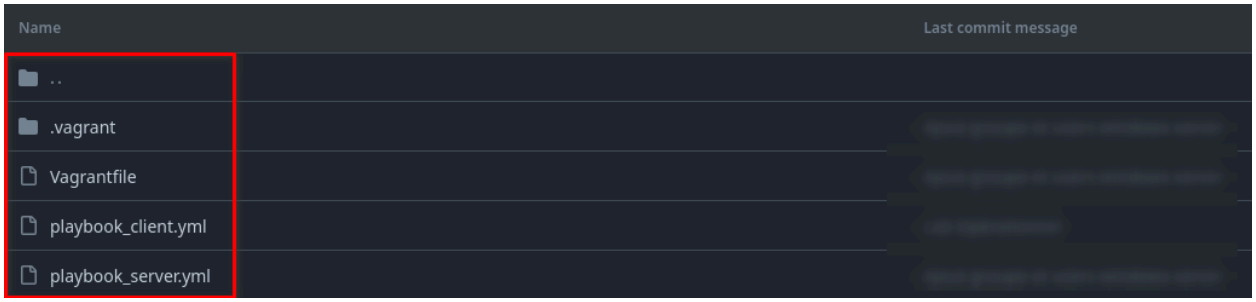
Pour le déploiement de nos systèmes, nous allons utiliser des fichiers Vagrantfile. Le fichier vagrantfile comprend toutes nos machines virtuelles. A l'aide de la commande vagrant up nous pourrons déployer notre infrastructure. Nous allons suivre un ordre précis pour mener au mieux notre preuve de concept. Cette partie doit durer au maximum 40 minutes avec préparation et démonstration incluses.

1. **Préparation de notre environnement (15 minutes)**

Tout d'abord nous allons récupérer nos box qui contiennent nos machines virtuelles qui comprennent :

- WIndows Server
- Windows Client
- Ubuntu Server
- Kali Linux

Notre fichier de configuration se trouve sur un repo GitHub contenant notre infrastructure.



Ici nous avons notre fichier Vagrantfile contenant toute l'infrastructure. Le fichier se trouve à l'annexe 14.2. Également les 2 playbooks requis pour la configuration d'un active directory et ajout d'un client au domaine (annexe 14.7 et 14.8). Les images seront téléchargées et préparées depuis notre Vagrant Cloud.

2. Lancement de notre laboratoire (5 minutes)

Lors de la vérification effectuée précédemment, il nous faut lancer notre laboratoire avec la commande “vagrant up”. La commande va alors initialiser notre infrastructure et commencer son déploiement. Vagrant va alors passer sur chaque machine pour vérifier qu’elles sont bien importées et si le provisionnement est bien effectué.

3. Provisionnement de nos systèmes (5 minutes)

Une fois le fichier Vagrantfile de configuré, nous allons provisionner celui-ci. Cela va permettre d’appliquer notre configuration sur les machines virtuelles. Ce provisionnement va se faire à l’aide des scripts **SHELL** présent par défaut dans Vagrantfile (exemple évoqué dans la partie 7.3.2) ou par des fichiers externes avec Ansible (exemple dans les annexes 14.7 et 14.8).

Lors de notre phase d’implémentation, nous avons deux possibilités, soit la création de l’infrastructure via un script “shell” qui va lire chacun de nos fichiers Vagrantfile un à un :

Le fichier Vagrantfile pour le Windows Server se trouve à l’annexe **14.3**

Le fichier Vagrantfile pour le Windows Client se trouve à l’annexe **14.4**

Le fichier Vagrantfile pour l’Ubuntu Server se trouve à l’annexe **14.5**

Le fichier Vagrantfile pour la Kali Linux se trouve à l’annexe **14.6**

Cela engendre beaucoup de fichiers pour pas grand-chose mais nous permet de mieux comprendre le fonctionnement de Vagrant car ils sont plus lisibles.

La seconde solution choisie est l’utilisation d’un seul script évoqué précédemment. Les fichiers se trouvent à l’annexe : **14.2**, **14.7**, **14.8**.

Extrait du provisionnement :

```
TASK [Ajout du DNS] ************************************************************
changed: [Windows-10]

TASK [Promouvoir le client] ****************************************************
changed: [Windows-10]

TASK [Afficher le résultat de l'ajout au domaine] ******************************
ok: [Windows-10] => {
    "domain_state": {
        "changed": true,
        "failed": false,
        "reboot_required": true
    }
}

TASK [Reboot après ajout au domaine] *******************************************
changed: [Windows-10]

PLAY RECAP *********************************************************************
Windows-10                 : ok=4    changed=3    unreachable=0    failed=0    skipped=0
```

Dans l'extrait de notre provisionnement, Vagrant et l'approvisionneur Ansible initient plusieurs étapes :

1. Ajout du DNS de notre machine Windows 10 pour permettre l'ajout au Windows Server
2. Ajout de l'active directory et promouvoir ce dernier
3. Redémarrage de la machine virtuelle pour appliquer les changements de WORKGROUP →poc.local

4. Démonstration du fonctionnement du lab (15 minutes)

Une fois l'infrastructure opérationnelle alors nous pouvons faire une démonstration du bon fonctionnement de celle-ci avec plusieurs étapes :

1. Montrer que notre environnement est bien configuré avec nos paramètres évoqués dans les fichiers **Vagrantfile** et **Ansible**.
2. Démontrer que la machine Windows est bien rentrée dans le domaine.
3. Le dossier partagé est accessible.
4. Utilisation d'une Kali Linux pour démontrer qu'il est possible de réaliser de simples attaques sur celle-ci.

6. Débat avec le jury

Nous serons ouverts à l'explication plus approfondie de notre environnement et répondre aux éventuelles questions. Également conclure la présentation par une analyse critique de Vagrant.

11. Conclusion

Pour conclure, Vagrant est une solution utile pour le monde de l'entreprise et permet de déployer toute une infrastructure à l'aide d'une commande : "vagrant up".

Ce mémoire à pu nous permettre d'exploiter l'application et sa capacité dans le domaine de la cybersécurité, ainsi de montrer comment un tel outil peut standardiser et simplifier la gestion d'un environnement.

L'utilisation de ce dernier répond à un fort besoin dans le monde du devops, il permet d'avoir une cohérence entre les environnements de développements et de productions. Il intègre également beaucoup d'outils tiers pour faciliter son déploiement.

Vagrant possède également quelques défauts. Il est assez complexe pour la configuration d'environnements complexes. Seulement les bénéfices apportés sont colossaux.

En conclusion Vagrant présente une solution fiable et innovante pour le monde de l'informatique.

12. Sources

1. https://kinsta.com/fr/blog/vagrant-vs-docker/#:~:text=Docker%20permet%20aux%20%C3%A9quipes%20DevOps,de%20mani%C3%A8re%20diff%C3%A9rente%20mais%20compl%C3%A9mentaire. consulté le : 02/01/2024, Auteur : Salman Ravoof
2. https://blog.stephane-robert.info/docs/infra-as-code/provisionnement/vagrant/introduction/ consulté le : 05/01/2024, Auteur : Stephane Robert
3. https://www.vagrantup.com/ consulté le : 06/01/2024, Auteur : Vagrant
4. https://gist.github.com/turtlemonvh/2f40fa7ca0acaaabaf6a6ecb0ee556c7 consulté le : 08/01/2024, Auteur : turtlemonvh
5. https://connect.ed-diamond.com/linux-pratique/lp-127/deployer-des-environnements-de-developpement-avec-vagrant consulté le : 20/01/2024, Auteur : Julien Morot
6. https://developer.hashicorp.com/vagrant/docs/networking consulté le : 02/02/2024, Auteur : Vagrant
7. https://github.com/hashicorp/vagrant-spec consulté le : 02/02/2024, Auteur : Vagrant
8. https://hal.science/hal-03192628/document consulté le : 02/02/2024, Auteurs : Loic Houde, Daniel Jacob, Tovo Rabemanantsoa, Jean-François Rey
9. https://www.it-connect.fr/vmware-workstation-creer-une-vm-windows-server-avec-packer-et-vagrant/ consulté le : 06/02/2024, Auteur : Florian Burnel
10. https://developer.hashicorp.com/vagrant/docs/vagrantfile consulté le : 06/02/2024, Auteur : Vagrant
11. https://solydev.net/deployez-de-la-debian-avec-vagrant-en-creant-vos-propres-box/ consulté le : 06/02/2024, Auteur : Sébastien Reuiller
12. http://tvaira.free.fr/bts-sn/admin/Creation-Box-Vagrant.pdf consulté le : 19/02/2024, Auteur : Tvaira
13. https://github.com/cosad3s/CVE-2022-35914-poc/tree/main consulté le : 20/02/2024, Auteur : cosad3s
14. https://fr.wikipedia.org/wiki/Vagrant consulté le : 22/02/2024, Auteur : Wikipédia
15. https://github.com/hashicorp/vagrant consulté le : 10/03/2024, Auteur : Hashicorp
16. https://www.youtube.com/watch?v=YpkPAb7q10k consulté le : 20/03/2024, Auteur :Grafikart.fr
17. https://www.ypsi.fr/tuto-deploiement-de-vos-machines-virtuelles-avec-vagrant/#:~:text=Vagrant%20est%20un%20logiciel%20open,en%20place%20d'un%20projet. consulté le : 21/03/2024, Auteur : Quentin Bassemayousse
18. https://luxagraf.net/src/create-custom-debian-9-vagrant-box consulté le : 01/04/2024, Auteur : Scott Gilbertson
19. https://github.com/kraksoft/vagrant-box consulté le : 01/04/2024, Auteur : kraksoft
20. https://tferdinand.net/creer-des-boxes-vagrant-facilement-en-utilisant-packer/ consulté le : 02/04/2024, Auteur : Teddy FERDINAND

21. https://www.it-connect.fr/deployer-un-domaine-active-directory-avec-vagrant-et-ansible/ consulté le : 25/04/2024, Auteur : Geoffrey Sauvageot-Berland
22. https://blent.ai/blog/a/vagrant-automatisation-machines-virtuelles consulté le : 26/04/2024, Auteur : Maxime Jumelle
23. https://aws.amazon.com/fr/docker/ consulté le : 15/05/2024, Auteur : Amazon
24. https://kinsta.com/fr/blog/vagrant-vs-docker/#:~:text=Docker%20permet%20aux%20%C3%A9quipes%20DevOps,de%20mani%C3%A8re%20diff%C3%A9rente%20mais%20compl%C3%A9mentaire. consulté le : 15/05/2024, Auteur : Salman Ravoof
25. https://docs.ansible.com/ansible/latest/collections/community/windows/win_security_policy_module.html consulté le : 29/05/2024, Auteur : Ansible
26. https://www.it-connect.fr/chapitres/creation-dun-partage-smb-en-powershell/#:~:text=I.-,Cr%C3%A9er%20un%20dossier%20pour%20le%20partage%20avec%20PowerShell,(Directory%20pour%20un%20dossier).&text=Le%20dossier%20%C3%A9tant%20cr%C3%A9%C3%A9%2C%20nous%20allons%20pouvoir%20le%20partager. consulté le : 29/05/2024, Auteur : Florian BURNEL

13. Glossaire

1. **Machine virtuelle** : C'est le principe de virtualiser un appareil informatique à l'aide d'un hyperviseur.
2. **Hyperviseur** : Un hyperviseur permet de gérer notre parc de machine virtuelle
3. **VM** (Virtual Machine): Machine Virtuelle
4. **Packer** : est un outil créé par HashiCorp et permettant de créer des images de machine identiques pour plusieurs plateformes à partir d'une configuration source unique (Iso). Dans notre exemple, il va nous servir à créer une Base Box à partir d'une image Windows, Linux ou autres. Puis par la suite il passe le relai à Vagrant qui va créer une nouvelle machine virtuelle basée sur la Base Box, cette dernière étant utilisée comme source pour déployer la nouvelle VM (sans altérer la box qui est statique).
5. **Vagrant Cloud** : sert d'index public et permet aux utilisateurs de consulter des BaseBox déjà existantes (Windows Serveur, Ubuntu, Kali Linux, Windows 7-8-10-11 etc) et de les utiliser dans leurs projets.
6. **YAML** : est un langage utilisé pour la configuration d'infrastructure.
7. **Wrapper :** qui enveloppe plusieurs logiciels.
8. **Provisionneur** est un outil utilisé pour automatiser le processus de configuration des environnements et des applications sur des machines physiques ou virtuelles.
9. **GitHub** : Github est une solution pour stocker des répertoires destinés à l'utilisation communautaire de code informatique.
10. **Ruby** : est un langage de programmation notamment utilisée avec Vagrant/
11. **Système d'exploitation** : Un système d'exploitation est un ensemble de programmes qui permettent d'utiliser les ressources matérielles d'un ordinateur.
12. **Socket virtuel** : Un socket virtuel reproduit le fonctionnement d'un processeur physique pour virtualiser un processeur virtuel.
13. **ssh** : Secure Shell de son vrai nom, est un programme qui permet d'établir une connexion sécurisée entre plusieurs machines.

14. Annexes

14.1 Configuration d'une box Vagrant Linux :

Sur les machines Linux, il nous faut ajouter la clé ssh officielle de Vagrant. Cette clé permettra de prendre la main en ssh à nos machines via la commande vagrant ssh. Cela change comparé à une machine Windows qui n'a pas besoin de configurations spécifiques.

```
$ sudo apt-get install linux-headers-$(uname -r) build-essential dkms
$ sudo mount /dev/cdrom /media/cdrom
$ sudo sh /media/cdrom/VBoxLinuxAdditions.run
$ useradd -m -s /bin/bash vagrant
$ passwd vagrant
# Mettre vagrant en mot de passe
$ mkdir /home/vagrant/.ssh
$ wget -O /home/vagrant/.ssh/authorized_keys
https://github.com/hashicorp/vagrant/raw/master/keys/vagrant.pub
$ chown -R vagrant:vagrant /home/vagrant/.ssh
$ chmod 700 /home/vagrant/.ssh
$ chmod 600 /home/vagrant/.ssh/authorized_keys
```

14.2 Fichier Vagrantfile

```
Vagrant.configure("2") do |config|

    ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'

    boxes = [
        { :name => "Ubuntu-Server", :ip => "192.168.56.10", :box =>
"Vagrant_Project_M2I/Linux", :memory => "6024", :cpus => 2, :usb => "off",
:os => "linux", :forwarded_port => { :guest => 21, :host => 2121 } },
        { :name => "Kali", :ip => "192.168.56.11", :box =>
"kalilinux/rolling", :memory => "6024", :cpus => 2, :usb => "off", :os =>
"linux" },
        { :name => "Windows-Server", :ip => "192.168.56.12", :box =>
"peru/windows-server-2016-standard-x64-eval", :memory => "6024", :cpus =>
4, :usb => "off", :os => "windows" },
        { :name => "Windows-10", :ip => "192.168.56.13", :box =>
"gusztavvargadr/windows-10", :memory => "6024", :cpus => 4, :usb => "off",
:os => "windows" }
    ]

    boxes.each do |box|
        config.vm.define box[:name] do |machine|
            machine.vm.box = box[:box]
            machine.vm.network "private_network", ip: box[:ip]
        config.vm.provider "virtualbox" do |v|
            v.memory = box[:memory]
            v.cpus = box[:cpus]
            v.customize ["modifyvm", :id, "--usb", box[:usb]]
        end

            if box[:forwarded_port]
                machine.vm.network "forwarded_port", guest:
box[:forwarded_port][:guest], host: box[:forwarded_port][:host]
            end
        end
    end

    config.vm.define "Ubuntu-Server" do |ubuntu|
        ubuntu.vm.provision "shell", inline: <<-SHELL
            apt-get update
            apt-get upgrade -y
            apt-get install -y apache2 mariadb-server php
```

```
libapache2-mod-php php-mysql
            apt-get install -y php-mbstring php-curl php-gd php-simplexml
php-intl php-ldap php-xmlrpc php-zip php-bz2 php-apcu
            mysql -u root -e "CREATE DATABASE glpi;"
            mysql -u root -e "CREATE USER 'poc'@'localhost' IDENTIFIED BY
'pocL0cal!';"
            mysql -u root -e "GRANT ALL PRIVILEGES ON glpi.* TO
'poc'@'localhost';"
            mysql -u root -e "FLUSH PRIVILEGES;"
            wget
https://github.com/glpi-project/glpi/releases/download/9.5.5/glpi-9.5.5.tgz
            tar -xvzf glpi-9.5.5.tgz -C /var/www/html/
            chown -R www-data:www-data /var/www/html/glpi
            chmod -R 755 /var/www/html/glpi
            echo "<VirtualHost *:80>
                DocumentRoot /var/www/html/glpi
                <Directory /var/www/html/glpi>
                    AllowOverride All
                    Require all granted
                </Directory>
            </VirtualHost>" > /etc/apache2/sites-available/glpi.conf
            a2ensite glpi.conf
            a2dissite 000-default.conf
            service apache2 restart
            #Activation de GLPI
            phpenmod mbstring
            phpenmod curl
            phpenmod gd
            phpenmod simplexml
            phpenmod intl
            phpenmod ldap
            phpenmod xmlrpc
            phpenmod zip
            phpenmod bz2
            phpenmod apcu
            phpenmod cas
            # Installation de proftpd
            wget ftp://ftp.proftpd.org/distrib/source/proftpd-1.3.5.tar.gz
            tar xfz proftpd-1.3.5.tar.gz
            cd proftpd-1.3.5
            sudo ./configure --with-modules=mod_copy
            make && sudo make install
            sudo /usr/local/sbin/proftpd
        SHELL
```

```
    end

    config.vm.define "Windows-Server" do |winsrv|
        winsrv.vm.provision "shell", inline: <<-SHELL
            Set-WinUserLanguageList -LanguageList fr-FR -Force
            New-Item -ItemType Directory -Path "C:\\Shared_Folder"
            New-SmbShare -Name "Shared_Folder" -Path "C:\\Shared_Folder"
-FullAccess "Everyone"
        SHELL
        winsrv.vm.provision "ansible" do |ansible|
            ansible.playbook = "playbook_server.yml"
            ansible.compatibility_mode = "2.0"
        end
    end

    config.vm.define "Windows-10" do |win10|
        win10.vm.provision "ansible" do |ansible|
            ansible.playbook = "playbook_client.yml"
            ansible.compatibility_mode = "2.0"
        end
    end
end
```

14.3 Vagrantfile Windows Server

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "peru/windows-server-2016-standard-x64-eval"
  config.vm.network "private_network", ip: "192.168.56.12"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "6024"
    vb.cpus = 4
  end

  config.vm.provision "shell", inline: <<-SHELL
    powershell -Command "Set-WinUserLanguageList -LanguageList fr-FR
-Force"
  SHELL

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbook.yml"
    ansible.compatibility_mode = "2.0"
  end
end
```

14.4 Vagrantfile Windows Client

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "gusztavvargadr/windows-10"
  config.vm.network "private_network", ip: "192.168.56.13"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "6024"
    vb.cpus = 4
  end

  config.vm.provision "shell", inline: <<-SHELL
    powershell -Command "Set-WinUserLanguageList -LanguageList fr-FR
-Force"
  SHELL

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbook.yml"
    ansible.compatibility_mode = "2.0"
  end
end
```

14.5 Vagrantfile Ubuntu Server

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.define "Ubuntu_Server" do |ubuntu|
    ubuntu.vm.box = "Vagrant_Project_M2I/Linux"
    ubuntu.vm.network "private_network", ip: "192.168.56.10"
    ubuntu.vm.network "forwarded_port", guest: 21, host: 2121
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
    vb.cpus = 2
    vb.customize ["modifyvm", :id, "--usb", "off"]
    end
  end

  config.vm.provision "shell", inline: <<-SHELL
  # Installation des prérequis
    apt-get update
    apt-get upgrade -y
    apt-get install -y apache2 mariadb-server php libapache2-mod-php
php-mysql
    apt-get install -y php-mbstring php-curl php-gd php-simplexml php-intl
php-ldap php-xmlrpc php-zip php-bz2 php-apcu
    # Configuration de la base de donées GLPI
    mysql -u root -e "CREATE DATABASE glpi;"
    mysql -u root -e "CREATE USER 'glpi'@'localhost' IDENTIFIED BY
'password';"
    mysql -u root -e "GRANT ALL PRIVILEGES ON glpi.* TO
'glpi'@'localhost';"
    mysql -u root -e "FLUSH PRIVILEGES;"
    # Télécharement de GLPI
    wget
https://github.com/glpi-project/glpi/releases/download/9.5.5/glpi-9.5.5.tgz
    tar -xvzf glpi-9.5.5.tgz -C /var/www/html/
    chown -R www-data:www-data /var/www/html/glpi
    chmod -R 755 /var/www/html/glpi
    # Configuration d'apache
    echo "<VirtualHost *:80>
        DocumentRoot /var/www/html/glpi
        <Directory /var/www/html/glpi>
            AllowOverride All
```

```
            Require all granted
        </Directory>
    </VirtualHost>" > /etc/apache2/sites-available/glpi.conf
    a2ensite glpi.conf
    a2dissite 000-default.conf
    service apache2 restart
    #Activation de GLPI
    phpenmod mbstring
    phpenmod curl
    phpenmod gd
    phpenmod simplexml
    phpenmod intl
    phpenmod ldap
    phpenmod xmlrpc
    phpenmod zip
    phpenmod bz2
    phpenmod apcu
    phpenmod cas
    # Installation de proftpd
    wget ftp://ftp.proftpd.org/distrib/source/proftpd-1.3.5.tar.gz
    tar xfz proftpd-1.3.5.tar.gz
    cd proftpd-1.3.5
    sudo ./configure --with-modules=mod_copy
    make && sudo make install
    sudo /usr/local/sbin/proftpd
    # Ajout du flag
    echo "NHM2I{glpi}" > /home/vagrant/.notes
  SHELL
end
```

14.6 Vagrantfile Kali Linux

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "Vagrant_Project_M2I/Kali"
  config.ssh.username = "vagrant"
  config.ssh.password = "vagrant"
  config.ssh.insert_key = false
  config.vm.network "private_network", ip: "192.168.56.11"

  config.vm.provision "shell", inline: <<-SHELL
    # Ajout payload GLPI
    cd /home/vagrant/Documents
    git clone https://github.com/cosad3s/CVE-2022-35914-poc.git
    cd CVE-2022-35914-poc
    pip install -r requirements.txt
  SHELL
end
```

14.7 Playbook Ansible Windows Server

```
- name: Config Active directory
  hosts: all
  tasks:

  - name: Run powershell commands to prepare the DC
    ansible.windows.win_powershell:
      script: |
        # Your PowerShell script here

- name: Configure role(s) on a Windows server instance
  hosts: all
  tasks:

  - name: Modification du clavier en français
    ansible.windows.win_powershell:
      script: |
        Set-WinUserLanguageList fr-FR -Force

  - name: Ajout du rôle AD-DS
    ansible.windows.win_powershell:
      script: |
        Add-WindowsFeature AD-Domain-Services
        Set-TimeZone -Name "Romance Standard Time"

  - name: Ajout du nom d'hôte
    ansible.windows.win_hostname:
      name: DC-01
    register: hostname

  - name: Reboot
    ansible.windows.win_reboot:
    when: hostname.reboot_required

  - name: Installation du domaine
    ansible.windows.win_domain:
      dns_domain_name: poc.local
      safe_mode_password: Vagrant!2024
    register: domain_install

  - name: Reboot after promotion
    ansible.windows.win_reboot:
```

```
    when: domain_install.reboot_required

  - name: Création du goupe PocAdmin
    ansible.windows.win_group:
      name: PocAdmin
      description: PocAdmin Users
      state: present

  - name: Création du goupe PocUtilisateur
    ansible.windows.win_group:
      name: PocUsers
      description: Poc users
      state: present

- name: Ajout des utilisateurs au domaine
  hosts: all
  tasks:
    - name: Ajouter l'utilisateur "law" au domaine
      win_domain_user:
        domain_username: "administrator"
        domain_password: "vagrant"
        name: "law"
        password: "azerty123!"
        state: present
        domain_server: "poc.local"
    - name: Ajout de l'utilisateur "travis" au domaine
      win_domain_user:
        name: "travis"
        password: "3CAn5F@2024"
        state: present
        domain_server: "poc.local"
    - name: Ajout de l'utilisateur "tyson" au domaine
      win_domain_user:
        name: "tyson"
        password: "M&UYNr2024!"
        state: present
        domain_server: "poc.local"
    - name: Add users to PocUsers group
      win_shell: |
        Add-ADGroupMember -Identity "PocUsers" -Members "travis"
        Add-ADGroupMember -Identity "PocUsers" -Members "tyson"
        Add-ADGroupMember -Identity "PocAdmin" -Members "law"
```

14.8 Playbook Ansible Windows Client

```
---
- name: Ajout du client au domaine
  hosts: all
  gather_facts: false
  vars:
    ansible_winrm_server_cert_validation: 'ignore'
    ansible_connection: 'winrm'
    ansible_port: '55985'
    ansible_winrm_transport: 'ntlm'
    ansible_winrm_scheme: 'http'
    domain_admin_user: "administrator"
    domain_admin_password: "vagrant"

  tasks:

  - name: Ajout du DNS
    win_dns_client:
      adapter_names: '*'
      ipv4_addresses: '192.168.56.12'
    register: dns_result

  - name: Promouvoir le client
    win_domain_membership:
      dns_domain_name: 'poc.local'
      domain_admin_user: 'Administrator@poc.local'
      domain_admin_password: 'vagrant'
      state: domain
    register: domain_state

  - name: Afficher le résultat de l'ajout au domaine
    debug:
      var: domain_state

  - name: Reboot après ajout au domaine
    win_reboot:
      msg: "Rebooting to complete domain join"
    when: domain_state.reboot_required
```